EOMS 1.3

EOMS

Herzlich Willkommen auf der Online-Dokumentationsseite des EOMS!

Hier finden Sie alle Informationen zur aktuellen Version 1.3 des EOMS (**E**nterprise **O**utput **M**anagement **S**ystem).

Erste Schritte

- Das EOMS

Online-Archiv und Online-Dokumentationen

- Online-Archiv des EOMS
- Ergänzende Online-Dokumentationen

Endbenutzer

- Zur Nutzer-Dokumentation
- RCML lernen

EOMS-Administrator

- Zur EOMS-Admin Dokumentation
- Installation des EOMS

Erweiterte Informationen

- Rechtliche Hinweise
- Service und Support
- Sitemap
- Download der Dokumentation

Sie suchen Hilfe für ein anderes Produkt oder verwenden eine ältere Version des EOMS? Hier gelangen Sie zur Übersicht aller Online-Dokumentationen!

Was ist neu in Version 1.3

In der Version 1.3 wurden im Vergleich zur Version 1.2 folgende wesentliche Änderungen/Erweiterungen am EOMS vorgenommen:

- ***Änderungen im EOMS-Core:***

  - Die EOMS-Oberfläche ist jetzt auch in Deutscher Sprache verfügbar.
  - Neben Systeminformationen werden jetzt auch explizit Clientinformationen in der "Über"-Seite angezeigt.
  - Ressource-Fetcher eines Workers werden in der Worker-Ansicht angezeigt.
  - Verbesserte Filtern-/Suchen-Funktion in der Worker-/Prozess-Ansicht.
  - Callback-Dispatcher für eine schnellere Interaktion zwischen EOMS und Spooler.
  - Es wurden einige neue Konfigurationsdirektiven für den Core eingeführt, siehe dazu die Konfigurationsseite für den Core.

- ***Änderungen im EOMS-Worker:***

  - Für die einzelnen Konsumenten eines Workers kann genau angegeben werden, welche Jobs diese verarbeiten sollen.
  - Es wurden einige neue Konfigurationsdirektiven für den Worker eingeführt, siehe dazu die Konfigurationsseite für Worker.
  - Die Verzeichnisstruktur des EOMS-Workers hat sich geringfügig geändert.

- ***Technische Hinweise speziell für Administratoren zu Änderungen und Migrationshinv***

  - EOMS 1.3 verwendet Java 8 und Tomcat 8. Es muss deshalb ein JRE/JDK in der Version 1.8 installiert sein.
  - Passwörter werden jetzt in der eoms.invoker.authentication als PBKDF2-Hashes gespeichert.
  - Im Internet Explorer werden Grafiken jetzt richtig angezeigt.

Beachten Sie auch die Liste der Änderungen in Version 1.2.

Einfuehrung

Auf den Seiten dieser Dokumentation möchten wir Sie beim Einsatz des EOMS unterstützen. In der Nutzer-Dokumentation wird vor allem auf die Benutzung des EOMS (Core und Worker) sowie deren Konfiguration, insbesondere auf die Workerkonfiguration durch die Rich Client Markup Language (RCML Kompendium), eingegangen. Die Administrator-Konfiguration hingegen unterstützt Sie bei Installation, Sicherheit und der Anbindung an die Auftrags-Systeme.

Was ist das EOMS ?

Das EOMS ist ein verteiltes System zur Lastenverteilung von Prozessen aus externen Systemen ("Auftrags-Systeme") auf beliebig viele Recheneinheiten mit Unterstützung für Multicore-Prozessoren.

Mit Hilfe des EOMS lassen sich unterschiedlichste Prozesse auf beliebig viele Workstations (Worker) aufteilen. Neben reiner Datenverarbeitung unterstützt das EOMS u.a. auch die Interaktion mit dem EOMS-Input-Service und eignet sich damit optimal zur Bewältigung großer Datenmengen für docxworld.

Das EOMS ist voll kompatibel zum Redaktions-System und bietet damit die ideale Grundlage für die Weiterverarbeitung der Daten aus dem R-S.

Wie funktioniert das EOMS ?

Das EOMS besteht aus 2 Komponenten: Dem EOMS-Core sowie den EOMS-Workern. Die Komponenten des EOMS sind untereinander und mit den angeschlossenen Auftrags-Systemen für maximale Sicherheit nur lose gekoppelt. Das EOMS-Core empfängt Jobs vom Auftrags-System und vergibt diese an die Worker. Dort können die Jobdaten direkt aus dem Auftrags-System extrahiert und gemäß dem vordefinierten Prozess abgearbeitet werden. Die fertigen Jobdaten können dann entweder direkt zurück zum Auftrags-System, zum EOMS-Input oder an andere Systeme geschickt werden. Worker lassen sich per RCML extrem flexibel und für nahezu jede beliebige Aufgabe einsetzen. Hier finden Sie detaillierte Informationen zum Aufbau des EOMS.

Es wird empfohlen, mit dem Aufbau des EOMS zu beginnen. Wenn Sie bereits mit dem EOMS vertraut sollten Sie direkt in die Nutzer-/Administrator-Dokumentation einsteigen.

Aufbau des EOMS

In diesem Artikel soll nun zunächst eine Übersicht über den grundlegenden Aufbau eines EOMS gegeben werden. Dies bietet Ihnen einen (vereinfachten) Einblick in die technische Funktionsweise des EOMS.

Ein EOMS besteht aus 2 Haupteinheiten: Dem EOMS-Core sowie den EOMS-Workern. Das Zusammenspiel zwischen dem EOMS-Core und den EOMS-Worker wird später in dem Kapitel Core - Worker detailliert behandelt. Die Anbindung an externe Systeme (z.B. Spooler) erfolgt über das EOMS-Core. Beachten Sie, dass das EOMS nur einwandfrei arbeiten kann, wenn sowohl die Anbindung an externe Systeme, wie auch das EOMS selbst ordnungsgemäß konfiguriert sind. Wie das EOMS richtig konfiguriert wird entnehmen Sie u.a. dem Abschnitt über die Konfigurationen für Core und Worker.

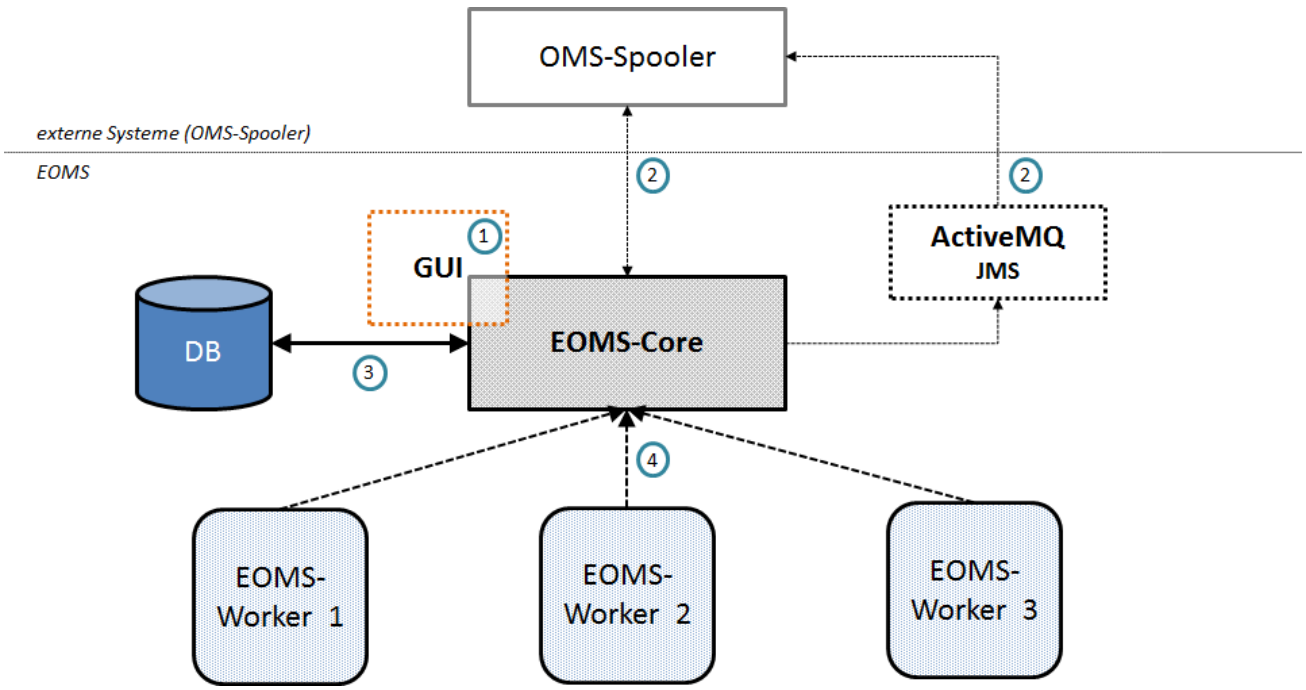


Abbildung A: Vereinfachter Aufbau des EOMS mit Anbindung an den Spooler.

EOMS-Core

Das EOMS-Core stellt das Herzstück des EOMS dar. Es dient zum Monitoring der Worker und empfängt Jobs von Auftrags-Systemen (da das EOMS vor allem auf die Zusammenarbeit mit dem Spooler konzipiert ist, wird in dieser Dokumentation exemplarisch die Kopplung mit dem Spooler beschrieben), die es an die EOMS-Worker weitergibt. Zwischen dem EOMS-Core und dem Spooler findet kein Austausch der zu verarbeitenden Daten statt, sondern das EOMS-Core nimmt lediglich die Beschreibung der zu verarbeitenden Jobs vom Spooler entgegen. Die Kommunikation zwischen dem EOMS-Core und dem Spooler läuft standardmäßig direkt zwischen Spooler und EOMS (seit Version 1.2, *Abb. A (2)*). Das EOMS unterstützt aber auch weiterhin die indirekte Kommunikation über den Messaging-Server Apache ActiveMQ (JMS-Provider), es wird aber die direkte Kommunikation empfohlen. Wie Sie den Kommunikationsweg konfigurieren können, entnehmen Sie dem Artikel über die Anbindung des Spoolers. Beachten Sie jedoch, dass in zukünftigen Versionen die indirekte Kommunikation über ActiveMQ eventuell nicht mehr unterstützt wird.

Die Interaktion mit dem EOMS-Core verläuft über die grafische Benutzeroberfläche. Rufen Sie dazu im Browser den Link zum Tomcat-Server auf: <protocol>://<ip>:<port>/omsinvoker. Standard: http://localhost:8080/omsinvoker

EOMS-Worker

Die EOMS-Worker führen vom EOMS-Core empfangene Jobs aus *Abb. A (4)*. Jeder Worker besitzt eine List of Task der von ihm verarbeitbaren Jobs. Worker können nur Jobs ausführen, die in ihrer List of Task definiert sind. Ist ein Worker also zur Verarbeitung der Tasks "ReportWriter" und "copy" befähigt, so werden vom EOMS-Core nur Jobs an den Worker weitergegeben, bei denen diese Prozesse ausgeführt werden sollen (die List of Task ist nicht bindend, d.h. es kann auch passieren, dass Worker einen Job erhalten, den Sie nicht verarbeiten können. In diesem Fall verweigern sie die Bearbeitung und das EOMS-Core vergibt den Job automatisch neu). Die Anbindung der Worker an das EOMS-Core erfolgt über eine lose Kopplung der Worker an das EOMS-Core. Es besteht also keine direkte bidirektionale Verbindung zwischen EOMS-Core und EOMS-Workern. Stattdessen erfragen EOMS-Worker in fest definierten Zeitintervallen (Standard: 1s) vom EOMS-Core, ob offene Jobs vorliegen, die mit seiner List of Task übereinstimmen. In diesem Fall vergibt das EOMS-Core den offenen Job (inklusive Job-ID) an den Worker. Standardmäßig kann ein Worker nicht mehr als 1 Job gleichzeitig bearbeiten (gilt für OMS-Worker). Worker können aber auch so konfiguriert werden, dass sie mehrere Jobs parallel bearbeiten (für den Einsatz auf Multicore-Maschinen, siehe dazu "Konsumenten" in Konfiguration von Workern). Nachdem der Worker den Jobauftrag vom EOMS-Core enthalten hat, nimmt dieser direkten Kontakt zur Spooler (oder anderen Systemen) per HTTP auf und erfragt und empfängt mit Hilfe der Job-ID die zugehörigen Daten (z.B. Rohdokumente) über die Input-Schnittstelle des Spoolers. Auf diesen Daten werden dann vom Worker die angeforderten Prozesse ausgeführt (z.B. ReportWriter). Die genaue Abfolge der Prozesse, die von einem Worker ausgeführt werden, lassen sich mit RCML beschreiben. Diese sehr mächtige XML-basierte Beschreibungssprache erlaubt auch die Einbindung von Skriptsprachen wie JavaScript und ermöglicht somit nahezu unbegrenzte Prozessdefinitionen. Lesen Sie dazu den Abschnitt über RCML Kompendium. Nach erfolgreicher Beendigung des Jobs sendet der Worker das Resultat zurück an den Spooler (nur OMS-Worker, siehe Typen von Workern).

Worker können auch über Fileshare mit dem Spooler interagieren. Diese Methode ist allerdings obsolet; es wird die Kommunikation über HTTP empfohlen.

Typen von Workern

Es existieren 2 verschiedene Arten von EOMS-Workern:

OMS-Worker	Standardworker zur Verarbeitung von Jobs aus z.B. dem Spooler. Legt Resultat beim Auftrags-System ab.
EOMS-Input-Worker	Unterstützt die Einlieferung per EOMS-Input-Client. Sendet zum EOMS-Input-Server.

Der OMS-Worker bildet den Standardtyp des EOMS und dient zur Lastverteilung von Jobprozessen aus dem Spooler, während EOMS-Input-Worker zur Kommunikation mit dem EOMS-Input-Server von docxworld 5.8 konzipiert sind. Der AKI-Worker wird in der Version 1.3 des EOMS nicht mehr unterstützt. AKI-Worker werden nur bis Version 1.2.4 unterstützt. Mehr zu AKI-Workern finden Sie in der Dokumentation für das EOMS 1.2.

Die einzelnen Workertypen werden detailliert in der Worker-Dokumentation beschrieben. In dieser Einführung wird der OMS-Worker beschrieben, da dieser den typischen Einsatzzweck des EOMS implementiert. Zwar dienen die einzelnen Workertypen unterschiedlichen Einsatzzwecken, konzeptuell unterscheiden sich die verschiedenen Worker allerdings nur gering. Die Spezialisierung eines Workers geschieht ausschließlich über seine RCML.

Ablauf eines Jobs

Im Folgenden soll nun der typische Ablauf eines Jobs vom Eingang des Jobs im EOMS bis zu seiner Terminierung detailliert beschrieben werden (spezifisch für OMS-Worker).

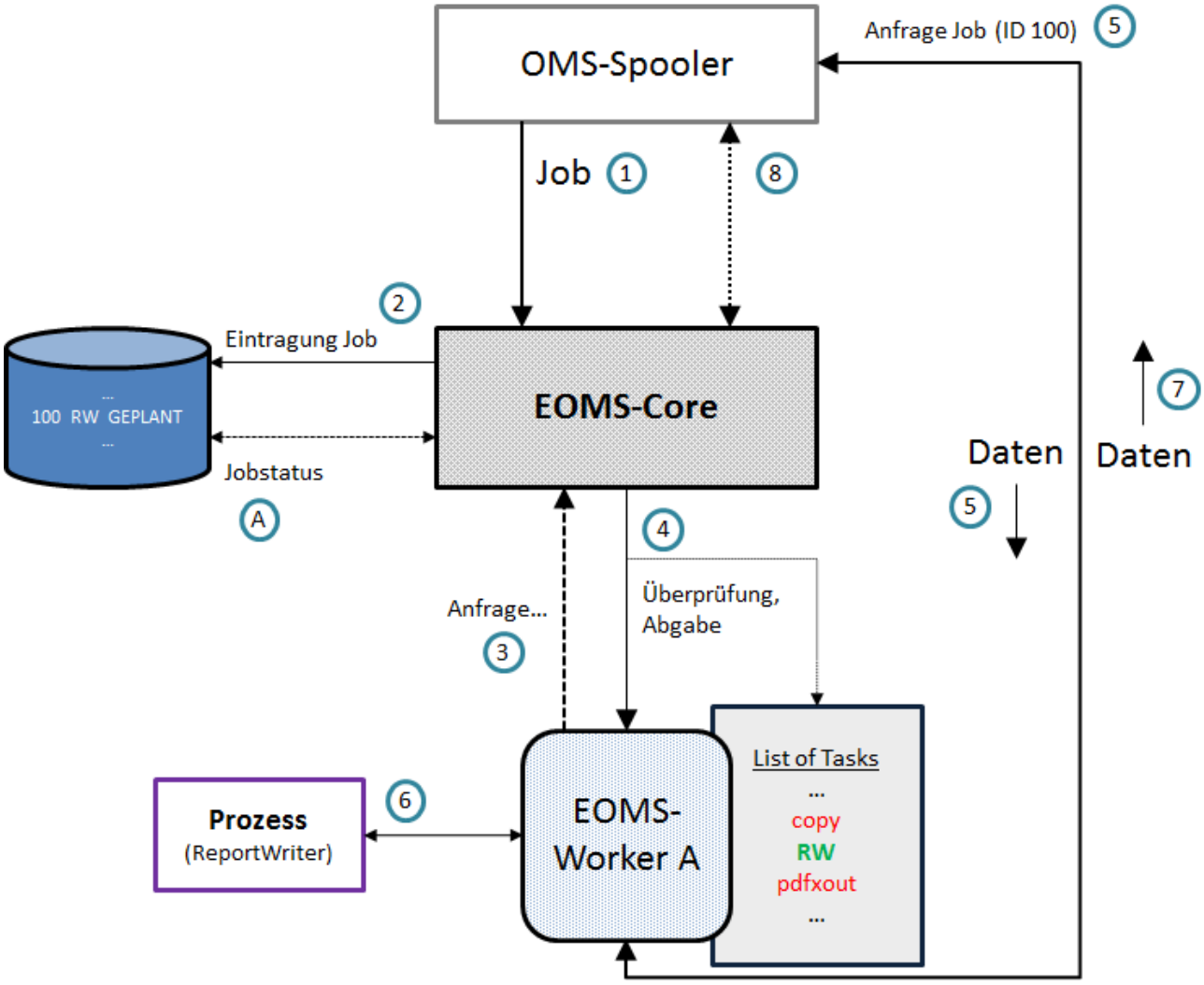


Abbildung B: Abarbeitung eines Jobs aus dem Spooler durch das EOMS.

Die Bearbeitung eines Jobs durch das EOMS verläuft chronologisch in folgenden Schritten (bei Kommunikation von EOMS und Spooler über HTTP, siehe dazu auch Core - Worker):

Schritt	Beschreibung
1	Der Spooler sendet einen offenen Job an das EOMS-Core. Die Jobbeschreibung enthält u.a. die Job-ID *(hier 100)* und den angeforderten Prozess *(RW für ReportWriter)*.
2	Das EOMS-Core trägt den offenen Job in seine Datenbank ein. Der offene Job erhält den Status **GEPLANT**.
3	EOMS-Worker A fragt am EOMS-Core an, ob offene Jobs vorliegen.
A	Das EOMS-Core sucht in seiner Datenbank nach offenen Jobs (Status = **GEPLANT**) und findet Job 100.
4	Das EOMS-Core überprüft, ob Worker A für die Abarbeitung des Jobs geeignet ist (ob in seiner List of Tasks der Task **RW** eingetragen ist). Nur bei erfolgreicher Überprüfung übergibt das EOMS-Core den Job an Worker A.
A	Das EOMS-Core aktualisiert in seiner Datenbank den Status des Jobs 100 auf **AKTIVIERT** (in Bearbeitung und schon an einen Worker vergeben).
5	Worker A verbindet sich zur Input-Schnittstelle des Spoolers und erhält vom Spooler alle Daten, die zum Job mit ID = 100 gehören.
6	Worker A führt den in seiner RCML-Spezifikation vorgegebenen Prozess **RW** aus (es wird also nicht direkt der ReportWriter aufgerufen, sondern der RCML-Prozess RW, aus dem dann (in diesem Fall) der ReportWriter aufgerufen wird).
7 / A	Nach erfolgreicher Bearbeitung sendet Worker A die Ergebnisdaten (Dokumentdaten sowie Fehlerberichte, Logs, etc.) zurück an den Spooler. Dem EOMS-Core meldet der Worker, dass die Bearbeitung von Job 100 vollendet ist. Daraufhin aktualisiert das EOMS-Core den Status des Jobs 100 auf **BEENDET**. Aus Sicht des EOMS ist die Verarbeitung des Jobs hiermit abgeschlossen.
8	In regelmäßigen Abständen fragt der Spooler beim EOMS-Core den Status seines Jobs ab. Gibt das EOMS-Core an dem Spooler zurück, dass der Job **BEENDET** ist, weiß der Spooler, dass der Job fertig verarbeitet ist und liest die vom Worker empfangenen Daten aus (der Spooler weiß davor nicht, dass er bereits Ergebnisse erhalten hat).

Erfolgt die Kommunikation zwischen EOMS und Spooler nicht über HTTP sondern über Fileshare, so werden die Daten nicht zum Worker geschickt, sondern direkt in der Directory des Spoolers durch den EOMS-Worker bearbeitet.

Dieses Ablaufschema bezieht sich spezifisch auf OMS-Worker, ist aber auch für EOMS-Input-Worker und für Worker generell größtenteils gültig. Unterschiede in den verschiedenen Workertypen liegen hauptsächlich in den Schritten 1 (Empfang der Jobs; EOMS-Input-Worker können Daten auch direkt aus SAP-Systemen erhalten) und Schritt 7 (EOMS-Input-Worker senden Ergebnisdaten an den EOMS-Input-Server von docxworld).

Core - Worker

Hier soll nun etwas genauer auf das Zusammenspiel zwischen dem EOMS-Core und EOMS-Workern eingegangen werden. Wie bereits erwähnt fungiert das EOMS-Core als Steuerungszentrum, an dem sich auch die EOMS-Worker anmelden.

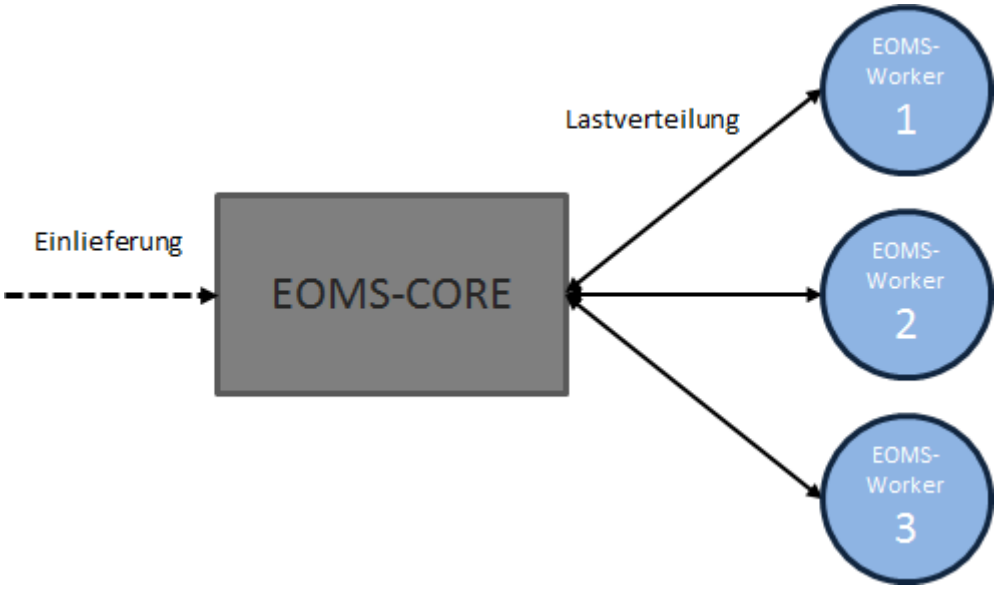


Abbildung A: Einfache Darstellung der Lastverteilung auf EOMS-Worker durch das EOMS-Core.

Das EOMS-Core verteilt also sämtliche eintreffende Jobs auf die bei ihm angemeldeten Worker. Dabei vergibt das EOMS-Core lediglich den Auftrag zur Jobausführung, der Datenaustausch findet ausschließlich zwischen Auftrags-System (z.B. Spooler) und dem zugewiesenen Worker statt *(Vergleiche Aufbau des EOMS)*. Im Folgenden wird nun detailliert beschrieben, wie sich Worker am EOMS-Core registrieren und wie die Jobvergabe durch das EOMS-Core stattfindet.

1. Registrierung des Workers am Core

Nachdem ein Worker gestartet wurde *(mit der jeweiligen .cmd),* versucht dieser automatisch den Verbindungsaufbau zu dem in seiner ***eoms.invoker.client.properties***-Datei angegebenen EOMS-Core. Dort wird die IP und der Port des Hostsystems angegeben *Abb. B (1)*, die natürlich mit dem Port und der IP des EOMS-Core-Hostsystems übereinstimmen müssen *Abb. B (2)*.

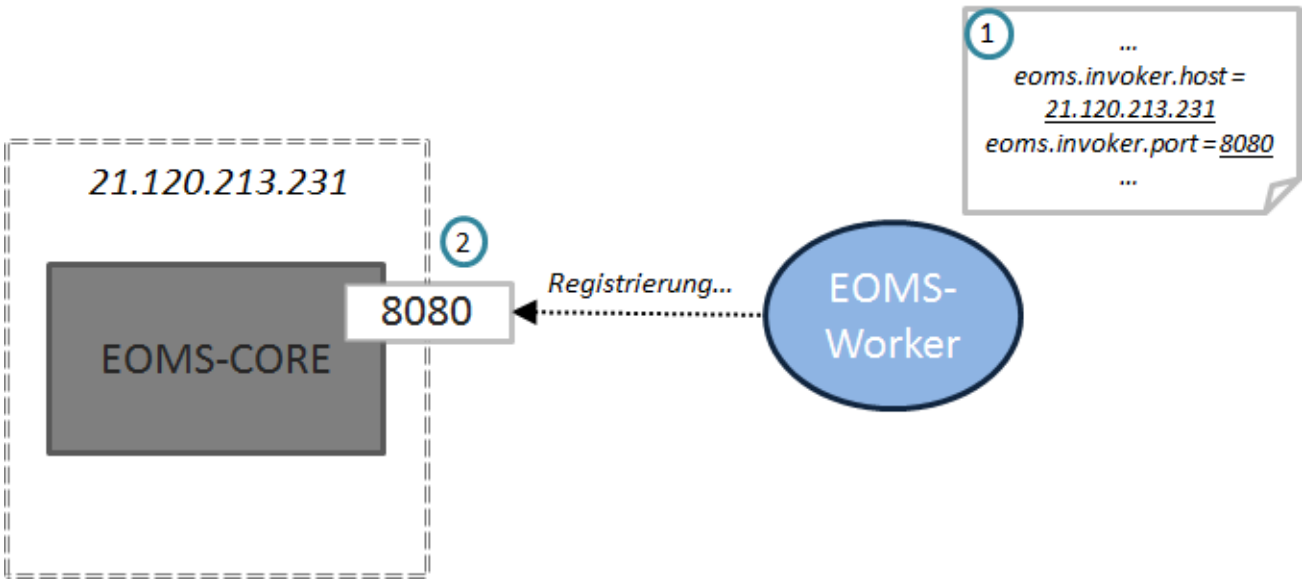


Abbildung B: Worker registriert sich beim EOMS-Core.

Bei erfolgreicher Registrierung ist der Worker als Worker des EOMS-Cores angemeldet und kann von diesem EOMS-Core Jobs empfangen.

Ausgabe des EOMS-Workers bei erfolgreicher Registrierung:

```
2014-08-13 15:43:47,053 927  [oms-worker.1432868859@345295755] INFO  eoms.invoke
r.worker-monitor-service  - registered worker[0]
```

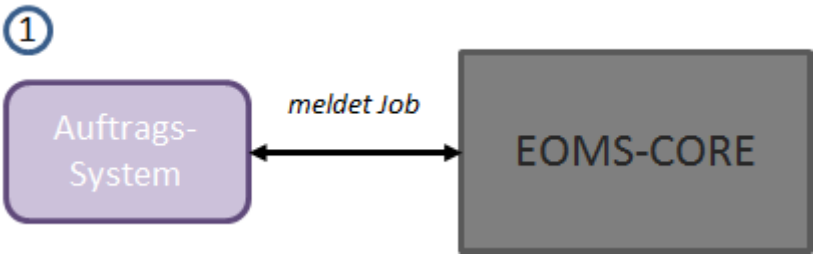
Der Worker wird dann auch im Worker-Monitor gelistet:

ID ▲	Name	CPU	RAM	Konsumenten	Warten	Aktualisiert
40001	oms-worker	83%	3,8 Gb	4/4	1,0 s	07.10.2015 11:26:12

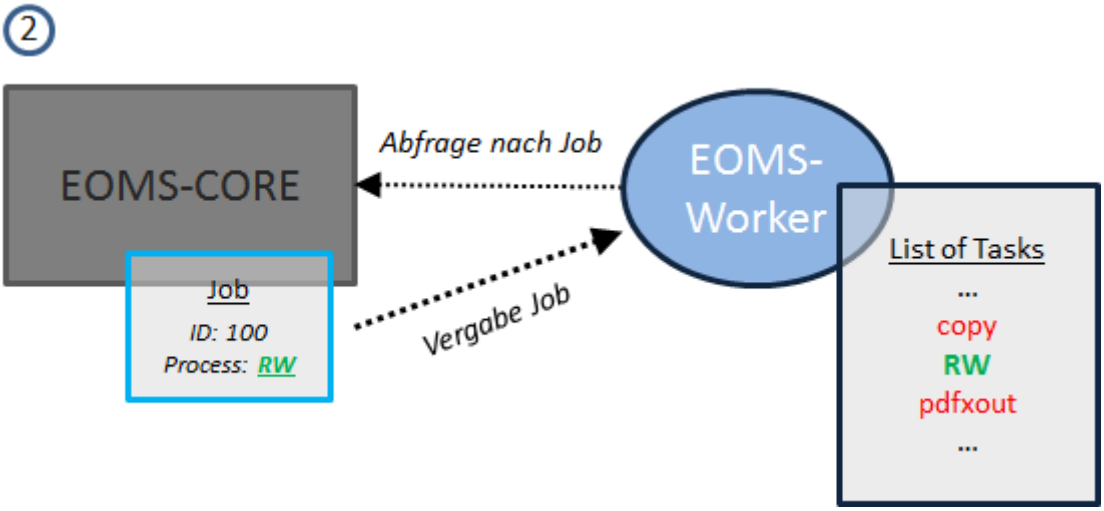
Schlägt die Registrierung des Workers am Core fehl, überprüfen Sie die Logfiles unter */logs*. und beachten Sie die Hinweise zur Fehlerbehandlung.

2. Empfang eines Jobs

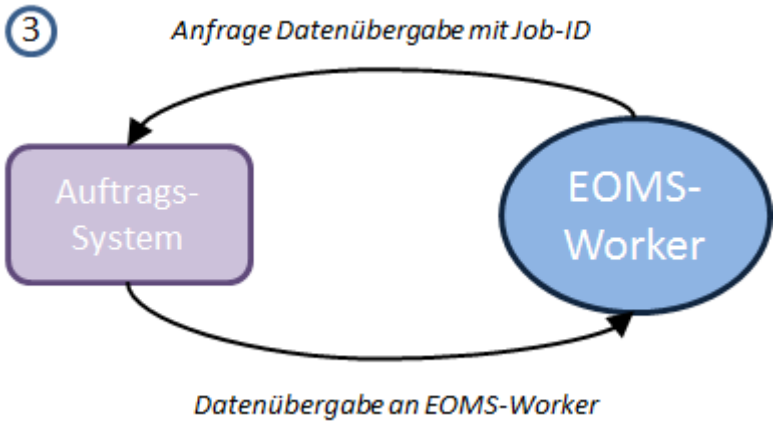
Nach der Registrierung am EOMS-Core ist der Worker bereit, Jobs entgegenzunehmen.



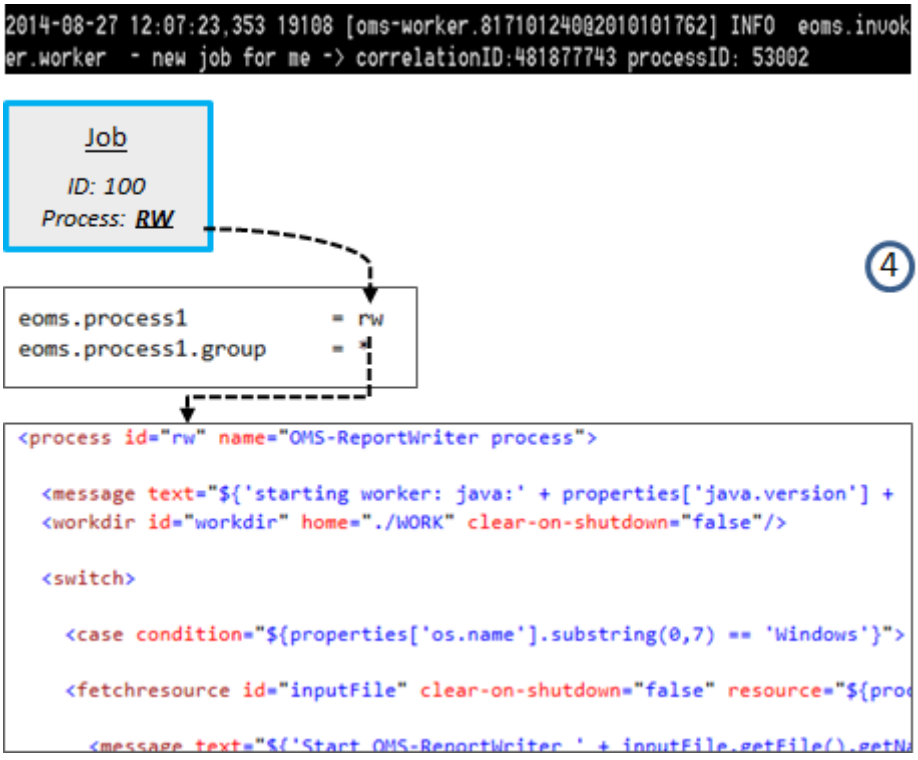
1. Das angebundene Auftrags-System meldet sich beim EOMS-Core und signalisiert einen neuen Job. Dabei übergibt das Auftrags-System seine Adressdaten an das EOMS-Core, damit später die Worker wissen, wo sie das Auftrags-System erreichen können (siehe 4.). Nach Empfang ist der Job als **GEPLANT *[SCHEDULED]*** für Worker freigegeben. Jobs werden chronologisch abgearbeitet, weshalb der Job allerdings erst vergeben wird, wenn keine aktuelleren Jobs mehr warten.



2. Der Job verbleibt solange **GEPLANT *[SCHEDULED]***, bis sich ein geeigneter Worker für den Job meldet. Dazu muss der Worker den angeforderten Prozess für diesen Job beherrschen (er muss in seiner List of Task eingetragen sein). In diesem Fall soll im Job der Prozess "RW" für ReportWriter ausgeführt werden. In der List of Task des Workers ist dieser Prozess explizit aufgeführt, der Worker beherrscht also diesen Prozess (in seiner RCML ist dann der genaue Ablauf festgelegt). Das EOMS-Core vergibt dann den Job an den Worker und stellt den Jobstatus auf **AKTIVIERT *[ACTIVATED]***. Der Job ist dann so lange gesperrt, bis das EOMS-Core Meldung des arbeitenden Workers erhält (bzw. automatisch die Bearbeitung abbricht).

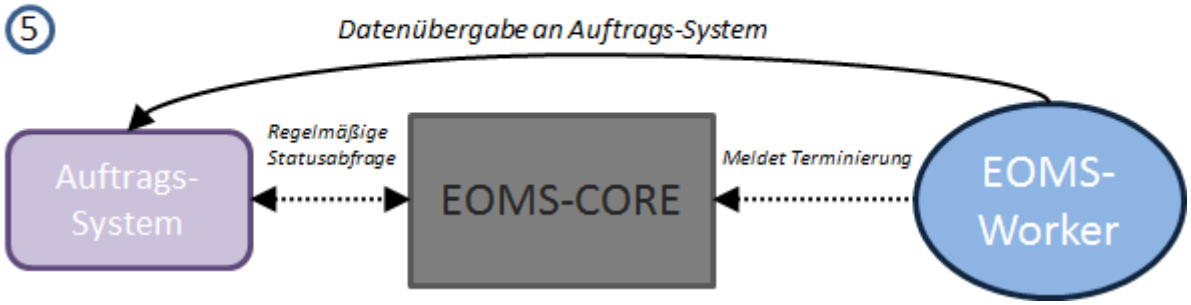


3. Der vom EOMS-Core an den Worker übergebenen Jobinformation entnimmt der Worker auch, von welchem Auftrags-System er sich die Jobdaten, auf die der Prozess angewandt werden soll, abholen kann. Dies kann über verschiedene Wege geschehen, direkt oder indirekt. Im Folgenden wird exemplarisch die direkte Anbindung beschrieben, die auch empfohlen wird. Dabei kontaktiert der Worker das in den Jobinformationen angegebene Auftrags-System und übergibt bei der Anfrage die Job-ID. Das Auftrags-System übergibt dann alle zugehörigen Daten zur Bearbeitung an den Worker, behält die Daten aber weiterhin, falls die Bearbeitung durch den Worker abgebrochen und erneut aufgenommen werden muss.



4. Hat der Worker die Jobdaten vom Auslieferungs-System erhalten, startet es die Ausführung des Prozesses aus seiner RCML-Spezifikation. Dazu vergleicht er den Eintrag in den Jobinformationen mit seiner List of Task, die in der ***worker.properties***-Datei definiert ist. Der dort passende Eintrag wird in der ***worker.rcml***-Datei aufgerufen und abgearbeitet (bei OMS-Workern). Näher auf RCML wird hier eingegangen.

```
2014-08-27 12:07:23,696 19451 [oms-worker.81710240@2010101762] INFO  eoms.invok
er.worker  - terminated :53002
```



5. Nach Abarbeitung des RCML-Prozesses schickt der Worker die fertigen Datenpakete zurück an das Auftrags-System. Die Übertragung erfolgt über die gleiche Schnittstelle wie bei der Übertragung von Auftrags-System zu Worker. Es erfolgt hier keine Statusmeldung an das Auftrags-System. Stattdessen meldet der Worker an das EOMS-Core, dass der Job terminiert ist. Das EOMS-Core setzt daraufhin den Status des Jobs von **AKTIVIERT** ***[ACTIVATED]*** auf **BEENDET** ***[TERMINATED]***. Das Auftrags-System fragt in regelmäßigen Abständen die Status seiner Jobs ab und erfährt dadurch bei der nächsten Abfrage, dass bereits Resultate vorliegen und der Job abgefertigt ist. Das Auftrags-System gibt die Daten dann frei. Die Jobbearbeitung ist jetzt komplett abgeschlossen und der nächste Job kann verarbeitet werden.

3. Abmeldung des Workers am Core

Um einen Worker herunterzufahren und ihn vom Core abzumelden genügt es, das Kommandofenster des Workers zu schließen. Das EOMS-Core registriert nach einer Latenzzeit beim nächsten Refresh selbstständig, dass der Worker nicht mehr aktiv ist. War dem Worker zur Zeit des Herunterfahren ein Job zugeteilt und der Worker konnte den Job nicht vollenden, so wird der Job vom EOMS-Core zurückgesetzt und erneut vergeben. Der Worker wird dann auch automatisch aus dem Worker-Monitor entfernt.

Der Prozess-Pool

Das EOMS-Core verfügt über einen internen Zwischenpuffer für geplante *[scheduled]* Prozesse, den Prozess-Pool. In diesem Prozess-Pool werden einkommende Prozesse eingelagert, bevor Sie verarbeitet werden. Das EOMS-Core vergibt nur Prozesse aus dem Prozess-Pool an Worker, nicht direkt aus der Prozessdatenbank.

Aufbau des Prozess-Pools

Der Prozess-Pool ist ein Buffer, der dazu dient, Prozesse aus der Prozessdatenbank zwischenzuspeichern, bevor sie an Worker vergeben werden. Einer der Vorteile dabei ist, dass Sie diesen Buffer relativ detailliert konfigurieren können. Dies gibt Ihnen eine bessere Kontrolle über die Prozessverteilung.

Der Prozess-Pool ist folgendermaßen aufgebaut:

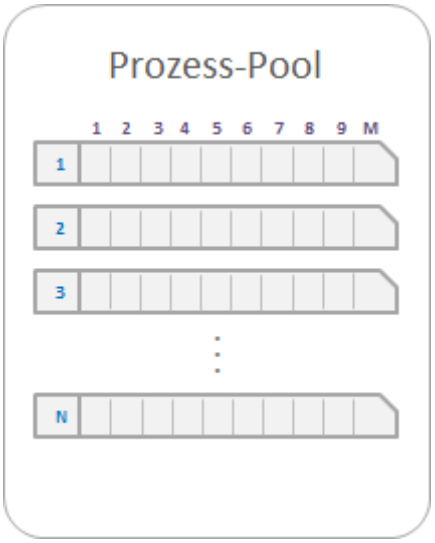


Abbildung A: Aufbau des Prozess-Pools.

Der Prozess-Pool besteht aus N Parzellen mit jeweils M Speicherplätzen. Pro Speicherplatz kann genau 1 Prozess aufgenommen werden. Im obigen Beispiel ist M = 10, jede Parzelle kann also genau 10 Prozesse zwischenspeichern. Jede Parzelle kann nur Prozesse von genau einem Typ aufnehmen, jede Parzelle ist also für einen anderen Prozess-Typ "zuständig". Im obigen Beispiel könnte z.B. Parzelle 1 für Prozesse vom Typ "Report-Writer" verantwortlich sein, Parzelle 2 für Prozesse vom Typ "rs", usw. Diese Verantwortlichkeit ist nicht fest, leere Parzellen haben keine Bestimmung bezüglich des Prozess-Typs, den sie aufnehmen können. Nur wenn mindestens 1 Prozess in einer Zelle gespeichert ist kann diese Parzelle nur noch Prozesse vom selben Typ aufnehmen. Dies führt also dazu, dass im Prozess-Pool Prozesse von maximal N verschiedenen Prozess-Typen zwischengelagert werden können. Insgesamt fasst der Prozess-Pool also N * M Prozesse. Wichtig hierbei ist aber zu beachten, dass von jedem unterschiedlichen Prozess-Typ nur maximal M Prozesse in den Prozess-Pool aufgenommen werden können. Wenn Sie also grundsätzlich nur 1 Prozess-Typ mit Ihrem EOMS verarbeiten macht es keinen Sinn, im Prozess-Pool 20 Parzellen mit jeweils 100 Speicherplätzen zu reservieren, da in diesem Fall 1900 Speicherplätze immer frei bleiben werden.

Allgemeine Funktionsweise

Im Folgenden soll erläutert werden, welche Aufgabe der Prozess-Pool bei der Prozessverarbeitung spielt. Dieser Ablauf gilt für den Typ **cluster**.

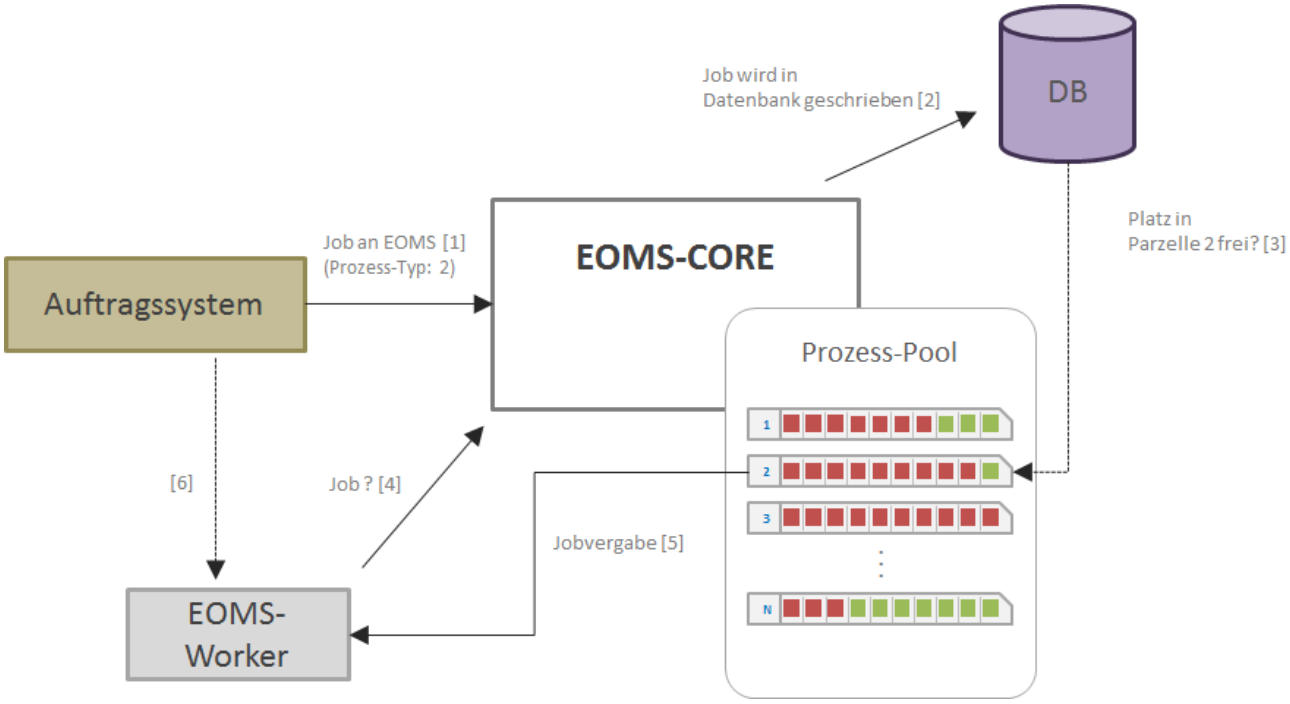


Abbildung B: Ablauf Prozesseingang und -verarbeitung.

Abb. B zeigt auf, wie der Prozess-Pool als Zwischenspeicher für Prozesse im EOMS-Core fungiert.

Dazu wird in *Abb. B* der typische Ablauf vom Prozesseingang bis zur Prozessvergabe vereinfacht dargestellt (detailliert wurde dies bereits im Artikel zum Aufbau des EOMS erläutert).

Im Folgenden werden die einzelnen Schritte, die in *Abb. A* mit [#] gekennzeichnet sind, erklärt:

1. Das Auftragssystem (z.B. Spooler oder Redaktions-System) vergibt einen Job (Prozess) an das EOMS. Dabei wird lediglich ein Prozessauftrag beim EOMS registriert, die Daten verbleiben beim Auftragssystem. Der Prozess ist hier vom Typ 2 (wir stellen hier Prozess-Typen vereinfacht als Zahlen da, real z.B. vom Prozess-Typ "ReportWriter").
2. Das EOMS-Core schreibt den Prozess (Job) in die Datenbank.
3. Es wird periodisch überprüft, ob im Prozess-Pool für den Prozess Platz ist. Dazu muss in Parzelle 2 (also in der passenden Parzelle für diesen Prozess-Typ) min. 1 Speicherplatz frei sein (ist hier der Fall). Ist dies der Fall, wird der Prozess im Prozess-Pool eingetragen. Ist dies nicht der Fall, verbleibt der Prozess weiter in der Datenbank und kann vorerst nicht an einen Worker vergeben werden.
4. Der Worker fragt an, ob zu bearbeitende Prozesse für ihn verfügbar sind (der Worker muss in seiner List of Tasks den Prozess 2 definiert haben).
5. Das EOMS vergibt den Prozess aus dem Prozess-Pool (und nicht direkt aus der Datenbank) an den Worker.
6. Der Worker empfängt die für den Prozess benötigten Daten vom Auftragssystem.

Konfiguration des Prozess-Pools

Ein Vorteil des Prozess-Pools ist, dass Sie ihn relativ frei konfigurieren können. Folgende Anpassungen sind möglich:

- Anzahl Parzellen des Prozess-Pools

- Bestimmt die Anzahl der Parzellen im Prozess-Pool, also wie viele unterschiedliche Prozess-Typen im Prozess-Pool gespeichert können.
- Konfiguration über eoms.invoker.process-pool.nodes

Speicherplätze pro Parzelle

- Bestimmt, wie viele Prozesse pro Parzelle gespeichert werden können, also wie viele Speicherplätze jede Parzelle zur Verfügung
- Konfiguration über eoms.invoker.process-pool.size

Wahl der Prozess-Pool Befüllungslogik

- Bestimmt, wie Prozesse im Prozess-Pool gespeichert werden sollen.
- Konfiguration über eoms.invoker.process-pool.type
- Mögliche Werte:

1. **cluster**: Für jeden Prozesstyp wird eine Parzelle im Prozesspool reserviert. Ein Prozess wird nur zum Prozesspool hinzugefügt, wenn die entsprechende Parzelle einen freien Platz hat, sonst muss der Prozess so lange warten, bis ein Prozess dieses Typs abgefertigt wurde und die Parzelle wieder Prozesse aufnehmen kann. Ist der in diesem Artikel vorgestellte Fall.
2. **default**: Es werden so lange Prozesse zum Pool hinzugefügt, bis dieser voll ist, danach müssen alle Prozesse warten. Wird ein Platz frei, wird ein beliebiger Prozess in den Pool hinzugefügt. Es wird nicht nach Prozesstypen unterschieden. In diesem Fall ist der Prozess-Pool ein einfacher Puffer.

Es ist wichtig, den Prozess-Pool an Ihre Bedürfnisse anzupassen, um maximale Effizienz zu erreichen. Die optimale Konfiguration des Prozess-Pools hängt dabei stark von der Struktur Ihrer Prozessverteilung und von den Kapazitäten Ihres Systems ab. Prinzipielle Empfehlungen:

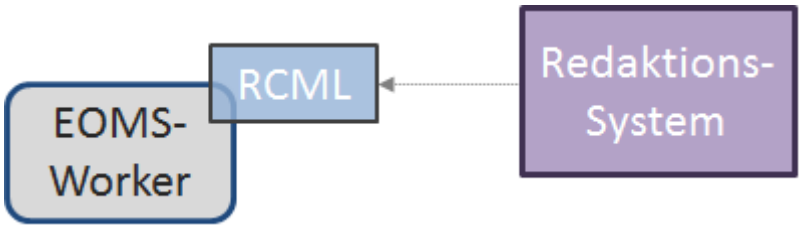
- Bei wenigen Prozessen und wenigen unterschiedlichen Prozess-Typen: Standardwerte.
- Bei vielen Prozessen, aber wenigen unterschiedlichen Prozess-Typen (< 20): size-Attribut eventuell erhöhen, nodes-Attribut Stan
  oder verringern oder Umstellung auf **default**.
- Bei wenigen Prozessen, aber vielen unterschiedlichen Prozess-Typen (> 20): size-Attribut eventuell verringern, nodes-Attribut erh
- Bei vielen Prozessen und vielen unterschiedlichen Prozess-Typen (> 20): size-Attribut erhöhen, nodes-Attribut erhöhen.

Wenn Sie wenig verschiedene Prozess-Typen verarbeiten kann es Sinnvoll sein, den Prozess-Pool komplett auf **default** umzu stellen.

Vergleichen Sie hierzu die Konfigurationsseite des Cores.

EOMS als Basis fuer das Redaktions-System

Mit dem EOMS ist es ohne Weiteres möglich, Daten aus einem Redaktions-System (Pakete) zu laden und zu verwenden. Damit stellt das EOMS eine Möglichkeit dar, Daten aus einem Redaktions-System zu extrahieren. Ein Redaktions-System ist somit erst in Kombination mit einem EOMS in vollem Umfang einsetzbar.



Die Anbindung (Datenaustausch) eines Redaktions-System geschieht (wie alles, was ein Worker tut) über seine RCML-Spezifikation. Diese ist völlig frei programmierbar. Mehr dazu hier.

Ein Redaktions-System wird nie dauerhaft an einen Worker "angebunden", die Verbindung besteht also nur temporär für die Zeit des Datenaustauschs.

Seit Version 1.2.4 können Produktions- und Binärpakete auch dann empfangen werden, wenn der R-S Server eine Zeit lang nicht erreichbar ist (offline). Zur Konfiguration müssen in der **rs.client.properties** folgende Direktiven gesetzt werden:

Eigenschaft	Beschreibung	Standard
rs.client.production-bundle-fetcher.offline-timeout	Zeit in ms, wie lange das Redaktions-System nicht erreichbar sein darf, bis keine Produktions-Pakete mehr empfangen werden dürfen.	3600000
rs.client.binary-bundle-fetcher.offline-timeout	Zeit in ms, wie lange das Redaktions-System nicht erreichbar sein darf, bis keine Binär-Pakete mehr empfangen werden dürfen.	3600000

Benötigt Redaktions-System Version 2.1.1.

Typischerweise läuft eine Zusammenführung von Rohdaten und Paketen aus dem Redaktions-System folgendermaßen ab:

1. Empfang der Rohdaten (XML) aus dem Auftrags-System (Spooler).
2. Empfang des dazugehörigen Binär- und Produktions-Paket aus dem R-S (nur falls die Pakete nicht bereits im Cache (also auf dem Workerrechner) vorliegen).
3. Ausführung des Programms aus dem Binär-Paket auf die XML-Rohdaten und das Produktions-Paket.
4. Weiterverarbeitung (z.B. Rücksendung) der Ergebnisdaten.

> Sie sind aber, wie bei allem, was in RCML definiert wird, völlig frei bei der Festlegung, was getan werden soll.

Damit überhaupt Pakete vom Redaktions-System zum EOMS-Worker übertragen werden können, müssen folgende Parameter gesetzt sein:

- docxworld-Vertrag **oder** docxworld-Link: Zur eindeutigen Identifizierung, welche Pakete für diesen Job (diese Rohdaten) benötigt
- binary-bundles-home: Wo sollen Binär-Pakete auf dem Workerrechner abgelegt werden?
- production-bundles-home: Wo soll Produktions-Pakete auf dem Workerrechner abgelegt werden?
- merge-home / runtime-home: u.a.: Wo soll die Ausführung des Programms aus dem Binär-Paket stattfinden (hier hängt es davon welche Methode zur Ausführung Sie verwenden: *merged* oder *shared.* Mehr dazu finden Sie in der Elementbeschreibung von <docxworld-fetch-production-environment>).

All dies geschieht über das ***<docxworld-fetch-production-environment>*** - RCML-Element. Dieses Element kümmert sich um alles, Sie müssen nur die oben aufgeführten Parameter zur Verfügung stellen. Falls Sie sich bereits mit RCML auskennen, hier ein vollständiges (mehr ist prinzipiell nicht zu tun, um eine Basisanbindung zu realisieren) Beispiel zur Interaktion mit dem Redaktions-System. Falls Sie sich noch nicht mit RCML auskennen, ist das auch nicht schlimm. Dies lernen Sie in einem späteren Abschnitt der Dokumentation.

```
<workdir id="workdir" home="./WORK" />
<fetchresource id="inputFile" clear-on-shutdown="false"
resource="${process['eoms.process.input']}" />

<docxworld-fetch-production-environment id="rsProductionEnvironment"
runtime-environment="shared">

<docxworld-contract>${process['eoms.procedure']}</docxworld-contract>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->

<binary-bundles-home>${binaryBundleHome}</binary-bundles-home>
<production-bundles-home>${productionBundleHome}</production-bundles-ho
me>

<runtime-home>${workdir.getAbsolutePath()}</runtime-home>
<merge-home>${buildPath4ClientProcedure(process['eoms.client'],
process['eoms.procedure'])}</merge-home>

</docxworld-fetch-production-environment>
```

Danach können Sie bereits mit den Paketen arbeiten. Zur Ausführung des Programms aus dem Binär-Paket, verwenden Sie am Besten den Kommandobefehl aus den XML-Rohdaten:

```
<exec id="rwProcess" process="rwProcess" workdir="workdir">
<param name="cmd"
value="${rsProductionEnvironment.getCommandLine(inputFile.getFile())}"
/>
<commandline processor="velocity">$cmd</commandline>
</exec>
```

Nun schicken wir die Ergebnisdaten zurück:

```
<switch>
<case condition="${rwProcess.testReturnCode('0')}">
<message text="${'####ReturnCode OK:'+rwProcess.getReturnCode()}" />
<upload file="workdir" destination="${process['eoms.process.output']}"
/>
</case>
<case condition="true">
<message text="${'####Fehler ReturnCode:'+rwProcess.getReturnCode()}"
/>
<upload file="workdir" destination="${process['eoms.process.output']}"
/>
</case>
</switch>
```

Eine vollständige Anleitung finden Sie in dem HowTo Artikel Interaktion mit dem Redaktions-System.

Hinweise zur Online-Dokumentation

Teile dieser Online-Dokumentation können die Nutzer und Administratoren des EOMS für Nutzer zur eigenen Verwendung ausdrucken, e s gelten die Rechtlichen Hinweise. Die verwendeten Symbole, Schriftarten und deren Bedeutung werden auf dieser Seite erläutert: Verw endete Symbole. Abbildungen des EOMS, Grafiken und Diagramme wurden teilweise aus Platzgründen in ihrer Größe bearbeitet. Für eine optimale Darstellung der Online-Dokumentation sollte Ihre Bildschirmauflösung 1680x1050 Pixel oder höher betragen, mindestens jedoch 1440x900. Falls Sie weitergehende Fragen haben, die in der Dokumentation nicht thematisiert werden finden Sie hier eine Liste ergänzender Dokumentationen. Ansonsten wenden Sie sich bitte an unseren Support.

Verwendete Symbole

In der Nutzer-Dokumentation gibt es diverse Symbole und Zeichen.

Um Ihnen einen Überblick zu verschaffen, sind hier die wichtigsten Symbole erklärt.

Symbol	Beschreibung
	Dieses Symbol gibt zusätzliche Informationen zu einem Textabschnitt.
	Dieses Symbol gibt Informationen, die unbedingt beachtet werden müssen.
	Dieses Symbol gibt Hinweise, die beachtet werden sollten.
	Dieses Symbol enthält ein Beispiel für einen Textabschnitt.
 	Verfügbare Aktionen werden durch diese Hintergründe gekennzeichnet.
	Dieses Symbol beinhaltet vor allem Inhaltsverzeichnisse, Auflistungen und weiterführende Links.
Hallo!	Beinhaltet Informationen, die Sie durch einen Klick auf die Überschrift ausklappen können.

EOMS-Dokumentation	Verlinkung auf eine andere Seite der EOMS-Dokumentation.
	Symbole entsprechen den Symbolen 1 - 4, können aber auch als Symbole in Tabellen verwendet werden.
	Diese Symbole kennzeichnen Vor- und Nachteile oder geben zusätzliche Funktionen oder Merkmale bzw. Warnungen an.
	Zwingende Angabe / Pflichtfeld / Zwingende Aktion.
	Nichteingabe erzeugt zwar keinen Fehler, ist aber notwendig für die korrekte Arbeitsweise des EOMS-Core.
	Optionale Angabe
	Wenn dieses Symbol in den Screenshots auftaucht, gibt es unter der Abbildung Erklärungen zu der/den Nummer/n.
	Dieses Symbol fasst mehrere Elemente in einer Abbildung zusammen.
	Dieses Symbol hebt Bildausschnitte hervor.
Abb. A (1)	Gibt im Text an, dass sich eine Erklärung auf eine Abbildung mit dem jeweiligen Buchstaben und der jeweiligen Nummer bezieht. Enthält eine Seite nur eine Abbildung, kann statt *Abb. A (1)* auch nur *(1)* stehen. Enthält eine Seite mehrere Abbildungen, beziehen sich Angaben ohne explizite Abbildungsangabe wie *(1)* immer auf Abbildung A.
STRG + **ALT** + **ENTF**	Tastenkombination.

Nutzer-Dokumentation

In der Nutzer-Dokumentation lernen Sie den sicheren Umgang Nutzung mit dem EOMS. Da sich das EOMS in 2 unterschiedliche Systeme (Core und Worker) unterteilt, ist auch die Nutzer-Dokumentation zweigeteilt.

Installation, Betrieb, Sicherheitsaspekte usw. werden in der Nutzer-Dokumentation nicht behandelt. Beachten Sie hierzu die Administrator-Dokumentation.

Als Nutzer beschränken sich Ihre Aufgaben im EOMS im Wesentlichen auf:

- Bedienung / Steuerung des EOMS-Core über die grafische Oberfläche (GUI),
- Basiskonfiguration des EOMS-Core,
- Basiskonfiguration und Verwaltung der EOMS-Worker.

Vermittelt wird das dazu nötige Wissen in folgenden Kapiteln:

Bevor Sie mit der Nutzer-Dokumentation beginnen ist es ratsam, zum Einstieg die Einführung zu lesen.

Systemstart

Nach erfolgreicher Installation ist das EOMS startbereit. Die zur Inbetriebnahme notwendigen Schritte können je nach verwendetem Workertyp, Messagingart (über ActiveMQ oder direkt), verwendetem Datenbanksystem usw. variieren. Dieser Artikel leitet Sie deshalb durch die für Ihre Systemkonfiguration nötigen Maßnahmen, um das EOMS einsatzbereit zu machen.

Beachten Sie, dass dem Start des EOMS die ordnungsgemäße Konfiguration vorausgeht. Sind die einzelnen Komponenten nicht richtig konfiguriert ist das System eventuell nicht lauffähig.

Beachten Sie dazu die verschiedenen Konfigurationshinweise in dieser Dokumentation. Hier finden Sie eine Kurzübersicht über die Basiskonfigurierung, die auch noch einmal den Verbindungsaufbau beschreibt.

Beachten Sie außerdem, dass dies nur eine Kurzanleitung zum Systemstart darstellt. Detailliertere Informationen finden Sie in der Administrator-Dokumentation.

Generell besteht der Start des EOMS aus folgenden Teilschritten:

Start des Datenbank-Servers

Das EOMS-Core legt Jobinformationen in einer externen Datenbank ab. Es werden verschiedene relationale Datenbanken unterstützt, die Voreinstellung bei Auslieferung ist derby. Eine Übersicht über unterstützte Datenbanken finden Sie hier.

Der Start des Datenbank-Server hängt davon ab, welches Datenbanksystem Sie verwenden. Den Derby-Datenbankserver starten Sie, indem Sie das Kommandozeilentool unter Windows öffnen und folgenden Befehl ausführen:

```
> java -jar %DERBY_HOME%\lib\derbyrun.jar server start
```

Wobei %DERBY_HOME% das Verzeichnis ist, in das Sie die derby Datenbank installiert haben. Die Angabe -p XXXX startet den Server auf Port XXXX. Sie können den Derby-Server auch auf einem anderen Port (ohne Angabe auf dem Standardport) starten. Dann müssen Sie diesen Port in der Konfiguration des EOMS-Core angeben (dortige Standardeinstelung: 1527).

Zuvor muss eine Datenbank für das EOMS manuell angelegt werden. Wie Sie die Datenbank in derby richtig erstellen lesen Sie hier.

Sie können Ihren Datenbank-Server auch als Service starten lassen. Der Server fährt dann bei Systemstart automatisch hoch. Dies erfordert jedoch eventuell weitere Schritte, die hier nicht behandelt werden. Das manuelle Starten des Datenbank-Servers wird dann überflüssig.

Start des Messaging-Servers

Der Start des Messaging-Servers ApacheMQ ist nur notwendig, wenn Sie die Kommunikation zwischen EOMS-Core und Auftrags-Systemen indirekt über JMS leiten wollen.

Wenn ActiveMQ bei der Installation richtig konfiguriert wurde ist es bereits als Service eingetragen und startet sich bei Systemstart automatisch. Falls Sie dennoch einmal ActiveMQ manuell starten müssen, können Sie das im Kommandozeilentool mit folgender Befehlszeile tun:

```
> %ACTIVEMQ_HOME%/activemq
```

Falls die Kommunikation nicht über ActiveMQ geleitet werden soll sind außer der Konfigurierung keine weiteren Schritte nötig.

Start des EOMS-Core

Nun können Sie das EOMS-Core starten. Dazu müssen Sie lediglich den Tomcat-Server des EOMS starten. Je nachdem, welche Tomcat-Distribution Sie verwenden geschieht dies entweder per Kommandozeilenaufruf oder direkt mit Hilfe des Tomcat Starttools (oder Sie tragen Tomcat direkt als Service ein).

EOMS 1.3 benötigt Tomcat 8 oder höher.

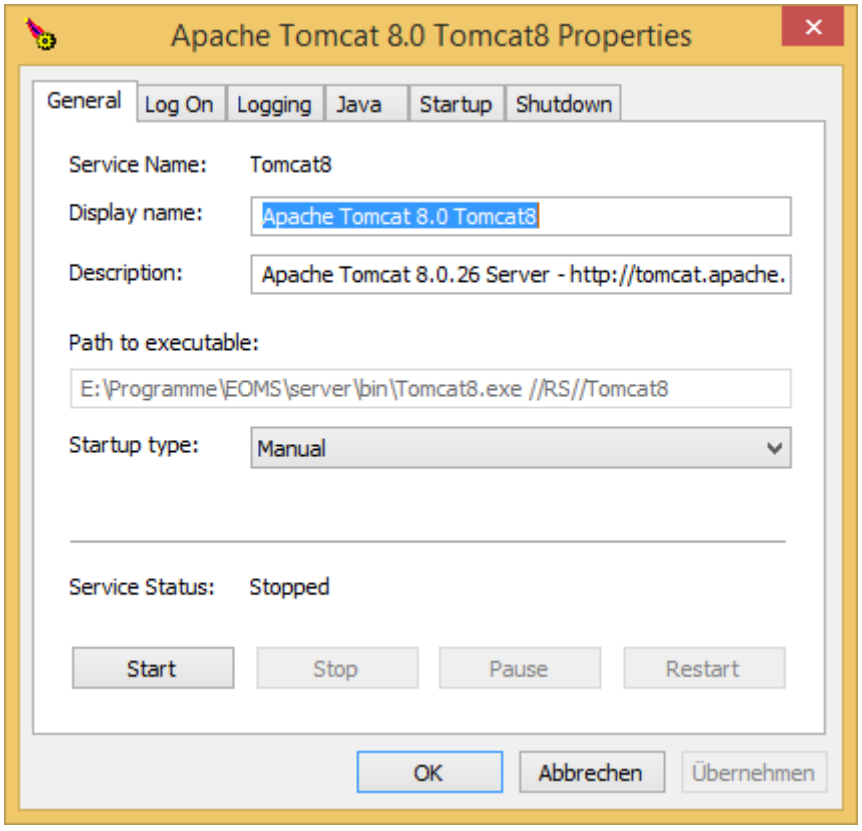
1. Start über die Befehlszeile:

```
%EOMS-CORE_HOME%/bin/catalina start
```

Wobei %EOMS-CORE_HOME% das Verzeichnis ist, in das Sie EOMS-Core installiert haben.

2. Start durch das Tomcat Starttool:

- Öffnen Sie *%EOMS-CORE_HOME%/bin/tomcatXw.exe [wobei X durch die Versionsnummer von Tocmat ersetzt wird].*



- Klicken Sie auf Start.

Rufen Sie dann die grafische Oberfläche des EOMS-Core mit Ihrem Browser auf. Der Aufruflink hat die Form "<ip>:<port>/omsinvoker". Falls Sie Tomcat mit den Standardports starten ist <port> = 8080. Greifen Sie vom gleichen Rechner auf die GUI zu, auf dem auch EOMS-Core installiert ist, so lautet der Aufruflink: "localhost:8080/omsinvoker".

Start der EOMS-Worker

Worker können Sie ganz einfach durch Ausführen der jeweiligen Batchdatei starten. Diese liegen in *%WORKER_HOME%*. Verwenden Sie

- start-oms-worker.cmd zum Start eines OMS-Workers.
- start-eomsinput-worker.cmd zum Start eines EOMS-Input-Workers.

Der AKI-Worker wird in der Version 1.3 des EOMS nicht mehr unterstützt. AKI-Worker werden nur bis Version 1.2.4 unterstützt. Mehr dazu finden Sie in der Dokumentation für das EOMS 1.2.

Anbindung an Auftrags-Systeme

Die Anbindung an Auftrags-Systeme (externe Systeme) erfolgt per Konfiguration, weshalb keine weiteren Schritte nötig sind. Stellen Sie jedoch sicher, dass das angebundene System gestartet und erreichbar ist. Weiteres entnehmen Sie diesem Artikel.

Herzlichen Glückwunsch!

Falls keine Fehler beim Start des EOMS aufgetreten sind, können Sie jetzt loslegen. Sollte doch etwas schief gelaufen sein, finden Sie hier typische Probleme beim Start des EOMS und Tipps zur Fehlerbehandlung.

EOMS-Core

Das EOMS-Core ist der zentrale Steuerungsknoten des EOMS. Es verwaltet Worker, empfängt und verteilt Jobs der Auftrags-Systeme an diese und ermöglich die Überwachung des gesamten Prozessablaufs.

Als Endbenutzer werden Sie vor allem mit dem EOMS-Core-Monitor in Berührung kommen, welcher alle Aktivitäten des EOMS-Core aufzeichnet und übersichtlich darstellt. Die Konfiguration des EOMS-Core hingegen geschieht ausschließlich auf Dateiebene.

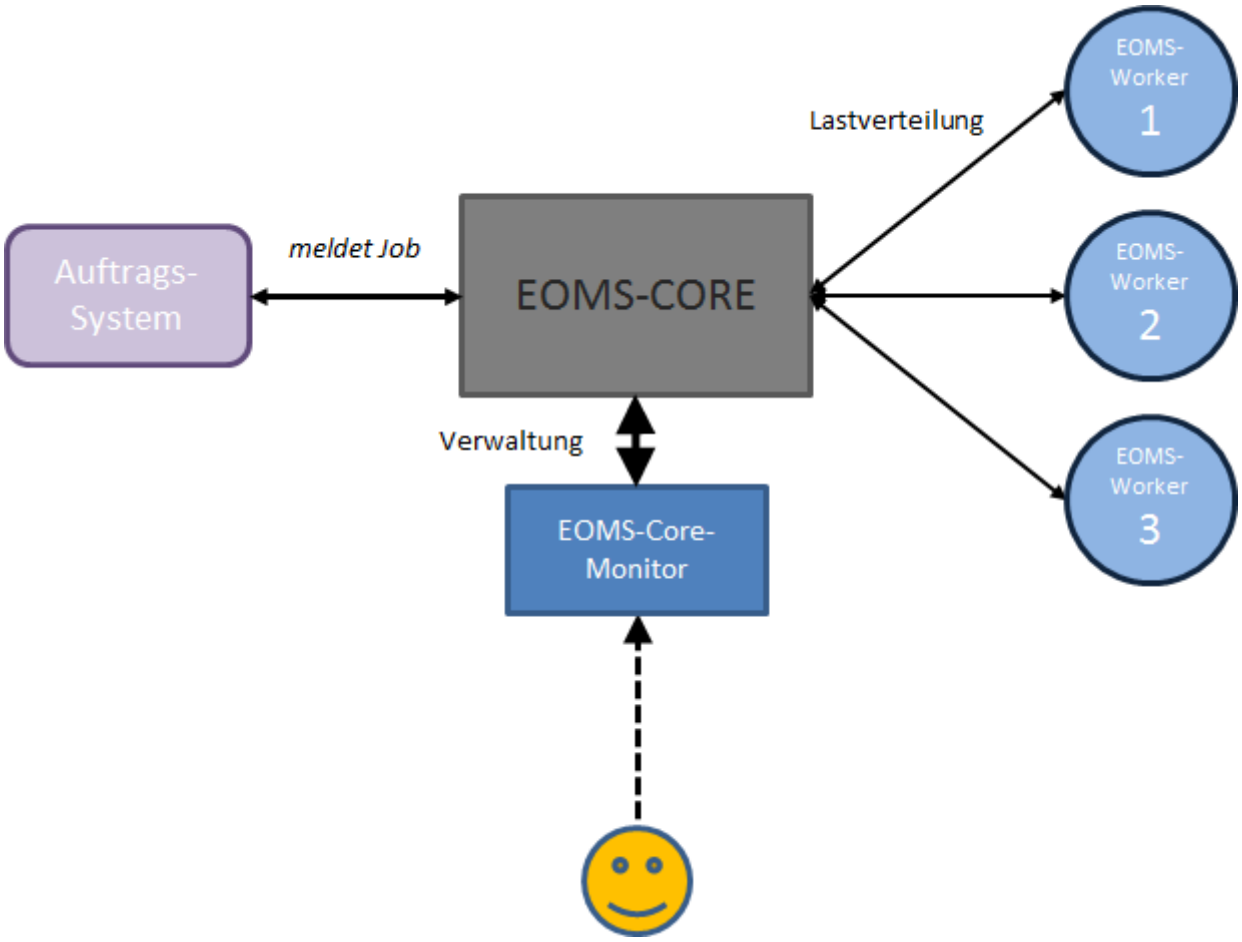


Abbildung A: Aufgaben des EOMS-Core

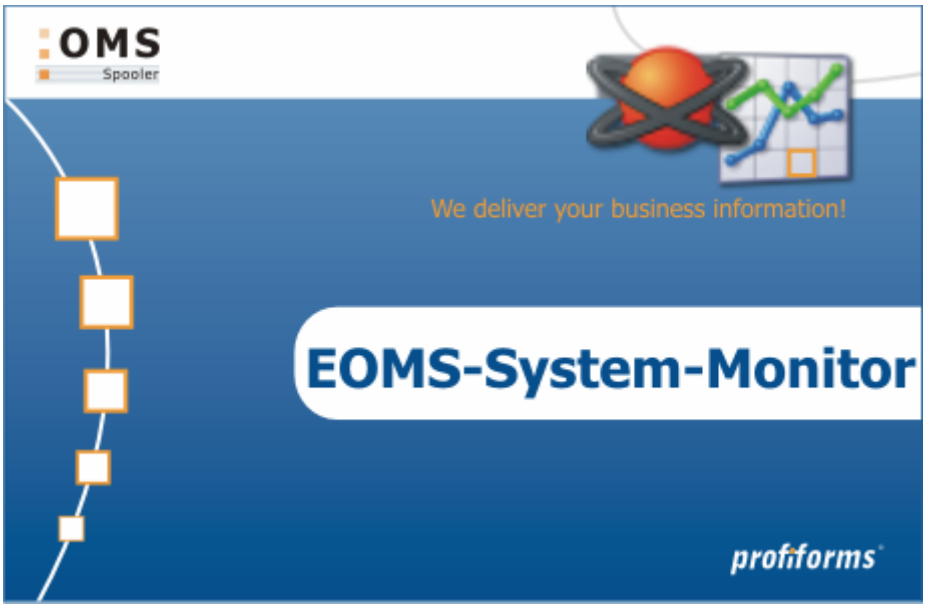
In *Abb. A* werden die Hauptaufgaben des EOMS-Core dargestellt:

1. Entgegennahme der Jobs von Auftrags-Systemen (Einlieferung) und Auslieferung des fertigen Jobergebnisse an Auftrags-Systeme.
2. Verteilung der Jobs auf Worker (Lastverteilung).
3. Verwaltung und Überwachung sämtlicher Prozesse und deren Bereitstellung für den Benutzer über den EOMS-Core-Monitor.

> Vergleichen Sie hierzu auch, falls noch nicht geschehen, die Einführung in den Aufbau des EOMS.

Dieses Kapitel über das EOMS-Core befasst sich vor allem mit dem EOMS-Core-Monitor sowie der Konfiguration auf Dateiebene.

Erste Schritte



Die grafische Oberfläche des EOMS-Core ("EOMS-Core-Monitor") ist übersichtlich und lässt sich intuitiv bedienen.

In diesem Abschnitt werden Sie zunächst mit der Benutzeroberfläche des EOMS-Monitors vertraut gemacht.

Verwenden Sie zum Öffnen einen unterstützten Webbrowser und öffnen Sie:

http://<ip>:<port>/omsinvoker

Wobei <ip> = die IP-Adresse des Hostrechners auf dem EOMS-Core gestartet ist (Wenn vom Rechner, auf dem EOMS-Core gehostet wird, zugegriffen wird: <ip> = localhost) und <port> = der Port, unter dem Tomcat erreichbar ist (Standard: 8080) (Je nach Konfiguration kann die URL abweichen, siehe dazu Konfiguration des Tomcat).

Bevor Sie sich mit dem Einstieg in das EOMS-Core beschäftigen, sollten Sie ein grundlegendes Verständnis über die Funktionsweise des EOMS haben.

Anmelden

Wenn Sie den EOMS-Core-Monitor in Ihrem Browser aufrufen werden Sie zunächst gebeten, sich anzumelden.

Abbildung A: Anmeldung

Im Auslieferungszustand des EOMS lautet der Benutzername 'eoms' und das Passwort 'eoms'.

Sie erhalten nur bei korrekten Anmeldedaten Zugriff auf den EOMS-Core-Monitor.

Es wird empfohlen, Benutzername und Passwort nach der Installation zu ändern und nicht die Standardkennung zu verwenden. Seit Version 1.3 werden Passwörter als Hashes gespeichert. Nutzer werden in der eoms.invoker.authentication an gelegt.

Seit der Version 1.3 können Sie neben Englisch auch Deutsch als Applikationssprache auswählen *Abb A (1)*. Standardmäßig ist die Sprache Deutsch eingestellt. Beachten Sie, dass nur die Oberfläche in deutscher Sprache dargestellt wird, interne Variablenwerte des EOMS aber in der Regel in Englisch (da diese direkt aus dem System extrahiert werden).

Falls trotz korrekter Anmeldedaten der Fehler "Could not connect to server" auftritt besteht womöglich ein Problem mit Ihrer EOMS-Core Konfiguration. Dies kann an einer korrupten Datenbank, falscher Konfiguration oder nicht gestartetem Datenbankserver liegen. Konsultieren Sie in diesem Fall die Log-Files des EOMS-Core *(%EOMS-CORE_HOME%/logs)*. Beachten Sie auch den Eintrag hierzu im Kapitel Fehlerbehandlung.

Nutzeroberflaeche und Navigation

Nachdem Sie sich erfolgreich angemeldet haben ist es an der Zeit, sich mit der Benutzeroberfläche vertraut zu machen. Seit Version 1.3 ist die Benutzeroberfläche auch in Deutsch verfügbar. In allen folgenden Abbildungen ist stets Deutsch als Applikationssprache gewählt. Die entsprechenden englischen Bezeichnungen werden, wo nötig, in eckigen Klammern mit angegeben.

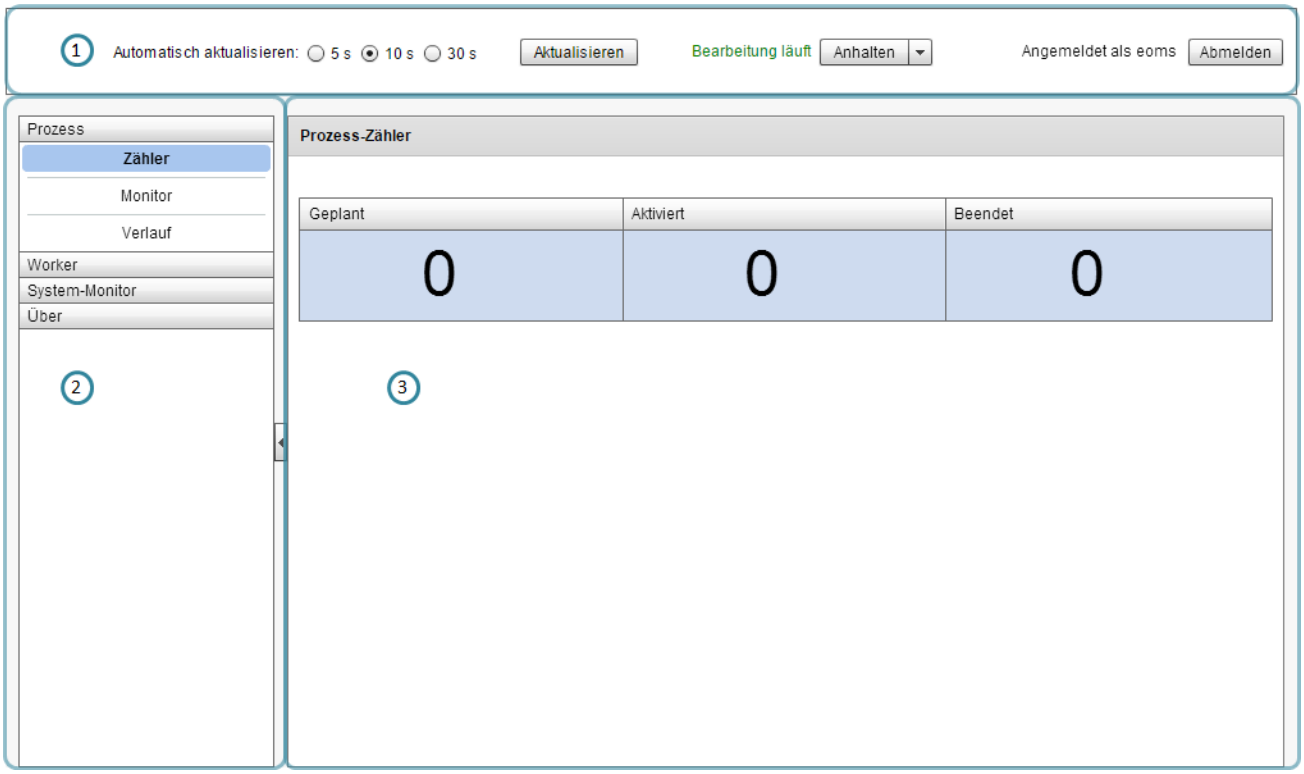


Abbildung A: Benutzeroberfläche des EOMS-Core-Monitors.

Das Layout der EOMS-Core-Monitor Oberfläche gliedert sich in 3 Bereiche:

(1) Die obere Informationsleiste

- Die obere Informationsleiste enthält grundlegende Einstellungsmöglichkeiten.

(2) Die Navigationsleiste

- Mit Hilfe der Navigationsleiste lassen sich die verschiedenen Funktionalitäten des EOMS-Core-Monitors anwählen.

(3) Das Inhaltsfenster

- Im Inhaltsfenster wird die in der Navigationsleiste ausgewählte Funktionalität angezeigt

In diesem Kapitel werden die obere Informationsleiste sowie die Auswahlmöglichkeiten der Navigationsleiste kurz vorgestellt. Wie Sie mit den Funktionen arbeiten können lernen Sie im Kapitel Bedienung.

Sie können das Layout der Fenster im EOMS-Core-Monitor mit

und

anpassen. Außerdem können Sie die Anordnung von Unterfenstern (*hier: Geplant [scheduled], Aktiviert [activated], Beendet [terminated]*) verändern, indem Sie das gewünschte Fenster mit gedrückter Maustaste an eine neue Position in der Anordnung verschieben.

Die GUI des EOMS-Core-Monitors ist in Adobe Flash Flex implementiert. Sie benötigen daher den Adobe Flash Player.

Die obere Informationsleiste

Sämtliche Einstellungs- und Verwaltungsmöglichkeiten, die Sie über die grafische Oberfläche haben, werden in der oberen Informationsleiste angezeigt. Da die grafische Oberfläche als Überwachungs- und Steuerungsmonitor des EOMS konzipiert ist und nicht als Konfigurationsoberfläche, haben Sie als Nutzer hier bewusst wenig Konfigurationsmöglichkeiten. Die Einstellungen beschränken sich auf die Verwaltung der aktuellen Sitzung.

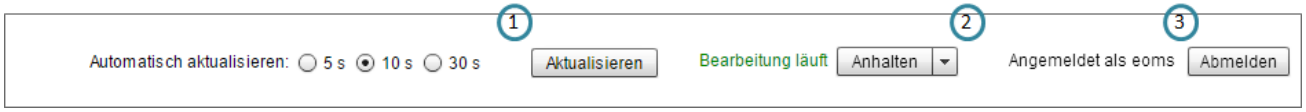


Abbildung A: Obere Informationsleiste.

Folgende Funktionen werden über die obere Informationsleiste bedient:

1. Die Autorefresh-Funktion

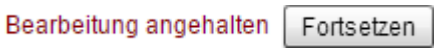
Legt fest, in welchen Zeitintervallen die Informationen im EOMS-Core-Monitor aktualisiert werden sollen (5 | 10 | 30 Sekunden). Eine Autorefresh-Zeit von 10 Sek. bedeutet, dass die Ansichten im EOMS-Core-Monitor alle 10 Sekunden aktualisiert werden. Sie können eine Aktualisierung auch mit

[Refresh] erzwingen.

2. Verarbeitung anhalten

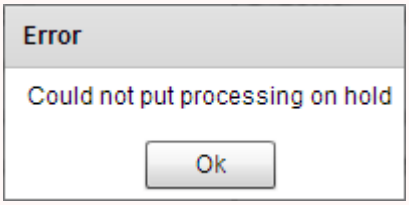
Sie können die Verarbeitung von Jobs mit Hilfe der Option ***Anhalten [Hold on]*** oder ***Jetzt Anhalten [Hold on now]*** anhalten. Das EOMS stoppt dann die Weiterleitung von Jobs an die Worker, es werden also keine neuen Jobs aktiv (bei "Jetzt anhalten" wird der Stopp erzwungen, d.h. auch Jobs, die gerade verarbeitet werden (Status "aktiviert"), werden unterbrochen):

Stoppen Sie die Verarbeitung mit ***Anhalten*** oder ***Jetzt anhalten***.



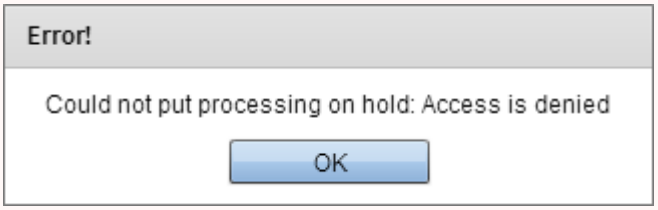
Die Verarbeitung ist gestoppt. Setzen Sie die Verarbeitung mit "Fortsetzen" *[Resume]* fort.

Falls folgender Fehler auftritt:



kann die Verarbeitung zum momentanen Zeitpunkt nicht angehalten werden. Warten Sie einige Sekunden und versuchen es dann erneut. Kann die Verarbeitung weiterhin nicht angehalten werden, starten Sie den EOMS-Core-Monitor neu.

Falls folgender Fehler auftritt:



verfügen Sie eventuell nicht über die nötigen Rechte (nur Nutzer mit ADMIN-Rechten dürfen Prozesse anhalten / fortsetzen). Konsultieren Sie Ihren Systemadministrator.

3. Abmelden

Verwenden Sie Abmelden *[Logout]*, um sich wieder vom EOMS-Core-Monitor abzumelden und zum Anmeldebildschirm zurückzugelangen. Ihnen wird außerdem angezeigt, mit welchem Benutzer Sie angemeldet sind.

Die Navigationsleiste

Die Navigationsleiste enthält alle Funktionen, die der EOMS-Core-Monitor bereitstellt. Über sie wählen Sie die gewünschte Funktion, die dann im Inhaltsfenster rechts angezeigt wird, aus.

Die eigentlichen Funktionen und ihr Umfang werden im Abschnitt Bedienung erklärt. Hier soll nur ein Überblick gegeben werden.

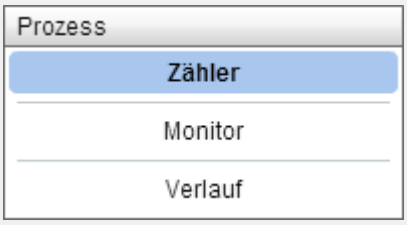
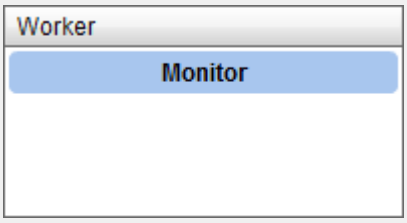
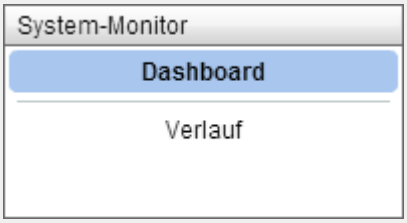
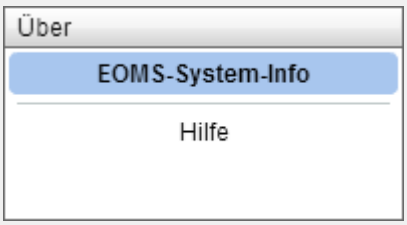
Die Funktionen des EOMS-Core-Monitors teilen sich in vier Kategorien auf:

- **Prozess *[Process]***
- **Worker**
- **System-Monitor**
- **Über *[About]***

Die Navigationsleiste lässt sich mit

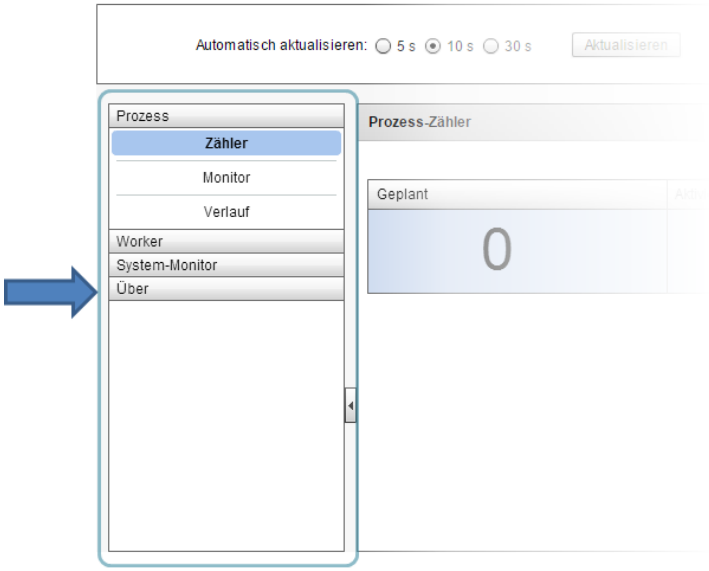
ein- und ausblenden.

Folgende Übersicht listet alle Funktionen mit einer Kurzbeschreibung auf, die im EOMS-Core-Monitor verfügbar sind:

	▪ **Zähler *[Counter]*** ▪ **Monitor *[Monitor]*** ▪ **Verlauf *[History]***
	▪ **Monitor**
	▪ **Dashboard** ▪ **Verlauf *[History]***
	▪ **EOMS Info** ▪ **Hilfe *[Help]***

Bedienung

Bedient wird der EOMS-Core-Monitor über die Navigationsleiste links. Dort finden Sie alle Monitoringfunktionen des EOMS-Core-Monitors.



Im Inhaltsfenster wird die jeweils ausgewählte Funktion angezeigt. Auf den folgenden Seiten werden die einzelnen Funktionen erklärt:

- **Prozessüberwachung (Prozess-Tab)** ***[Process]***
- **Workerüberwachung (Worker-Tab)**
- **System-Monitor**
- **Eigenschaften (Über-Tab)** ***[About]***

Prozesse

Der Prozess-Tab *(im EOMS 'Prozess')* dient zur Überwachung und Verwaltung der zu verarbeitenden Jobs.

Wählen Sie dazu in der Die Navigationsleiste 'Prozess'. Im Untermenü können Sie nun eine der zugehörigen Funktionalitäten auswählen:

1. Prozess-Zähler *[Counter]*
    - Zur Anzeige wartender, aktiver und terminierter Prozesse.
2. Prozess-Monitor *[Monitor]*
    - Zur Verwaltung und Überwachung von Prozessen.
3. Prozess-Verlauf *[History]*
    - Für eine grafische Darstellung (Historie) der Prozessverarbeitung.

Zaehler Prozess-Tab

Der Zähler *[Counter]* zeigt Ihnen die aktuelle Anzahl an wartenden (**GEPLANT *[SCHEDULED]***), aktiven (**AKTIVIERT / *[ACTIVATED]***) und verarbeiteten (**BEENDET / *[TERMINATED]***) Jobs an.

Wählen Sie in der Navigationsleiste 'Prozess' - 'Zähler' *['Process' - 'Counter']*.

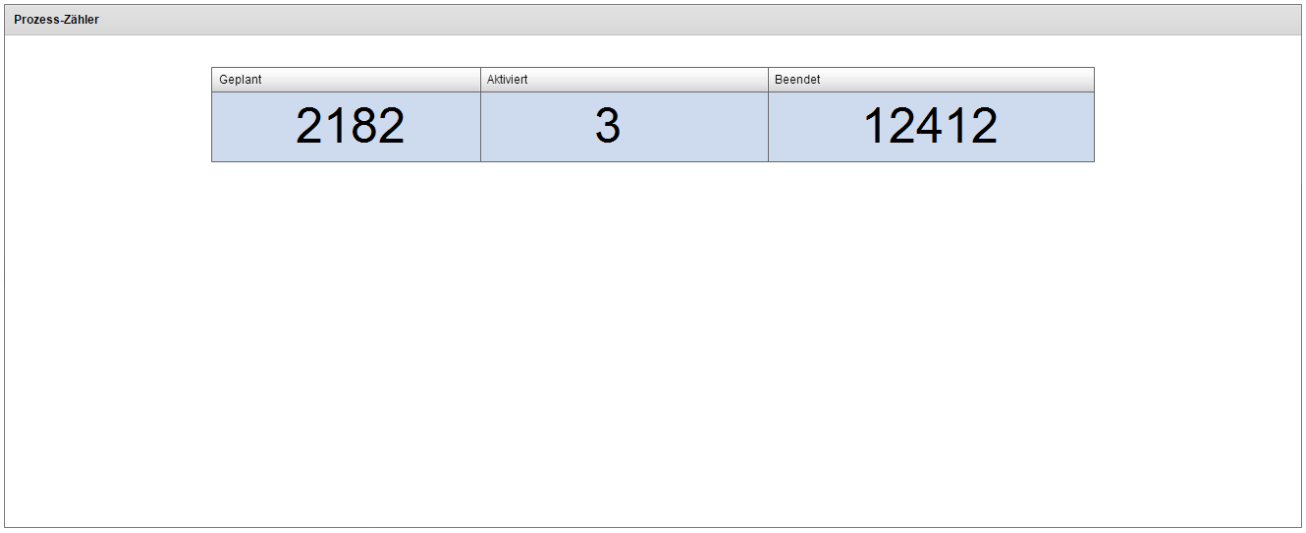


Abbildung A: EOMS-Core-Monitor mit geöffnetem Prozess-Counter (Process Counter).

GEPLANT [SCHEDULED]

Zeigt die Anzahl der Prozesse an, die sich in Warteschleife befinden und auf Vergabe an einen Worker warten. Diese trifft auf Jobs zu, die im EOMS-Core registriert wurden, aber noch von keinem Worker bearbeitet werden.

AKTIVIERT / *[ACTIVATED]*

Zeigt die Anzahl der Prozesse an, die momentan durch einen Worker bearbeitet werden. Die Anzahl simultan möglicher aktiver Prozesse ist durch die Anzahl arbeitender Worker und deren erlaubter Anzahl simultaner Prozesse limitiert. Bsp: Bei 8 aktiven Worker, die jeweils 4 Prozesse gleichzeitig auf ihrem Server ausführen dürfen, können max. 32 Prozesse aktiv sein.

BEENDET / *[TERMINATED]*

Wurde die Verarbeitung eines aktiven Jobs durch den Worker abgeschlossen wird dieser Job im EOMS-Core auf terminiert gesetzt. Auftrags-Systemen (z.B. Spooler) wird dadurch signalisiert, dass die Bearbeitung des Jobs abgeschlossen ist und die Ergebnisdaten zur Verfügung gestellt wurden.

Terminierte (beendete) Jobs werden nur eine gewisse Anzahl Stunden gespeichert und dann gelöscht (siehe Konfiguration). Es wird hier also nur die Anzahl der terminierten Jobs aus den letzten x Stunden angezeigt.

Dass ein Job terminiert hat bedeutet nicht, dass er auch erfolgreich abgeschlossen wurde. Es bedeutet nur, dass die Bearbeitung des Jobs abgeschlossen ist. Tritt ein Fehler während der Bearbeitung auf wird der Job ebenfalls terminiert. Ob der Job erfolgreich abgeschlossen wurde oder ob Fehler aufgetreten sind müssen Sie im Prozess-Monitor überprüfen.

Monitor - Prozess-Tab

Der Prozess-Monitor ist das umfangreichste Tool zur Jobüberwachung im EOMS-Monitor. Jobs werden hier mit sämtlichen Statusinformationen aufgelistet.

Wählen Sie in der Navigationsleiste 'Prozess' - 'Monitor' *['Process' - 'Monitor']*.

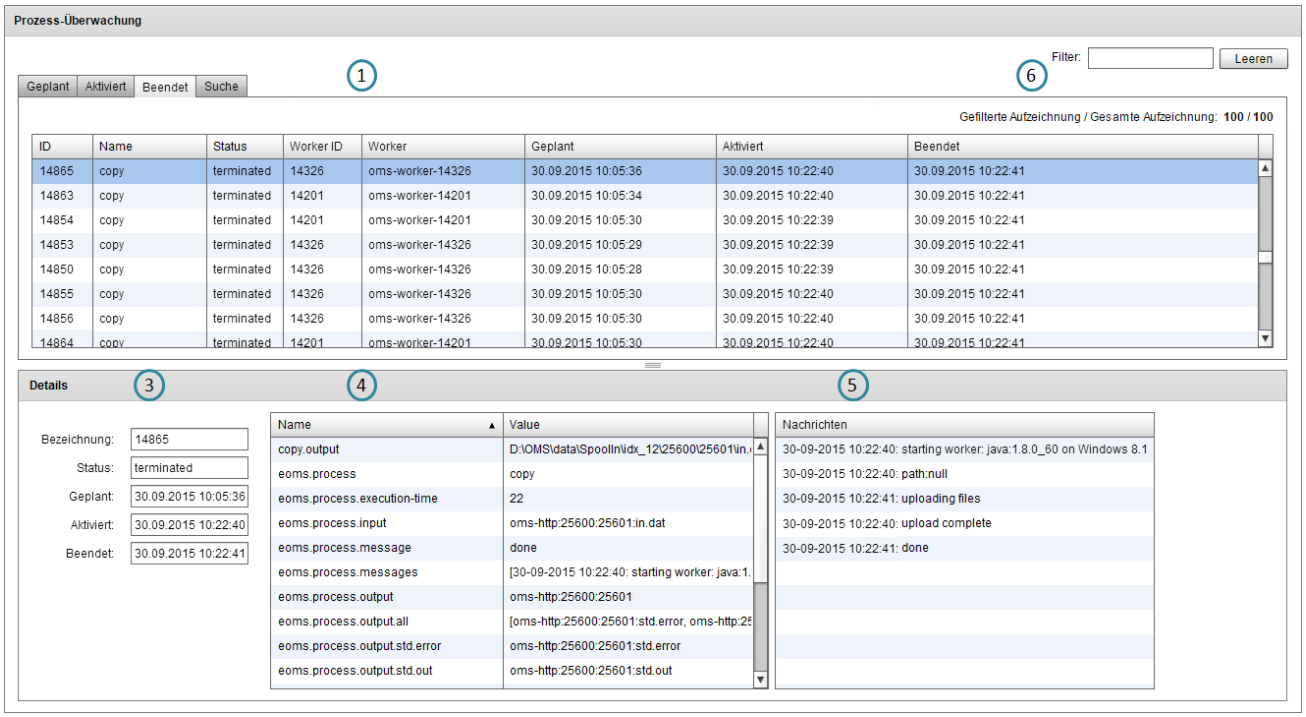


Abbildung A: EOMS-Core-Monitor mit geöffnetem Prozessmonitor (Process Monitor).

Jobs werden im Prozess-Monitor nach ihrem Status eingeteilt. Aufgelistet werden entweder nur geplante, aktive oder beendete Jobs *(1)*. Auf die Suche wird hier detailliert eingegangen.

Wählen Sie in *(1)* ***GEPLANT [SCHEDULED]***, ***AKTIVIERT [ACTIVATED]*** oder **BEENDET** ***[TERMINATED]*** aus. Ihnen werden dann in Liste *(2)* alle Jobs mit dem ausgewählten Status angezeigt.

Die Jobliste

Die Jobliste *(2)* enthält alle Jobs mit dem ausgewählten Status *(bei der Suchfunktion alle Jobs mit dem Suchmuster)*. Jobs in der Jobliste haben folgende Attribute:

ID	Die eindeutig zuzuordnende ID-Nummer des Jobs. Mit Hilfe der ID können Sie Jobs eindeutig identifizieren, suchen, etc.
NAME	Der Name des auszuführenden / ausgeführten Prozesses, der auf dem Worker ausgeführt werden soll / ausgeführt wurde. Dieser Prozess muss in der .RCML-Datei des Workers definiert sein.
STATUS	Der Status des Jobs.
WORKER ID	Die ID des Workers, der den Job verarbeitet / verarbeitet hat.
WORKER	Die Kennung des Workers, der den Job verarbeitet / verarbeitet hat.
GEPLANT *[SCHEDULED]*	Der Zeitstempel, an dem der Job in den **GEPLANT**-Status gewechselt hat (im EOMS-Core eintraf).
AKTIVIERT *[ACTIVATED]*	Der Zeitstempel, an dem der Job in den **AKTIVIERT**-Status gewechselt hat (einem Worker zur Bearbeitung übergeben wurde).
BEENDET *[TERMINATED]*	Der Zeitstempel, an dem der Job in den **BEENDET**-Status gewechselt hat (im EOMS-Core eintraf).

Die Jobliste enthält nur die letzten 100 Jobs. Sie können die Anzahl der gespeicherten Jobs in der ***eoms.invoker.properties*** ändern.

Wählen Sie einen Job in Liste *(2)* aus, um in *(3-5)* detaillierte Informationen über diesen Job anzuzeigen.

Die Jobliste variiert zwischen den verschiedenen Status:

- Geplante Jobs enthalten keine "Aktiviert"- und keine "Beendet"-Spalte, da der Job ja noch nicht aktiviert oder beendet wu
- Aktive Jobs enthalten keine "Beendet"-Spalte, da der Job aktiv ist und noch nicht beendet wurde.

Details

- In *(3)* werden die Informationen, die Sie bereits Liste *(2)* entnehmen können, noch einmal für den spezifisch ausgewählten Job wiederholt.

(3) enthält stehts aktuelle Informationen, während die Informationen in der Jobliste *(2)* erst aktualisiert werden, wenn der Job in den nächsten Status verschoben wird (Jobs werden nicht sofort in die passende Liste verschoben sobald sich ihr Job ändert, sondern erst bei einem *Refresh*, während die Informationen in *(3)* bei der Auswahl des Jobs in *(2)* abgerufen werden.

D.h.: Ist bspw. ein Job laut *(2)* **GEPLANT *[SCHEDULED]*** und auch in der Liste der **GEPLANT**-Jobs eingetragen, so ist es durchaus möglich dass sich der Status des Jobs seit dem letzten *Aktualisieren [Refresh]* schon verändert hat. Er wird dann in *(3)* schon mit Status **AKTIVIERT *[ACTIVATED]*** oder gar **BEENDET *[TERMINATED]*** angezei gt wird. Der Job wird dann demnächst in die passende Liste verschoben werden. Entnehmen Sie daher den aktuellen Status eines Jobs wegen dieser Verzögerung immer *(3)*.

- In *(4)* werden die Systemvariablen des Jobs aufgelistet. Im Folgenden sind die standard eoms.-Variablen aufgelistet und beschrie

eoms.process	Der Prozess, der auf dem Worker ausgeführt werden soll (z.B. copy, ReportWriter). Entspricht 'Name' in *(2)*.
eoms.process.execution-time	Gibt an, wie viel Zeit (in ms) der Worker zur Durchführung des Prozesses benötigt hat.
eoms.process.input	
eoms.process.message	Enthält die aktuellste Message vom Worker. Messages zeigen wichtige Informationen während der Verarbeitung an und dokumentieren die einzelnen Schritte des Workers während der Jobverarbeitung. Welche Messages ein Worker an den Core sendet wird in seiner .rcml definiert. Keine Fehlermeldung in ***eoms.process.mess age*** bedeutet nicht, dass der Job erfolgreich beendet wurde *(siehe **eoms.process.runtime-error**)*. Es bedeutet nur, dass durch die .rcml keine Fehlermeldung übertragen wurde.
eoms.process.messages	Enthält alle Messages, die während der Verarbeitung vom Worker ausgegeben wurden. Der Inhalt der Variable wird formatiert in *(5)* dargestellt.
eoms.process.output	
eoms.process.output.all	

eoms.process.output.std.error	
eoms.process.output.std.out	
eoms.process.return-code	**1** für erfolgreich beendete Jobs, **0** für nicht erfolgreich beendete Jobs (Nur wenn hier 1 gesetzt ist wurde der Job erfolgreich ausgeführt).
eoms.process.runtime-error	**true** falls während der Prozessausführung durch den Worker Fehler aufgetreten sind, **false** falls nicht. Selbst wenn die Variable auf false gesetzt ist bedeutet das nicht, dass der Job erfolgreich beendet wurde, sondern nur, dass während der Abarbeitung des Prozesses (eoms.process) keine Fehler aufgetreten sind. Es kann aber auch z.B. in der Nachbereitung oder dem Transport der Ergebnisdaten zu Störungen gekommen sein. Sollte es zu Fehlern gekommen sein, die nicht während des Prozesses durch den Worker aufgetreten sind, so ist eoms.process.runtime-error = false aber eoms.process.return-code = 0.
eoms.process.worker	Die Kennung des Workers, der den Job bearbeitet / bearbeitet hat. Entspricht 'Worker' in *(2)*.
eoms.session	Enthält Informationen zur Sitzung während der der Job den aktuellen Status wechselte. Der Teil vor dem @ gibt den Typ der Sitzung an, z.B. "eoms-polling-session" falls die Kommunikation zwischen EOMS und Spooler direkt *(polling)* stattfand und nicht über ActiveMQ *(siehe EOMS-Aufbau)*. Der Teil hinter dem @ gibt die ID der Session an. Bei jeder neuen Session des EOMS wird der Session eine einmalige ID zur Identifizierung zugewiesen. Eine Session im EOMS beginnt mit dem Start des EOMS (Start des Apache Tomcat Servers) und endet mit dem Herunterfahren des EOMS (Stopp des Apache Tomcat Servers).

Beachten Sie, dass nicht alle Systemvariablen während jedes Status verfügbar sind. ***eoms.process.execution-time*** gibt beispielsweise an wie lange es gedauert hat, den Prozess auf dem Worker auszuführen. Diese Systemvariable macht also erst Sinn, nachdem der Job verarbeitet wurde und ist daher nur bei terminierten Jobs gesetzt (auch wenn der Job noch als aktiv gekennzeichnet ist kann es sein, dass der Job bereits terminiert ist und deshalb schon alle Variablen gesetzt sind).

In *(4)* werden auch die Variablen angezeigt, die Sie beim Aufruf des EOMS durch z.B. den Spooler mit übergeben haben.

In *(5)* werden alle Nachrichten, die in ***eoms.process.messages*** gespeichert sind, formatiert und in chronologischer Reihenfolge dargestellt.

Sortieren und Filtern

Klicken Sie in der Jobliste *(2)* auf eine Spaltenüberschrift, um die Jobliste alphabetisch nach den Einträgen dieser Spalte zu sortieren. Durch mehrfaches Klicken bestimmen Sie, ob absteigend

oder aufsteigend

sortiert werden soll.

Sie können in *(6)* die Jobliste nach einem Suchmuster filtern. Es werden in der Jobliste dann nur Jobs angezeigt, die den in die Filtermaske eingegebenen Wert in einer der Spalten (außer der Spalte "Status") enthalten. Löschen Sie die Eingabe aus der Suchmaske, um wieder alle Jobs anzuzeigen (oder klicken Sie auf

[Clear]).

Suche

Die Suche erlaubt es Ihnen Jobs zu finden, die einem Suchmuster entsprechen (unabhängig vom Status). Wechseln Sie dazu in *(1)* Suche *[Search].* Tragen Sie dann in das Suchfeld den Wert ein, nach dem Sie suchen möchten. Es werden dann alle Jobs aufgelistet, auf die das Suchmuster passt. Ein Job passt auf ein Suchmuster, wenn es dieses in einer der Spalten enthält (hier wird auch die Spalte "Status" berücksichtigt, da die Suche statusübergreifend geschieht). Sie können jetzt durch auswählen eines Jobs wie gewohnt seine Details anzeigen lassen.

Sie können das EOMS anweisen, einem Job vorrangige Priorität einzuräumen. Das EOMS versucht dann, diesen Job so schnell wie möglich zu terminieren. Wählen Sie dazu den gesuchten Job in der Jobliste *(in **GEPLANT**, **AKTIVIERT** oder Suche)* aus und klicken Sie auf

[Terminate selected process]. Das EOMS versucht dann, diesen Job zu beenden. Ihnen wird angezeigt, ob dies erfolgreich war.

Verlauf - Prozess-Tab

Die Verlaufsansicht *[History]* stellt grafisch den Verlauf der Prozessabarbeitung im EOMS während den letzten 15 Minuten dar. Angezeigt wird:

- Die Anzahl der wartenden Jobs (**GEPLANT *[SCHEDULED]***).
- Die Workerauslastung: Anzahl der laufenden Prozesse (**AKTIVIERT *[ACTIVATED]***).
- Die Anzahl der terminierten Jobs (**BEENDET *[TERMINATED]***).

Wählen Sie in der Navigationsleiste 'Prozess' - 'Verlauf' *['Process' - 'History']*.

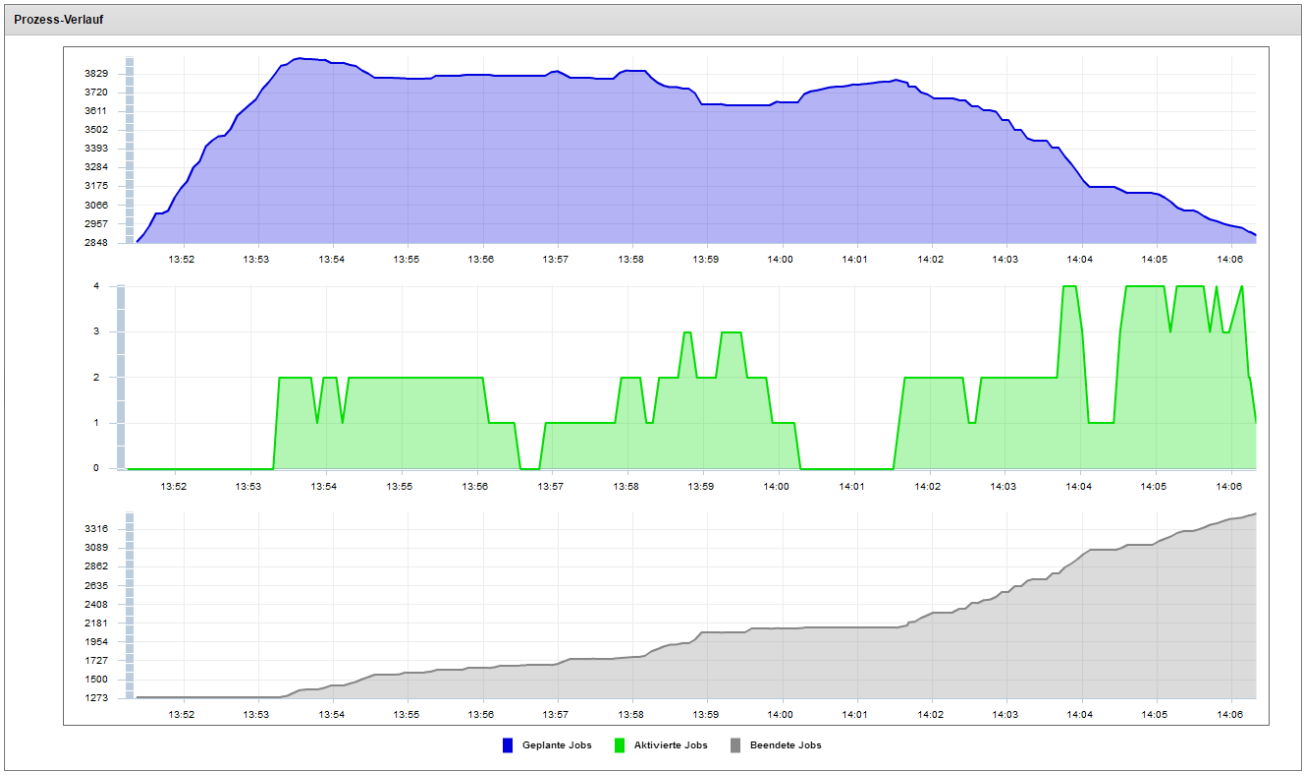
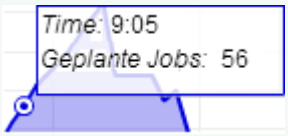


Abbildung A: EOMS-Core-Monitor mit geöffnetem Verlauf.

Beachten Sie, dass es sich bei den Zeitangaben um die Zeit auf dem Hostrechner des EOMS-Core handelt.

Der Verlauf aktualisiert sich bei jedem Autorefresh (oder manuellen Refresh). Die Genauigkeit der Verlaufskurve hängt deshalb von der Autorefreshzeit ab.



Platzieren Sie die Maus über der Verlaufskurve, um die exakte Anzahl Jobs zu einem Zeitpunkt abzufragen.

Worker-Monitor

Der Worker-Monitor verwaltet die aktiven Worker, die sich am EOMS-Core registriert haben.

Wählen Sie in der Navigationsleiste 'Worker' - 'Monitor' *['Worker' - 'Monitor']*.

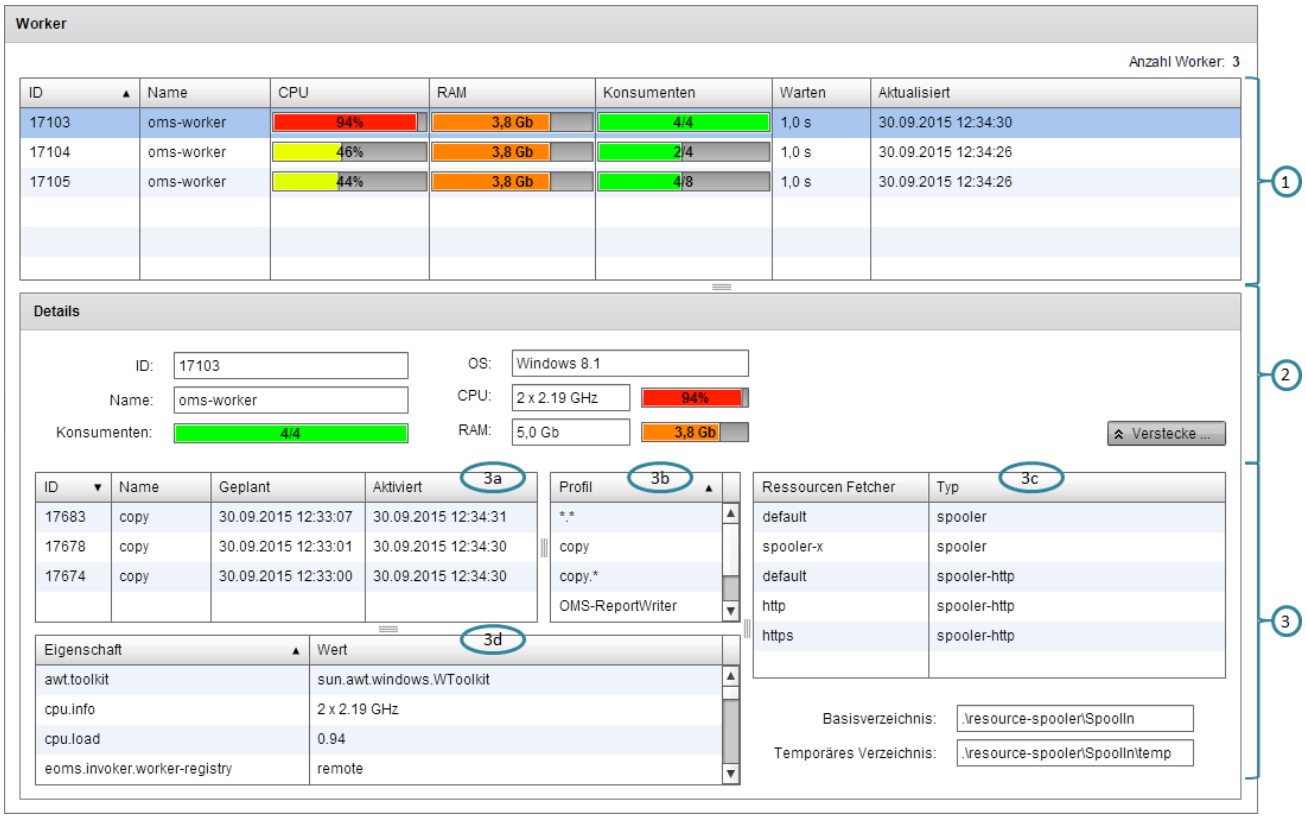


Abbildung A: EOMS-Core-Monitor mit geöffnetem Worker-Monitor.

Die Workerliste

Die Workerliste *(1)* zeigt alle Worker an, die am EOMS-Core registriert sind und sich nicht ausgetragen (wieder abgemeldet) haben. Dass ein Worker in der Workerliste angezeigt wird / am EOMS-Core angemeldet ist heißt nicht, dass er noch fehlerfrei arbeitet. Meldet sich ein Worker vom EOMS-Core ab wird dies erst beim nächsten Autorefresh / Refresh registriert und der Worker erst dann aus der Liste entfernt. Dies kann unter Umständen auch einige Zeit in Anspruch nehmen.

> Abgestürzte / durch Laufzeitfehler gestoppte Worker werden nach einer gewissen Zeit vom EOMS-Core eigenständig als funktionsunfähig erkannt. Noch aktive Jobs dieses Workers werden automatisch erneut an andere Worker übergeben. Es kann also durch den Absturz eines Workers nicht zum Datenverlust kommen.

Worker werden mit folgenden Attributen in der Workerliste angezeigt:

ID	Die eindeutig zuzuordnende ID-Nummer des Workers. Mit Hilfe der ID können Sie Worker eindeutig identifizieren.
NAME	Der Name des Workers (automatisch vergeben). Dem Namen können Sie auch den Typ (OMS / EOMS) des Workers entnehmen (hier: OMS-Worker).
CPU	Die CPU-Auslastung des Hosts, auf dem der Worker läuft (die Gesamtauslastung, nicht die vom Worker beanspruchte Rechenleistung).
RAM	Der belegte Arbeitsspeicher des Hosts, auf dem der Worker läuft (die Gesamtauslastung, nicht der vom Worker beanspruchte Speicherplatz). Die Breite und Farbe des Balkens (grün - gelb - orange - rot) spiegelt gleichzeitig die Auslastung im Verhältnis zum vorhandenen RAM-Speicher dar (grau = freier RAM-Speicher).
KONSUMENTEN *[CONSUMERS]*	Anzahl der Konsumenten (Consumers) des Workers. Pro Konsument kann 1 Prozess ausgeführt werden. Bei 1 Konsumen t kann ein Worker also immer nur 1 Prozess gleichzeitig bearbeiten, bei 4 Konsumenten 4 Prozesse simultan. Die erste Zahl gibt die Anzahl der aktiven Konsumenten an (grüner Balken), die 2. Zahl die Anzahl der maximalen Konsumenten dies es Workers.
WARTEN *[SLEEP]*	Die Zeit, die ein freier Worker wartet, bis er erneut am EOMS-Core nach neuen Jobs anfragt. Kann in der worker.properties bzw. in der eoms-input.properties konfiguriert werden.
AKTUALISIERT *[UPDATED]*	Zeitstempel, wann die Daten zuletzt aktualisiert wurden.

Details

Unterhalb der Workerliste werden Ihnen die Details zum ausgewählten Worker angezeigt *(2-4)* (bei aktiver Prozessverarbeitung kann das Laden der Details eventuell etwas Zeit in Anspruch nehmen). Der Ansicht können Sie folgende Informationen entnehmen:

- Detailliertere Angaben zu Worker und seinem Host *(2)*. Neben den bereits in der Workerliste enthaltenen Attributen ID, Name und Konsumentenanzahl werden hier noch aufgeführt:

OS	Zeigt Ihnen das Betriebssytem des Hostrechners an, auf dem der Worker läuft.
CPU	Wie in der Workerliste auch wird hier die CPU-Auslastung des Hostrechners angegeben, allerdings wird hier zusätzlich die Anzahl an CPU-Kernen und deren Takt angezeigt *(hier: 8 Kerne mit jeweils 2,3 Ghz)*. Pro Kern sollte in der Regel nicht mehr als 1 Konsument (nur für OMS-Worker) erlaubt werden. Bei einem Achtkernprozessor sollten Sie also die Anzahl an Konsumenten auf 8 begrenzen *(siehe Konfiguration von Workern)*.
RAM	Wie in der Workerliste auch wird hier die RAM-Auslastung des Hostrechners angegeben, allerdings wird hier zusätzlich die RAM-Größe des Hostrechners angezeigt.

- Laufende Prozesse des Workers und seine List of Tasks *(3a + 3b)*:
  - Die Tabelle *(3a)* enthält alle aktiven Prozesse auf dem ausgewählten Worker. Dabei ist die Anzahl der aktiven Prozesse d die Anzahl maximaler Konsumenten begrenzt. Die Anzahl der aktiven Prozesse entspricht der Anzahl der aktiven Konsur Prozess pro Konsument). In *(3a)* werden die ID und Name des Prozesses sowie der Zeitstempel des Statuswechsels zu ***HEDULED]*** und zu **AKTIVIERT** ***[ACTIVATED]*** angezeigt *(siehe Jobmonitor)*.
  - Die List of Task *(Profil(e))* in *(3b)* (ausführbare Prozesse auf dem Worker, siehe Aufbau des EOMS, Worker) des Worker *b)*. Der Worker in *Abb. A* kann folgende Prozesse ausführen:
    a. Prozesse, deren Name mit copy beginnt (auch copy selbst) (copy, copy.*),
    b. Prozesse, deren Name mit rw beginnt (auch rw selbst) (rw.*),
    c. Alle Prozesse (*.*)

    .* ist eine sog. Wildcard (beliebiges Zeichen beliebig oft)

Was es mit der List of Task und Prozessen auf Workern auf sich hat und wie Sie Prozesse für Worker definieren können erfahren Sie im Abschnitt über Worker.

Neu in Version 1.3!

- Ressourcen Fetcher Liste *(3c):*
  - Seit Version 1.3 wird Ihnen auch eine Liste mit den Ressource Fetchern des Workers angezeigt. Diese können in der eor .properties konfiguriert werden. Standardmäßig sind nur default-fetcher eingerichtet. In *Abb. A* wurden zusätzliche Fetche

Ressourcen Fetcher	Typ
default	spooler
spooler-x	spooler
default	spooler-http
http	spooler-http
https	spooler-http

Neben den default-Fetchern verfügt dieser Worker noch über einen "spooler-x"-Fetcher, einen "http"-Fetcher und einen "l (Beim Festlegen der Namen sind Sie völlig frei, der Name muss also nicht zwingend den Zweck beschreiben). In der rech Ihnen der zugehörige Typ angezeigt ("spooler-http" bedeutet z.B., dass der Fetcher per http Ressourcen für den Spooler

Unter der Fetchertabelle werden Ihnen, falls vorhanden, die für diesen Fetcher eingerichteten Verzeichnisse angezeigt (Ablageverzeichnis für Daten, temporäres Verzeichnis für Zwischenspeicherung von Daten). Ändern können Sie die Verz direkt in der eoms.invoker.client.properties. Eine Beschreibung für Administratoren, wie man Fetcher in der Konfiguration anlegt, finden Sie hier.

- Systemvariablen des Workers *(3d):*
  - In *(3d)* werden die Systemvariablen des Workers aufgelistet. Diese Systemvariablen entsprechen in weiten Teilen den im Prozess-Monitor erläuterten Variablen, allerdings beziehen sich die Werte hier auf den Hostrechner des Workers, während Eigenschaftsübersicht die Werte des Hostrechners, auf dem das EOMS-Core läuft, angezeigt werden. Nicht alle Variable den Anwendern von Bedeutung, weswegen im Folgenden auch nicht alle Variablen beschrieben werden.

> In Version 1.3 neu hinzugekommene Variablen sind orange markiert.

Eigenschaftsname	*Beschreibung*	*Standardwert*
awt.toolkit	Gibt das verwendete awt *(abstract window toolkit für die GUI)* an.	—
cpu.info	Gibt die Anzahl Prozessorkerne und ihre Taktung an (*kerne***X***Taktung*). Wird auch in *(2)* angezeigt.	—
cpu.load	Der Load des Prozessors.	—
eoms.invoker.worker-registry	Gibt an, über welche Schnittstelle der Worker Jobs empfangen und versenden soll (Standard (remote) oder REST-Schnittstelle).	**remote** (rest / remote)

eoms.invoker.worker-service	Gibt an, welche Schnittstelle der Worker ansprechen soll, um sich am Core zu registrieren (Standard (remote) oder REST-Schnittstelle).	**remote** (rest / remote)
file.encoding	Verwendete Zeichenkodierung.	Unter Windows: **Cp1252** *(ISO 8859-1 CP1252 West Eu*
file.encoding.pkg	Package für die Zeichenkodierung.	**sun.io**
file.separator	Trennzeichen für Dateipfade.	****
https.protocols	Für HTTPS-Kommunikation verwendete Protokolle.	**SSLv3, TLSv1**
java.awt.graphicsenv	—	—
java.awt.printerjob	—	—
java.class.path	Klassenpfad von Java.	—
java.class.version	—	—
java.endorsed.dirs	—	—
java.ext.dirs	Externe Verzeichnisse für den Class Loader.	—
java.home	Installationsverzeichnis des JDK/JRE.	—
java.io.tmpdir	Temporärer Ordner.	**${eoms.invoker.home}\temp**
java.library.path	Verzeichnisse für Java-Bibliotheken.	**${eoms.invoker.home}/bin** **und weitere**
java.runtime.name	Name des JRE.	—
java.runtime.version	Versionsnummer des JRE.	—
java.specification.name	—	—
java.specification.vendor	Anbieter der Javaspezifikation.	**Sun Microsystems inc.**
java.specification.version	Version der Javaspezifikation.	—
java.vendor	Anbieter der Java Distribution.	**Sun Microsystems inc.**
java.vendor.url	Homepage von java.vendor.	—
java.vendor.url.bug	Homepage des java.vendor, um einen Bug zu reporten.	—
java.version	Versionsnummer der Java Distribution	—
java.vm.info	Mode der Java Virtual Machine.	**mixed mode**
java.vm.name	Name der Java Virtual Machine.	—
java.vm.specification.name	—	—
java.vm.specification.vendor	Anbieter der Java Virtual Machine Spezifikation.	**Sun Microsystems inc.**
java.vm.specification.version	Versionsnummer der Java Virtual Machine Spezifikation.	**1**
java.vm.vendor	Anbieter der Java Virtual Machine.	**Sun Microsystems inc.**

java.vm.version	Versionsnummer der Java Virtual Machine.	—
line.separator	Trennzeichen für Zeilenumbrüche.	0
os.arch	Gibt den Prozessortyp des EOMS-Core Hosts an.	—
os.name	Gibt das Betriebssystem des EOMS-Core Hosts an.	—
os.version	Gibt die Versionsnummer des Betriebssystems an.	—
path.separator	Trennzeichen für Pfade bei Pfadangaben.	;
ram.free	Freier RAM-Speicher in Bytes.	—
ram.total	Gesamter RAM-Speicher in Bytes.	—
sun.arch.data.model	Architektur des Hostrechners.	—
sun.boot.class.path	—	—
sun.boot.library.path	\bin Verzeichnis des JDK.	**${java.home}\bin**
sun.cpu.endian	—	—
sun.cpu.isalist	wie ***os.arch***.	—
sun.desktop	wie ***os.name*** ohne Versionsnummer.	—
sun.io.unicode.encoding	Encoding für Unicode.	**UnicodeLittle**
sun.management.compiler	—	—
sun.os.patch.level	—	—
user.country	Ländercode des Hostrechners.	—
user.dir	wie ***eoms.invoker.home***.	—
user.home	Wurzelverzeichnis des Hosts.	—
user.language	Sprache auf dem Hostrechner.	—
user.name	Name des Benutzers, unter dem EOMS-Core auf dem Hostrechner ausgeführt wird.	—
user.timezone	Zeitzone des Hostrechners	—
user.variant	—	—
worker.consumer.active	Anzahl momentan aktiver Konsumenten. Entspricht der 1. Zahl in der Konsumentenanzeige in *(1)* und *(2)*.	—
worker.consumer.max	Anzahl maximaler Konsumenten. Entspricht der 2. Zahl in der Konsumenten anzeige in *(1)* und *(2)*.	**1**
worker.ip	IP-Adresse des Hostrechners, auf dem der Worker läuft.	—

worker.sleeptime	Die Zeit, die ein freier Worker wartet, bis er erneut am EOMS-Core nach neuen Jobs anfragt. Entspricht ***SLEEP*** in der Workerliste.	**1000**

System-Monitor

Der EOMS-Monitor ist seit Version 1.2.4 verfügbar.

Der System-Monitor des EOMS zeigt zahlreiche Echtzeit- und Verlaufsstatistiken über Core und Worker. Dazu gehören unter anderem Statistiken über die Datenbankanbindung, Prozesse, Worker und allgemeine Performancewerte.

Wählen Sie in der Navigationsleiste 'System-Monitor' und dann:

1. Dashboard
    - Für aktuelle Statistiken in Echtzeit.
2. Verlauf *[History]*
    - Für aufgezeichnete Statistiken zur Auswertung vergangener Performance- und Verlaufswerte.

Dashboard - System-Monitor-Tab

Das Dashboard zeigt Ihnen Echtzeitstatistiken über EOMS-Core und EOMS-Worker an.

Wählen Sie in der Navigationsleiste 'System-Monitor' - 'Dashboard'.

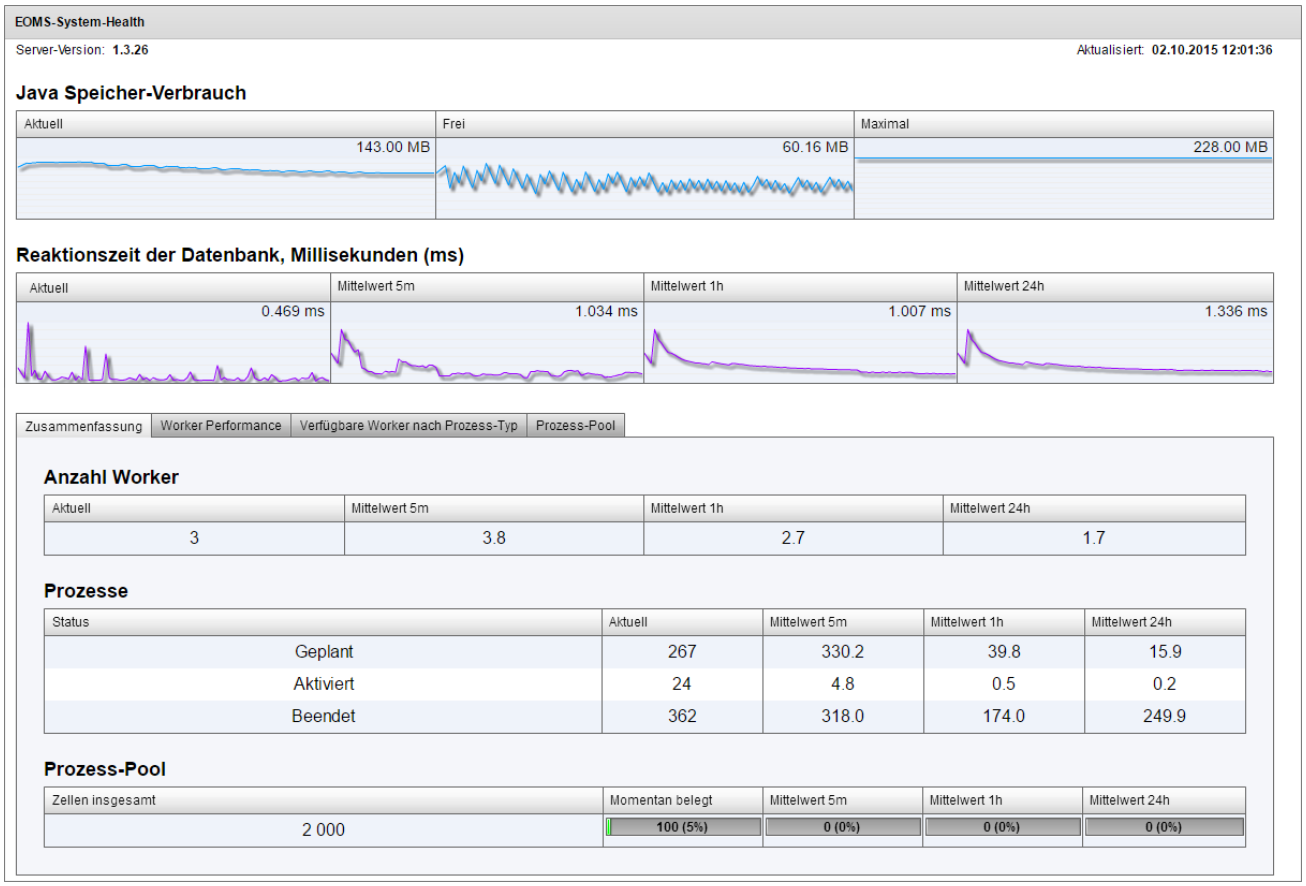


Abbildung A: EOMS-Core-Monitor mit geöffnetem Dashboard.

Java Speicher-Verbrauch *[Java memory consumption]*

Angezeigt wird der Speicherverbrauch der Java Virtual Machine (JVM), der maximale Speicherverbrauch sowie der freie, unbelegte Speicher. Der Speicherverbrauch sollte in der Regel relativ konstant bleiben, kann aber bei Lastspitzen ansteigen. Der maximale Speicherverbrauch ist konstant festgelegt und sollte in den meisten Fällen ausreichen. Der freie Speicher gibt an, wie viel Speicher noch von der JVM belegt werden kann, bis die Maximalgrenze erreicht ist. Falls öfters Speicherengpässe entstehen (Verbrauch > 90% des Maximalverbrauchs), kontaktieren Sie bitte unseren Support.

Reaktionszeit der Datenbank *[Database response time]*

Angezeigt wird die Datenbankreaktionszeit:

- Aktuell *[current response time]*

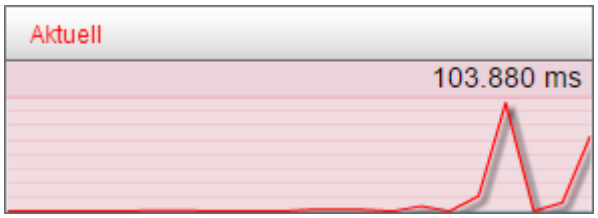
oder als durchschnittliche Reaktionszeit *[average response time]* der letzten:

- 5 Minuten
- Stunde

- 24 Stunden

Die Verlaufskurve selbst gibt den kürzlichen Verlauf der Reaktionszeit an. Weiter zurückliegende Daten finden Sie im Verlaufsmonitor.

Eine Überlastung der Datenbank resultiert in einer stark erhöhten Reaktionszeit. Stark erhöhte Reaktionszeiten werden Ihnen dadurch signalisiert, dass die Verlaufsgrafik rot hinterlegt wird:



Falls Ihre Datenbank ständig überlastet ist, sollten Sie gegebenenfalls infrastrukturelle Maßnahmen ergreifen oder unseren Support konta ktieren, da sonst eine reibungslose und vor allem zügige Prozessbearbeitung im EOMS nicht gewährleistet ist. Im schlimmsten Fall droht ein Systemausfall.

Worker- und Prozessinformationen

---

1. Zusammenfassung *[Summary]*
    - Anzahl Worker *[Workers count]*:

        Die Anzahl aktuell registrierter Worker sowie die durchschnittliche Anzahl registrierter Worker innerhalb der letzten 5 Minuten, der letzten Stunde und der letzten 24 Stunden. Der Mittelwert wird gebildet als das gewichtete arithmetische Mittel. Waren innerhalb der letzten Stunde also für 45 Minuten 8 Worker aktiv und für 15 Minuten 4 Worker, so ist der Mittelwert 7.

    - Prozesse *[Processes]*

        Die Anzahl der aktuell geplanten, aktiven und beendeten Prozesse sowie ebenfalls der jeweilige Mittelwert der letzten 5 Minuten / Stunde / 24 Stunden. Diese Werte können je nach Auslastung leicht verzögert sein.

    - Prozess-Pool *[Process Pool]*

        Die Gesamtgröße des Prozess-Pools (Anzahl der Zellen (nodes) * Zellengröße (size), kann in der eoms.invoker.properties angepasst werden), die momentane Belegung des Prozess-Pools (Gesamtanzahl Prozesse im Prozess-Pool ohne Unterscheidung von Prozesstypen, sowie prozentuale Belegung orientiert an der Gesamtprozesspoolgröße), sowie die Mittelwerte der letzten 5 Minuten / Stunde / 24 Stunden.
2. Worker Performance

    Zeigt an, wie viele Prozesse jeder registrierte Worker pro Minute aktuell bearbeitet und wie viele er in den letzten 5 Minuten / Stunde / 24 Stunden durchschnittliche pro Minute bearbeitet hat. Ist ein Worker erst seit einer Stunde aktiv, wird für die letzten 24 Stunden der Mittelwert angezeigt, den der Worker erbracht hätte, wäre er bei den geleisteten Jobs schon 24 Stunden aktiv, also z.B.: Worker ist seit 1 Stunde aktiv und hat in dieser Stunde 7200 Prozesse bearbeitet -> Mittelwert der letzten 24 Stunden: 5 Prozesse / Minute.

Prozesse pro Minute

ID	Aktuell	Mittelwert 5m	Mittelwert 1h	Mittelwert 24h
33200	90.0	20.2	11.2	11.2
33186	84.0	19.4	10.8	10.8
33177	86.0	19.6	10.9	10.9
33180	94.0	21.0	11.7	11.7

3. Verfügbare Worker nach Prozess-Typ *[Available workers by process type]*

   Zeigt an, wie viele Worker für bestimmte Prozesse (Prozesstypen) zur Verfügung stehen. Ein Worker kann nur die Prozesse bearbeiten, die in seiner List of Task definiert sind (zur Konfiguration der List of Tasks siehe hier).

   Entsprechend auch hier die Mittelwerte für 5 Minuten / 1 Stunde / 24 Stunden, die Berechnung erfolgt analog zu der von Prozessen/Minute.

Verfügbare Worker

Prozess-Name ▼	Aktuell	Mittelwert 5m	Mittelwert 1h	Mittelwert 24h
rw	4	4.0	4.0	4.0
rs	6	4.8	4.2	4.2
copy	6	4.8	4.2	4.2
OMS-ReportWriter	6	4.8	4.2	4.2

4. Prozess-Pool *[Process Pool]*

   Zeigt an, wie viele Prozesse jedes Typs im Prozess-Pool liegen. Der Prozess-Pool ist ein Zwischenspeicher für Prozesse. Das EOMS vergibt nur Jobs aus dem Prozess-Pool an Worker, nicht direkt aus der Datenbank.

Prozess-Pool

Filter	Zellen insgesamt	Momentan belegt	Mittelwert 5m	Mittelwert 1h	Mittelwert 24h
[copy.*]	250	250 (100%)	0 (0%)	0 (0%)	0 (0%)
[rw.*]	250	230 (92%)	0 (0%)	0 (0%)	0 (0%)
[any]	250	0 (0%)	0 (0%)	0 (0%)	0 (0%)

In der obigen Ansicht beträgt die Poolgröße 250 pro Prozess und der Pooltyp ist cluster. Die Parzelle für "copy"-Prozesse ist voll belegt, es können also keine weiteren copy-Prozesse aufgenommen werden, bevor nicht alte abgearbeitet werden. Beachten Sie, dass es äußerst wichtig ist, den Prozess-Pool in der eoms.invoker.properties passend für den Einsatzzweck des EOMS zu konfigurieren. Werden z.B. häufig gleiche Prozesstypen verarbeitet, sollten wenige Zellen mit großer Kapazität eingerichtet werden. Bei wenig Jobs pro Prozess-Typ, aber vielen verschiedenen Prozesstypen hingegen ist es oft effizienter, viele Zellen mit geringerer Kapazität einzurichten. Mehr dazu in der Prozess-Pool Beschreibung und der Konfigurationsbeschreibung Core.

Historische Daten und zum Teil noch umfangreichere Details bietet der Verlaufsmonitor.

Verlauf - System-Monitor-Tab

Die Verlaufsansicht ähnelt dem Dashboard, sie zeigt aber im Gegensatz zu diesem keine Echtzeit-, sondern historische (Verlaufs-)Statistiken über Ihr EOMS an.

Wählen Sie in der Navigationsleiste 'System-Monitor' - 'Verlauf' *['System-Monitor' - 'History']*.

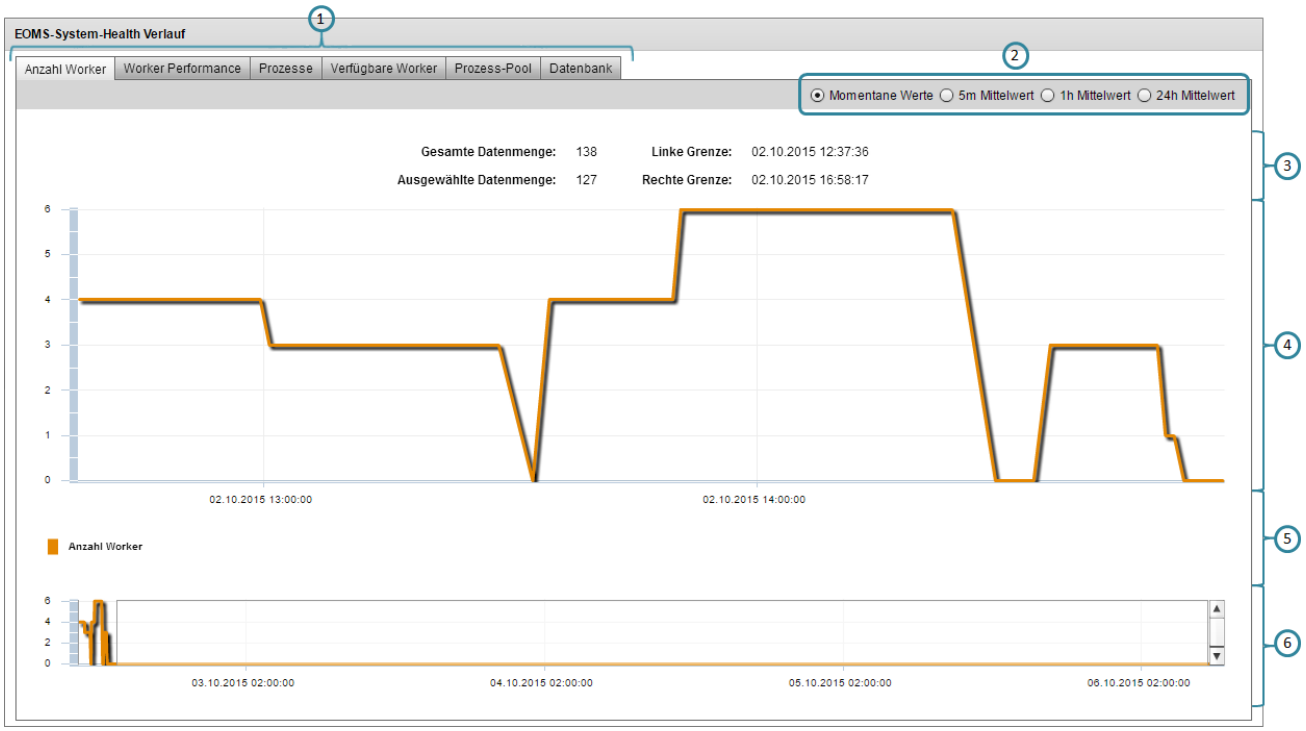


Abbildung A: EOMS mit geöffneter Verlaufsansicht (Anzahl Worker).

Der Verlaufsmonitor liefert detailliertere und zeitliche weiter zurückreichende Daten als das Dashboard. Hier können Sie genau betrachten, wie sich einzelne Performancewerte in festgelegten Zeiträumen entwickelt haben. Prinzipiell werden im Verlaufsmonitor die gleichen Eckdaten abgebildet wie im Dashboard.

In *Abb. A (1)* wählen Sie den Datensatz aus, den Sie betrachten möchten. Angezeigt wird dann die Verlaufskurve *Abb. A (4)* für diesen Datensatz, aus der Sie für jeden beliebigen Zeitpunkt der angezeigten Zeitspanne den in *Abb. A (2)* ausgewählten Wert entnehmen können: den aktuellen Wert zu jedem Zeitpunkt der Verlaufskurve (momentane Werte *[actual values]*) oder die Durchschnittswerte *[aver age]* über die vergangenen 5 min / Stunde / 24 Stunden zu diesem Zeitpunkt (Durchschnitt in den 5 min / Stunde / 24 Stunden vor diesem Zeitpunkt).

Über die Zeitleiste *Abb. A (6)* können sie festlegen, welche Ausschnitt Sie in *Abb. A (4)* betrachten möchten. Ziehen Sie dazu mit gedrückter Maustaste die linke und rechte Begrenzung *Abb. A (6a + 6b)* in *Abb. A (6)* an die gewünschte Position. Ihnen wird dann der ausgewählte Ausschnitt in *Abb. A (4)* angezeigt.

Unabhängig vom in *Abb. A (1)* ausgewählten Datensatz werden Ihnen immer zur Verlaufskurve zugehörig angezeigt *Abb. A (3)*:

- **Gesamte Datenmenge *[All data size]***: Gesamte Anzahl Zeitpunkte, zu denen Daten vorliegen.
- **Ausgewählte Datenmenge *[Selected data size]***: Anzahl abrufbarer Zeitpunkte in der aktuell dargestellten Verlaufskurve.
- **Linke / Rechte Grenze *[Left / Right boundary]***: Start- und Endzeitpunkt der aktuell dargestellten Verlaufskurve (ausgewählte G in *Abb. A (6)*).

Um Werte für einen bestimmten Zeitpunkt anzuzeigen, fahren Sie mit der Maus über die Verlaufskurve. Ihnen wird dann für den Zeitpunkt der entsprechende Wert angezeigt.

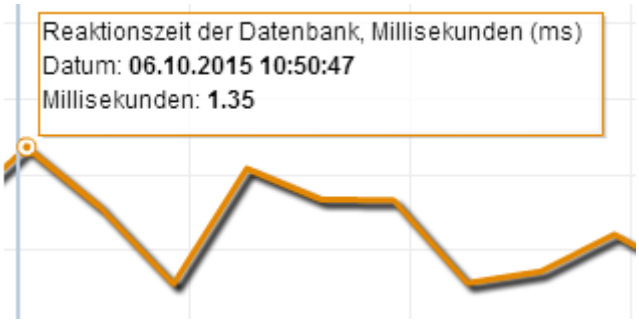


Abbildung B: Wert für einen Zeitpunkt. Hier: Datenbankreaktionszeit.

Der angezeigte Wert hängt natürlich davon ab, welchen Datensatz Sie gerade betrachten.

Sie können nicht für jeden beliebigen Punkt auf der Kurve einen Wert anzeigen lassen, da Daten nur in einem gewissen Abstand erhoben werden (max. 1 mal pro Minute).

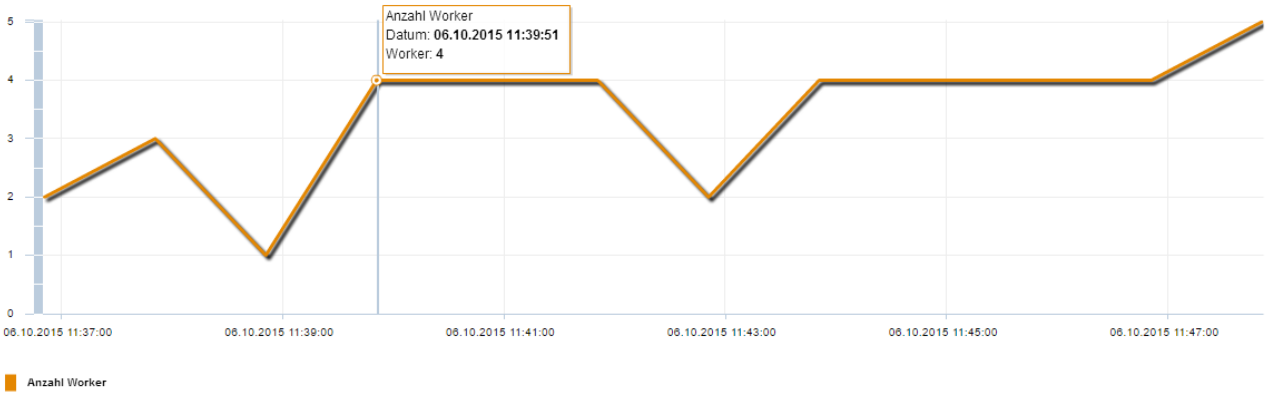
Die Skala der Verlaufskurve wird automatisch an die Maximalwerte der aktuell dargestellten Verlaufskurve angepasst.

Die in *Abb. A (1)* auswählbaren Datensätze sind:

- Anzahl Worker
- Worker Performance
- Prozesse
- Verfügbare Worker
- Prozess-Pool
- Datenbank

Anzahl Worker *[Workers count]*

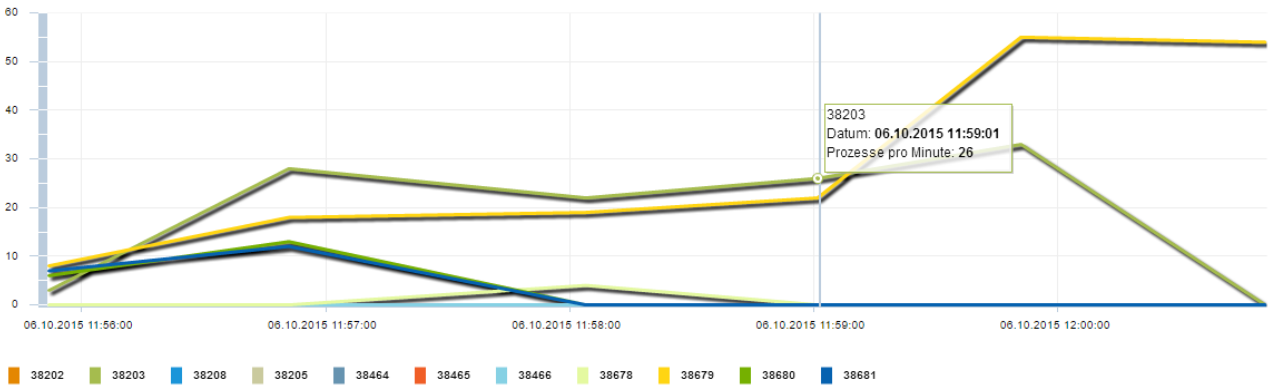
- **Gibt an, wie viele Worker zu einem bestimmten Zeitpunkt am Core registriert waren bzw. den entsprechenden Durchschnittswert für diesen Zeitpunkt.**



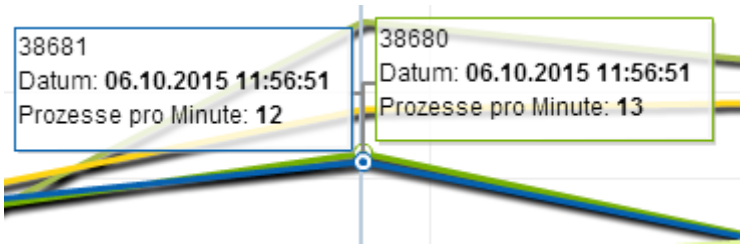
Diese Verlaufsgrafik gibt nicht an, wie viele Worker zu einem Zeitpunkt aktiv waren, sondern nur, wie viele Worker am Core registriert waren.

Worker Performance

- **Gibt an, wie viele Prozesse pro Minute jeder zu einem bestimmten Zeitpunkt aktive Worker verarbeitet hat bzw. den entsprechenden Durchschnittswert für diesen Zeitpunkt.**



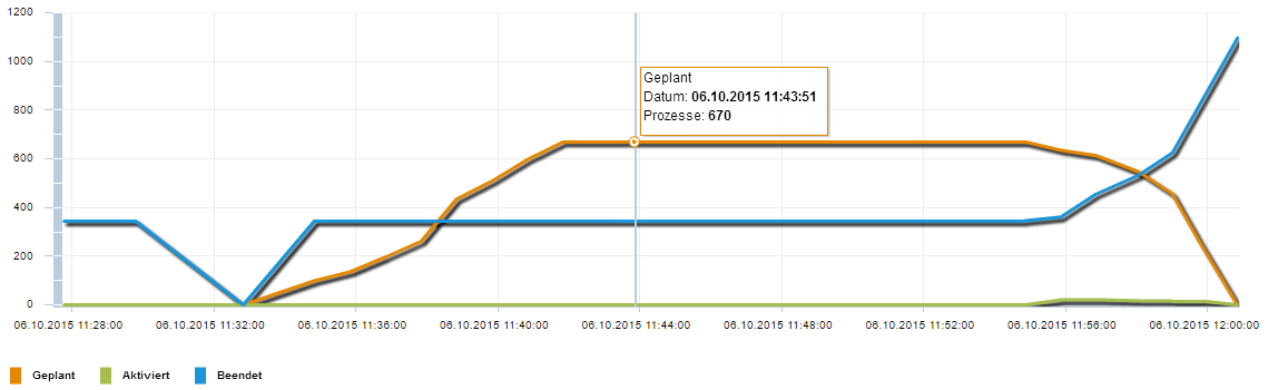
In dieser Verlaufsgrafik werden die Prozesse / Minute für alle Worker angezeigt, die in diesem Zeitraum registriert waren, also auch solche, die schon nicht mehr registriert sind bzw. nicht die ganze dargestellte Zeit über registriert waren. Solche Worker haben dann als Wert 0 Prozesse / Minute für die Zeiten, in denen Sie nicht mehr registriert waren. Als Legende werden Ihnen hier die ID's der Worker angezeigt. Wenn sich mehrere Workerlinien überlappen werden Ihnen zur vereinfachten Handhabung alle Werte dieser Worker angezeigt:



Dies gilt übrigens für alle Grafiken, bei denen mehr als eine Verlaufskurve dargestellt wird.

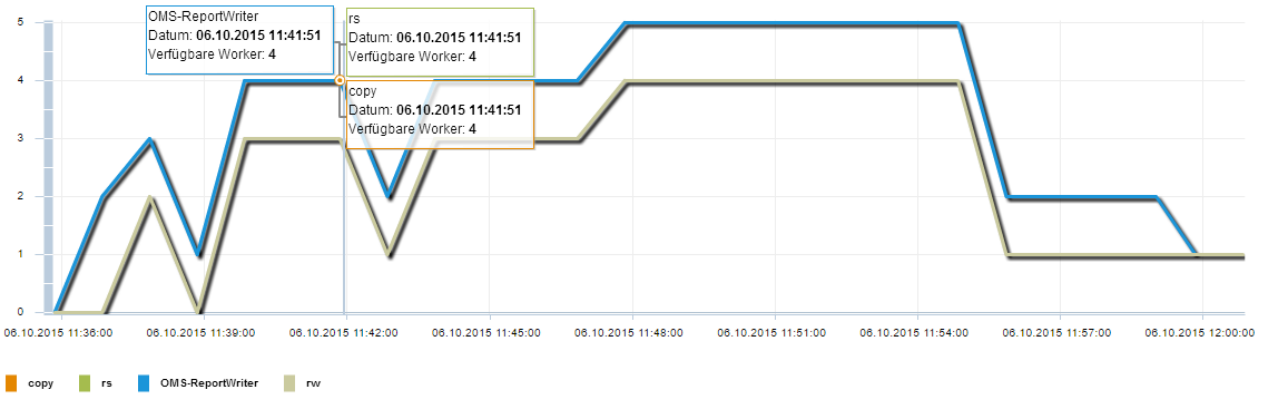
Prozesse *[Processes]*

- **Gibt die Gesamtzahl an geplanten *[scheduled]*, aktivierten *[activated]* und beendeten *[terminated]* Prozessen zu einem Z an bzw. den entsprechenden Durchschnittswert für diesen Zeitpunkt.**



Verfügbare Worker *[Available workers]*

- **Zeigt an, wie viele registrierte Worker zu einem Zeitpunkt für einen bestimmte Prozesstyp zur Verfügung stehen, diesen Prozesstyp also in ihrer List of Tasks definiert haben und solche Prozesse entgegennehmen können bzw. den entsprech Durchschnittswert für diesen Zeitpunkt.**

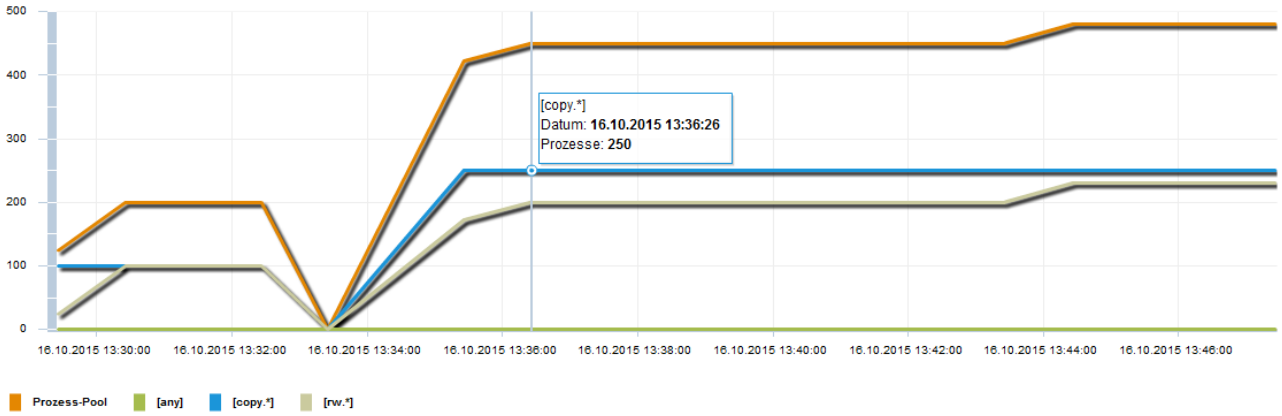


Dass ein Worker für einen bestimmten Prozesstyp zur Verfügung steht bedeutet nicht, dass der Worker auch solche Prozesse delegiert bekommt.

Wenn ein Worker für mehrere Prozesstypen konfiguriert ist, wird der Worker für jeden Prozesstyp als verfügbar angezeigt. Die Angabe zur Anzahl verfügbarer Worker ist also nicht exklusiv für einen Prozesstyp, sondern gibt an, wieviele aktuell registrierte Worker diesen Prozesstyp akzeptieren. Es stehen also im obigen Beispiel keine 12, sondern nur 4 Worker zur Verfügung.

Prozess-Pool *[Process Pool]*

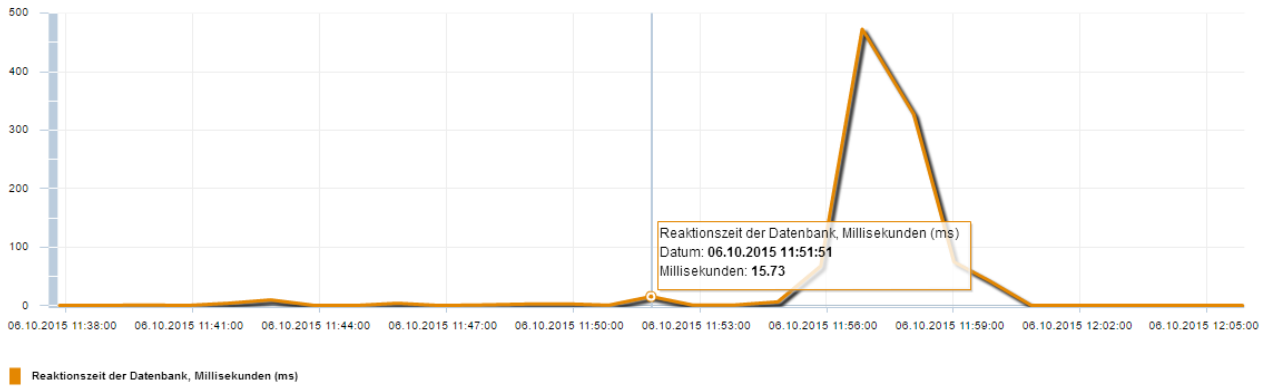
- **Zeigt an, wie die Prozessverteilung im Prozess-Pool ist, also wie viele Prozesse eines bestimmten Prozess-Typs im Prozess-Pool eingelagert sind und wie viele Prozesse der Prozess-Pool maximal fasst.**



Die Legende gibt den jeweiligen Prozess-Typ an, "Prozess-Pool" ist die Gesamtanzahl der im Prozess-Pool abgelegten Prozesse. Es können maximal Anzahl Parzellen (nodes in eoms.invoker.properties) * Größe pro Parzelle (size in eoms.invoker.properties) Prozesse im Prozess-Pool zwischengelagert werden. Die Größe pro Parzelle (size) limitiert die Anzahl maximal eingelagerter Prozesse pro Parzelle (oben: 250).

Datenbank *[Database]*

- **Zeigt die Reaktionszeit der Datenbank zu einem Zeitpunkt an (in ms) bzw. den entsprechenden Durchschnittswert für die Zeitpunkt.**



Falls regelmäßig hohe Reaktionszeiten auftreten sollten Sie überprüfen, ob genügend Ressourcen für die Worker zur Verfügung stehen.

Ueber (Eigenschaften)

Wenn Sie wissen möchten, welche Version des EOMS Sie benutzen oder einen bestimmten Konfigurationswert wissen müssen, finden Sie hier neben Versionhinweisen auch sämtliche Eigenschafts- und Konfigurationswerte des EOMS-Core. In der Version 1.3 des EOMS werden Ihnen neben den Systemvariablen auch die Clientvariablen (Eigenschaften der Oberfläche) angezeigt.

Wählen Sie in der Navigationsleitste 'Über' - 'EOMS-System-Info' *['About' - 'EOMS-System Info']*.

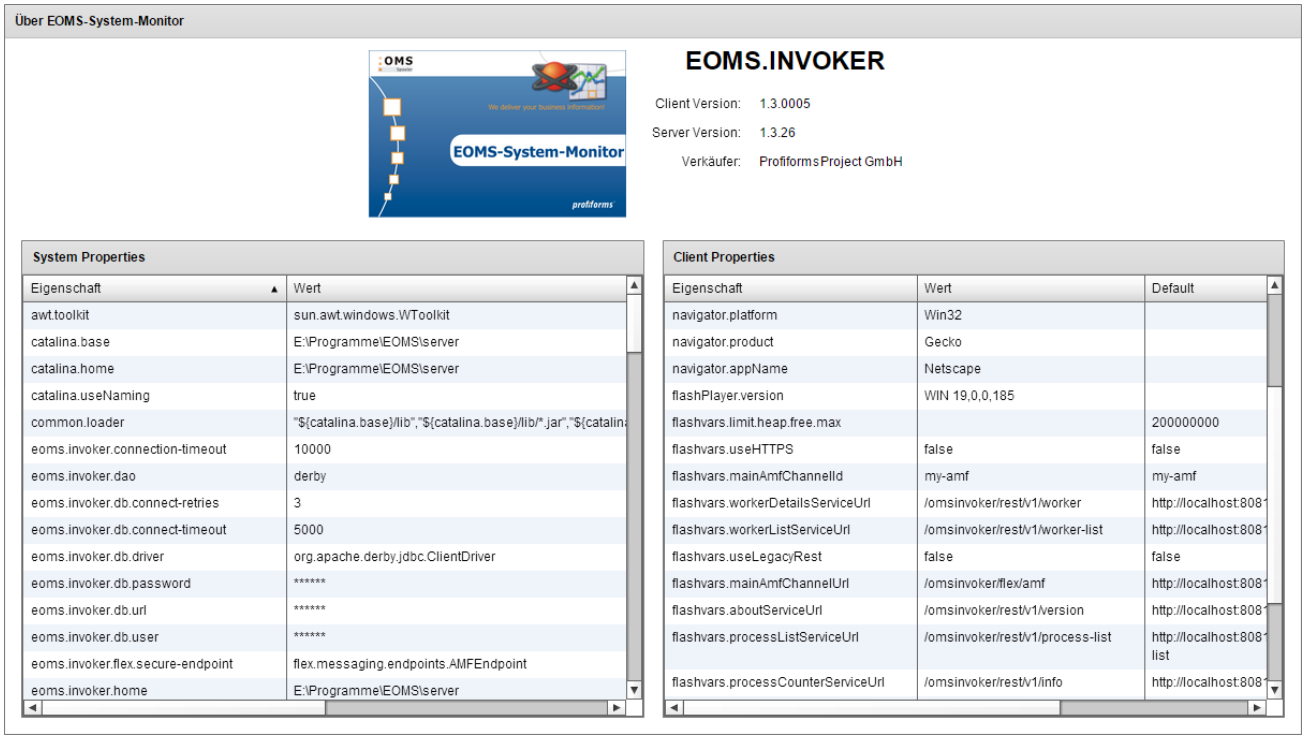


Abbildung A: EOMS-Core-Monitor mit geöffneter Eigenschaftsübersicht (Über).

Versionshinweise

- Client Version: Die Versionsnummer des EOMS-Core-Clients (der EOMS Zugriffsoberfläche für den Server) (zum Zeitpunkt der Erstellung dieser Dokumentation ist **1.3.0*** aktuell).
- Server Version: Die Versionsnummer des EOMS-Core-Servers (zum Zeitpunkt der Erstellung dieser Dokumentation ist **1.3.2*** akt

Eigenschaftswerte des Systems

Es folgt eine Erläuterung des Standardvariablen (Liste ist eventuell nicht vollständig) des EOMS-Core. Interne Java Variablen werden aufgeführt, aber nicht beschrieben. Der Großteil der hier aufgelisteten Eigenschaften ist auch in RCML über das properties-Feld ansprec hbar.

> In der Eigenschaftsübersicht werden die Werte des Hostrechners, auf dem das EOMS-Core läuft, angezeigt. Die Werte für die Hostrechner der Worker finden Sie im Worker-Monitor.

Eigenschaftsname	Beschreibung	Standardwert
awt.toolkit	Gibt das verwendete awt *(abstract window toolkit für die GUI)* an.	—
catalina.base	Das Installationsverzeichnis des Apache Tomcat Servers.	—
catalina.home	Das Homeverzeichnis (Speicherung von Nutzerdaten) des Apache Tomcat Servers.	—
catalina.useNaming	—	—
common.loader	Gibt die Verzeichnisse an, aus denen der Apache Tomcat Loader Klassen und .jar -files läd.	**${catalina.base}/lib** **${catalina.home}/lib**
eoms.invoker.connection-timeout	—	**10000**
eoms.invoker.dao	Das vom EOMS-Core verwendete Datenbanksystem.	**derby**
eoms.invoker.db.connect-retries	Gibt an, wie oft das EOMS nach einem fehlgeschlagenen Verbindungsaufbau zum Datenbankserver einen erneuten Versuch unternimmt, bevor abgebrochen wird. Kann in der ***eoms.invoker.properties*** festgelegt werden.	**3**
eoms.invoker.db.connect-timeout	Gibt an, wie viele Millisekunden gewartet wird, bis der neue Verbindungsaufbau initiiert wird. Kann in der ***eoms.invoker.properties*** festgelegt werden.	**5000**
eoms.invoker.db.driver	Gibt den Datenbanktreiber an.	**org.apache.derby.jdbc.ClientDriver** *(derby DB).*
eoms.invoker.db.password	Passwort der Datenbank. ⚠ wird dem Endnutzer nicht mehr angezeigt.	**eoms**
eoms.invoker.db.url	ConnectionString der Datenbank. ⚠ wird dem Endnutzer nicht mehr angezeigt.	**jdbc:derby://<ip>:8888/eoms;create=false** *wobei <ip>=IP des Hosts des Datenbankservers.*

eoms.invoker.db.user	Benutzername der Datenbank. ⚠ wird dem Endnutzer nicht mehr angezeigt.	**eoms**
eoms.invoker.home	Gibt das Installationsverzeichnis des EOMS-Core an.	—
eoms.invoker.max-total-connections	Maximale Anzahl an Verbindungen zum EOMS-Core.	**100**
eoms.invoker.messaging	Verwendeter Messagingtyp (direkt (stub) oder über ActiveMQ (jms)).	**stub**
eoms.invoker.messaging.apollo.uri	ConnectionString von Apollo (ActiveMQ).	**tcp://<ip>:63616** *wobei <ip>=IP des Hosts von ActiveMQ.*
eoms.invoker.process-pool.cluster.nodes	Anzahl Parzellen im Prozess-Pool	**20**
eoms.invoker.process-pool.size	Max. Anzahl Prozesse pro Parzelle im Prozess-Pool.	**100**
eoms.invoker.process-pool.type	Bestimmt, wie der Prozess-Pool Prozesse aufnimmt.	**default**
eoms.invoker.process.pessimistic-locking-exception.retries	—	—
eoms.invoker.process.pessimistic-locking-exception.sleep	—	—
eoms.invoker.process-terminate-callback-dispatcher.keep-alive	Minimale Anzahl an gleichzeitigen Dispatcher-Threads für Spooler-Callback. Siehe eoms.invoker.properties .	**150**
eoms.invoker.process-terminate-callback-dispatcher.max-threads	Maximale Anzahl an gleichzeitigen Dispatcher-Threads für das Spooler-Callback. Siehe eoms.invoker.properties.	**10**
eoms.invoker.process-terminate-callback-dispatcher.min-threads	Größe der Spooler-Callback-Warteschlange; Maximale Anzahl terminierter Jobs in der Warteschlange. Siehe eoms.invoker.properties .	**500**
eoms.invoker.process-terminate-callback-dispatcher.queue-capacity	Zeit in ms, wie lange ein Thread ohne ausgeführten Dispatch existieren darf, bevor er beendet wird. Siehe eoms.invoker.properties .	**3**
eoms.invoker.server-name	Der Servername (optionale Angabe in der eoms.invoker.properties).	—
eoms.invoker.process.terminated-process-collector.hours	Gibt an, wie viele Stunden terminierte Prozesse in der Datenbank gespeichert bleiben sollen, bevor sie entfernt werden.	**24**

file.encoding	Verwendete Zeichenkodierung.	Unter Windows: **Cp1252** *(ISO 8859-1 CP1252 West European)*
file.encoding.pkg	Package für die Zeichenkodierung.	**sun.io**
file.separator	Trennzeichen für Dateipfade.	****
java.awt.graphicsenv	⚠ wird dem Endnutzer nicht mehr angezeigt.	—
java.awt.printerjob	⚠ wird dem Endnutzer nicht mehr angezeigt.	—
java.class.path	Klassenpfad von Java. ⚠ wird dem Endnutzer nicht mehr angezeigt.	—
java.class.version	⚠ wird dem Endnutzer nicht mehr angezeigt.	—
java.endorsed.dirs	⚠ wird dem Endnutzer nicht mehr angezeigt.	—
java.ext.dirs	Externe Verzeichnisse für den Class Loader. ⚠ wird dem Endnutzer nicht mehr angezeigt.	—

java.home	Installationsverzeichnis des JDK/JRE.  wird dem Endnutzer nicht mehr angezeigt.	-
java.io.tmpdir	Temporärer Ordner.  wird dem Endnutzer nicht mehr angezeigt.	**${eoms.invoker.home}\temp**
java.library.path	Verzeichnisse für Java-Bibliotheken.  wird dem Endnutzer nicht mehr angezeigt.	**${eoms.invoker.home}/bin** **und weitere**
java.naming.factory.initial	 wird dem Endnutzer nicht mehr angezeigt.	—
java.naming.factory.url.pkgs	 wird dem Endnutzer nicht mehr angezeigt.	—
java.runtime.name	Name des JRE.  wird dem Endnutzer nicht mehr angezeigt.	—
java.runtime.version	Versionsnummer des JRE.  wird dem Endnutzer nicht mehr angezeigt.	—

java.specification.name	wird dem Endnutzer nicht mehr angezeigt.	—
java.specification.vendor	Anbieter der Javaspezifikation. wird dem Endnutzer nicht mehr angezeigt.	**Sun Microsystems inc.**
java.specification.version	Version der Javaspezifikation. wird dem Endnutzer nicht mehr angezeigt.	—
java.util.logging.config.file	Konfigurationsdatei für den log4j-Logger. wird dem Endnutzer nicht mehr angezeigt.	**${eoms.invoker.home}\conf\logging.properties**
java.util.logging.manager	wird dem Endnutzer nicht mehr angezeigt.	—
java.vendor	Anbieter der Java Distribution. wird dem Endnutzer nicht mehr angezeigt.	**Sun Microsystems inc.**

java.vendor.url	Homepage von java.vendor. wird dem Endnutzer nicht mehr angezeigt.	—
java.vendor.url.bug	Homepage des java.vendor, um einen Bug zu reporten. wird dem Endnutzer nicht mehr angezeigt.	—
java.version	Versionsnummer der Java Distribution. wird dem Endnutzer nicht mehr angezeigt.	—
java.vm.info	Mode der Java Virtual Machine. wird dem Endnutzer nicht mehr angezeigt.	**mixed mode**
java.vm.name	Name der Java Virtual Machine. wird dem Endnutzer nicht mehr angezeigt.	—
java.vm.specification.name	 wird dem Endnutzer nicht mehr angezeigt.	—

java.vm.specification.vendor	Distributor der Java Virtual Machine Spezifikation. ⚠ wird dem Endnutzer nicht mehr angezeigt.	**Sun Microsystems inc.**
java.vm.specification.version	Versionsnummer der Java Virtual Machine Spezifikation. ⚠ wird dem Endnutzer nicht mehr angezeigt.	**1**
java.vm.vendor	Distributor der Java Virtual Machine. ⚠ wird dem Endnutzer nicht mehr angezeigt.	**Sun Microsystems inc.**
java.vm.version	Versionsnummer der Java Virtual Machine. ⚠ wird dem Endnutzer nicht mehr angezeigt.	—
line.separator	Trennzeichen für Zeilenumbrüche.	**0**
oms.invoker.db.password	Passwort der OMS-Datenbank.	**eoms**
oms.invoker.db.url	Adresse der OMS-Datenbank.	**localhost**
oms.invoker.db.user	Benutzername der OMS-Datenbank.	**eoms**
os.arch	Gibt den Befehlssatz + die Architektur des EOMS-Core-Hosts an.	—
os.name	Gibt das Betriebssystem des EOMS-Core-Hosts an.	—
os.version	Gibt die Versionsnummer des Betriebssystems an.	—
package.access	—	—
package.definition	—	—
path.separator	Trennzeichen für Pfade bei Pfadangaben.	;
server.loader	—	—

shared.loader	—	—
sun.arch.data.model	Architektur des Hostrechners (64/32 bit).	—
sun.boot.class.path	—	—
sun.boot.library.path	\bin Verzeichnis des JDK.	**${java.home}\bin**
sun.cpu.endian	Byte-Speicherreihenfolge der CPU (little / big endian).	—
sun.cpu.isalist	wie ***os.arch***.	—
sun.desktop	wie ***os.name*** ohne Versionsnummer.	—
sun.io.unicode.encoding	Encoding für Unicode.	**UnicodeLittle**
sun.jnu.encoding	wie ***file.encoding***.	Unter Windows: **Cp1252** *(ISO 8859-1 CP1252 West European)*
sun.management.compiler	—	—
sun.os.patch.level	—	—
tomcat.util.buf.StringCache.byte.enabled	—	—
user.country	ISO 3166 alpha-2 Ländercode des Hostrechners.	—
user.dir	wie ***eoms.invoker.home***.	—
user.home	Wurzelverzeichnis des Hosts.	—
user.language	Verwendete Sprache auf dem Hostrechner.	—
user.name	Name des Benutzers, unter dem das EOMS-Core auf dem Hostrechner ausgeführt wird.	—
user.timezone	Zeitzone des Hostrechners	—
user.variant	—	—

Eigenschaftswerte des Clients

Neu in Version 1.3

Es werden werden Ihnen zusätzlich zu den EOMS-Core Eigenschaften auch die des EOMS-Clients (der Oberfläche) angezeigt.

Eigenschaftsname	Beschreibung	Standardwert
navigator.language	Browserspracheinstellung (nicht die ausgewählte Sprache im EOMS-Client, sondern die Standardsprache Ihres Browsers).	—
navigator.vendor	Hersteller Ihres Browsers.	—
navigator.appVersion	Versionsangaben zu Ihrem Browser (eventuell wichtig falls Probleme auftreten).	—
navigator.userAgent	Informationen zu Ihrem Browser (Header).	—
navigator.appCodeName	Name Ihres Browsers.	—
navigator.platform	Plattform (Betriebssystem), auf dem der Browser arbeitet und für den er geeignet ist.	—
navigator.product	Interner Produktname Ihres Browsers.	—
flashPlayer.version	Installierte Version des FlashPlayers (hier bestehen gewisse Voraussetzungen seite ns des EOMS).	—
flashvars.useHTTPS	Angabe, ob Ihr FlashPlayer HTTPS (sichere Verbindung) verwendet (true falls ja, false falls nein).	**false**
flashvars.workerDetailsServiceUrl	URL, über die die Serviceseite für Worker erreichbar ist (einzelne Worker, per REST). Siehe dazu auch den Abschnitt über REST.	**<server>/omsinvoker/rest/v1/worker**
flashvars.workerListServiceUrl	URL, über die die Serviceseite für Worker erreichbar ist (Workerliste, per REST). Siehe dazu auch den Abschnitt über REST.	**<server>/omsinvoker/rest/v1/worker-list**
flashvars.processDetailsServiceUrl	URL, über die Serviceseite für Prozesse erreichbar ist (einzelne Prozesse, per REST). Siehe dazu auch den Abschnitt über REST.	**<server>/omsinvoker/rest/v1/process**
flashvars.processListServiceUrl	URL, über die Serviceseite für Prozesse erreichbar ist (Prozessliste, per REST). Siehe dazu auch den Abschnitt über REST.	**<server>/omsinvoker/rest/v1/process-list**
flashvars.processSearchServiceUrl	URL, über die Prozesse gesucht werden können (per REST). Siehe dazu auch den Abschnitt über REST.	**<server>/omsinvoker/rest/v1/process-search**
flashvars.processCounterServiceUrl	URL, über die Infoseite für das EOMS erreichbar ist (per REST). Siehe dazu auch den Abschnitt über REST.	**<server>/omsinvoker/rest/v1/info**
flashvars.systemMonitorServiceUrl	URL, über die die Monitoringseite des EOMS erreichbar ist (per REST). Siehe dazu auch den Abschnitt über REST.	**<server>/omsinvoker/rest/v1/monitor**

flashvars.aboutServiceUrl	URL, über die Versionsinformation des EOMS abgerufen werden kann (per REST). Siehe dazu auch den Abschnitt über REST.	**<server>/omsinvoker/rest/v1/version**
flashvars.helpUrl	URL, über die die Onlinehilfe für das EOMS erreichbar ist.	—

Konfiguration und Struktur

Die Konfiguration des EOMS-Core ist in der Regel Aufgabe des Administrators, deshalb ist die Dokumentation zur Corekonfiguration in der Administrator-Dokumentation zu finden.

EOMS-Worker

Die EOMS-Worker empfangen die Jobs vom EOMS-Core, verarbeiten die Daten aus dem Auftrags-System und senden diese gegebenfalls an das Auftrags-System zurück oder an andere Endpunkte. Worker besitzen keine grafische Oberfläche wie das EOMS-Core. Im Auslieferungszustand des EOMS unterscheiden sich je nach Einsatzzweck 2 Typen von Workern:

- **OMS-Worker**
- **EOMS-Input-Worker**

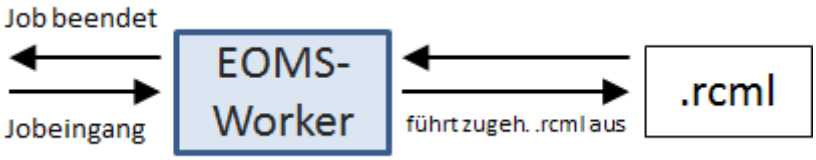
Genau genommen handelt es sich nicht um 2 verschiedene Workertypen, sondern um 2 Worker mit verschiedenen Vorkonfigurationen für verschiedene Zwecke. Worker können sehr tiefgreifend konfiguriert und spezialisiert werden. Die Konfiguration findet ausschließlich in verschiedenen Konfigurationsdateien auf Dateiebene statt. Die Prozesse, die ein Worker ausführt, werden mit Hilfe der Rich Client Markup Language (RCML) in der zugehörigen ***.rcml*** definiert. Vertiefend wird darauf im Abschnitt RCML Kompendium eingegangen.

Workertypen

EOMS-Worker dienen nicht nur einem statisch vorgegebenen Zweck. Stattdessen gibt es unterschiedliche Szenarien für den Einsatz von Workern. Dabei sind vor allem 2 grundsätzliche Workertypen zu unterscheiden, die sich ausschließlich in ihrer Konfiguration und RCML u nterscheiden. Sie sind nicht an diese Konfigurationen gebunden. Durch RCML Kompendium lassen sich Worker auch für abweichende Zwecke einrichten.

Arbeitsweise eines Workers

Egal welchen Workertyp Sie starten (start-oms-worker.cmd, start-eoms-input-worker.cmd), Sie starten dabei immer den gleichen Worker. Der Unterschied besteht nur darin, dass unterschiedliche RCML's geladen werden: Der OMS-Worker verwendet worker.rcml und der EOMS-Input-Worker eoms-input.rcml. Der Worker geht dabei aber immer gleich vor:

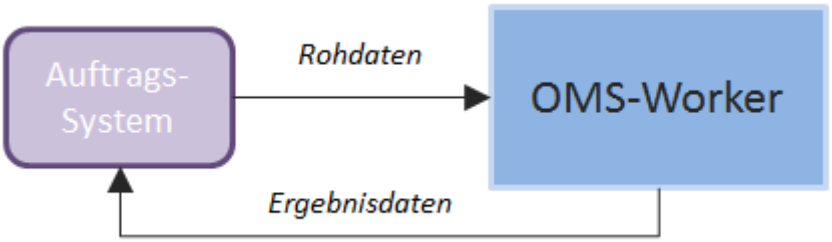


Nachdem der Jobauftrag im Worker eingetroffen ist, führt der Worker die zugehörige .rcml aus und meldet danach die Beendigung des Jobs zurück an das EOMS-Core. Die gesamte Tätigkeit eines Workers besteht also nur im Ausführen seiner .rcml. Ein Worker empfängt, verschickt oder verarbeitet von sich aus keine Daten, dazu müssen diese Prozesse in seiner .rcml definiert sein. Deshalb teilen Sie einem Worker nur über seine RCML-Spezifikation mit, was er zu tun hat. Welche RCML der Worker ausführt, hängt davon ab, welchen Workertyp Sie startet (wobei Sie die .rcml's natürlich trotzdem komplett umändern können). Sie können also auch Ihren eigenen Worker definieren, indem Sie die vorhandenen RCML's umändern oder komplett neue anlegen, die Sie dann beim Workerstart aufrufen.

Die 2 vorkonfigurierten Workertypen

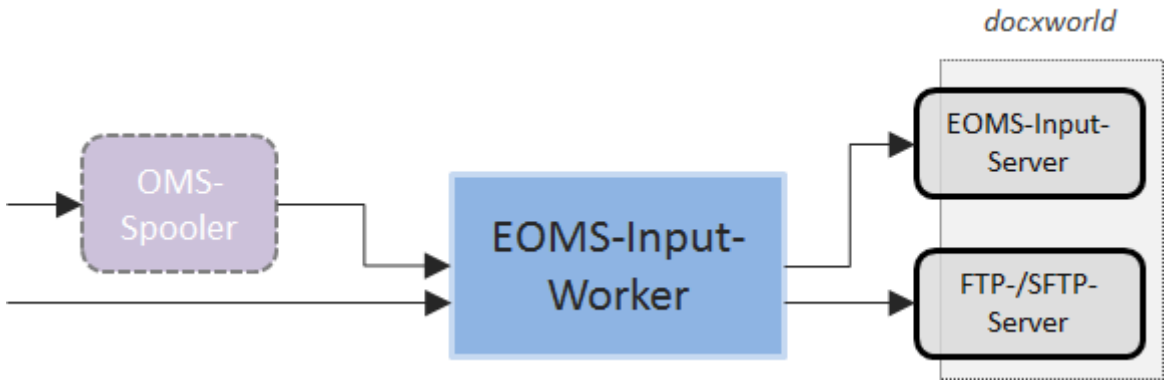
Mit ausgeliefert werden standardmäßig bereits 2 unterschiedliche RCML-Konfigurationen (AKI-Worker werden in Version 1.3 nicht mehr unterstützt):

OMS-Worker



OMS-Worker bilden den typischen Einsatzzweck für das EOMS. Dabei übernehmen die Worker die Verarbeitung der Spooler-Jobs (bzw. des Auftrags-Systems). Die Verarbeitungslogik des Workers ist dabei frei durch die Prozessdefinition per RCML festlegbar und nicht durch den Spooler (bzw. das Auftrags-System) begrenzt. OMS-Worker ermöglichen ebenfalls die Auslesung von Binär- und Produktions-Paketen aus dem Redaktions-System.

EOMS-Input-Worker



EOMS-Input-Worker stellen eine Verbindung zum EOMS-Input Server von docxworld her und ermöglichen so die Einlieferung von Daten per FTP/SFTP in docxworld. Die Rohdaten können dabei direkt aus einem SAP-System oder einem vorgeschalteten OMS-Spooler stammen. Es ist außerdem möglich, Statusabfragen über den eingelieferten Job zu tätigen und Ergebnisdaten zurück aus dem EOMS-Input-Server zu empfangen.

OMS-Worker

Der OMS-Worker ist einer der beiden vorkonfigurierten Workertypen. Er ist hauptsächlich dafür eingerichtet, Jobs aus dem Spooler zu empfangen und mit dem ReportWriter zu verarbeiten. Dabei ist der OMS-Worker auch standardmäßig dazu in der Lage, Daten aus einem Redaktions-System zu empfangen und mit diesen zu arbeiten.

Der OMS-Worker wird mit ***start-oms-worker.cmd*** gestartet. Die verwendete RCML ist ***config/worker.rcml***.

Der OMS-Worker ist bei Auslieferungszustand dazu in der Lage, folgende Prozesse auszuführen (die Sie beliebig verändern / ersetzen können):
1. Prozess "rs"

Der Prozess "rs" empfängt eine Spooler-XML (R-S Rohdaten) aus dem Spooler, die dann mit den Paketen aus einem Redaktions-System zusammengeführt wird. Nach Erhalt der Spooler-XML baut der Worker eine Verbindung zum Redaktions-System auf und läd die zum Job passenden Pakete (Binär- und Produktions-Paket). Die Pakete werden in einem Cache gespeichert, um die Ausführung für weitere Jobs zu beschleunigen und den Verbindungsaufwand zu reduzieren. Liegt das benötigte Paket bereits im Cache vor, müssen keine Daten extra vom Redaktions-System geladen werden. Danach führt der Prozess das Programm aus dem Binär-Paket aus (für Spooler-XML und Produktions-Paket). Das Ergebnis wird an die angegebene Output-Adresse gesendet (in der Regel zurück zum Spooler).

Die Implementierung des Prozesses in der oms.rcml wird hier beschrieben.
2. Prozess "copy"

Der Prozess "copy" führt einen simplen Kopiervorgang auf dem Workerrechner durch. Dazu wird eine Ressource aus dem Auftrags-System empfangen und an den angegeben Ort kopiert, bevor Sie wieder zurückgeschickt wird. Dieser Prozess dient auch als Testprozess zur Überprüfung der Funktionsfähigkeit des Workers.

Die Implementierung des Prozesses in der oms.rcml wird hier beschrieben.
3. Prozess "OMS-ReportWriter"

Der Prozess "OMS-ReportWriter" führt, wie der Name schon sagt, den ReportWriter auf einer empfangenen Ressource aus. Nach Erhalt der Ressource aus dem Auftrags-System (Spooler) wird der ReportWriter aufgerufen und ihm u.a. die Ressource zur Bearbeitung übergeben. Nach erfolgreicher Beendigung sendet der Prozess die Ergebnisdaten an die angegebene Output-Adresse (in der Regel zurück an das Auftrags-System).

Die Implementierung des Prozesses in der oms.rcml wird hier beschrieben.

EOMS-Input-Worker

Der EOMS-Worker ist einer der beiden vorkonfigurierten Workertypen. Er ist hauptsächlich dafür eingerichtet, Daten im EOMS-Input-Server (docxworld) einzuliefern.

Der OMS-Worker wird mit ***start-eoms-worker.cmd*** gestartet. Die verwendete RCML ist ***config/eoms-input.rcml***.

Der EOMS-Input-Worker ist bei Auslieferungszustand dazu in der Lage, folgende Prozesse auszuführen (die Sie beliebig umändern / ersetzen können):
1. Prozess "eoms-input"

Der Prozess "eoms-input" sendet Ressource aus dem Auftrags-System an die EOMS-Input-Schnittstelle und stellt sicher, dass die Daten erfolgreich übertragen und vom EOMS-Input-Server kein Fehler geworfen wurde. Nach Empfang der Ressource wird diese sofort weiter an das EMOS-Input-Server geschickt. Die Verbindungsdaten werden nicht in der *eoms-input.rcml*, sondern in der ***eoms.invoker.client.properties*** angegeben. Der Prozess wartet dann auf eine positive Rückmeldung vom EOMS-Input-Server. Danach werden die Ergebnisdaten (z.B. Statusreports) vom EOMS-Input-Server geladen und an die Output-Adresse (in der Regel das Auftrags-System) geschickt.

Die Implementierung des Prozesses in der eoms-input.rcml wird hier beschrieben.
2. Prozess "eoms-input-submitonly"

Der Prozess "eoms-input-submitonly" verläuft absolut analog zum Prozess "eoms-input", mit dem Unterschied, dass hier keine Überprüfung stattfindet, ob Fehler beim EOMS-Input-Server aufgetreten sind. Ebenso werden auch keine Ergebnisdaten empfangen und zurückgeschickt. Dementsprechend ist dieser Prozess schneller, aber auch fehleranfälliger. Sie sollten ihn deshalb nur bei zeitkritischen Systemen vorziehen.

Die Implementierung des Prozesses in der eoms-input.rcml wird hier beschrieben.

Konfiguration und Struktur

Die Konfiguration der Worker ist in der Regel Aufgabe des Administrators, deshalb ist die Dokumentation zur Workerkonfiguration in der Administrator-Dokumentation zu finden.

RCML

Worker werden mit Hilfe von RCML programmiert. Dies ist in der Regel Aufgabe des Administrators, deshalb ist die Dokumentation zu RCML in der Administrator-Dokumentation zu finden.

Fehlerbehandlung

Die Fehlerbehandlung ist in der Regel Aufgabe des Administrators, weshalb typische Fehler und deren vorgeschlagene Lösungen in der Administrator-Dokumentation zu finden sind.

Administrator-Dokumentation

Die Administrator-Dokumentation wendet sich an für mit Installation, Konfiguration und Betrieb betraute Personen.

Sie gliedert sich in folgende Unterkapitel:

System-Voraussetzungen

Um das Redaktions-System ordnungsgemäß betreiben zu können, muss Ihr System gewisse Mindestanforderungen erfüllen.

Je nachdem, ob auf dem System der EOMS-Server oder ein EOMS-Client läuft, werden unterschiedliche Systemanforderungen gestellt.

Systemvoraussetzungen EOMS-Core

Um das EOMS-Core ordnungsgemäß zu betreiben, sind folgende Voraussetzungen notwendig:

Technische Voraussetzungen:

- Unterstützte Betriebssysteme. Getestete und freigegebene Systeme sind:
  - Windows Server 2003 und 2008
  - Linux (SuseLinux SLES 10)

Logistische Voraussetzungen:

- JAVA JDK 1.8 muss auf dem System verfügbar sein und die Umgebungsvariable **`JAVA_HOME`** muss auf das JAVA JDK-Installationsverzeichnis zeigen.

  
  Das EOMS benötigt in der Version 1.3 Java 8.

- Eine der unter dem Abschnitt Datenbanken genannte SQL-Datenbank muß vom Kunden installiert/eingerichtet sein.


Die Nutzung anderer als der hier aufgeführten Betriebssysteme ist möglich, wenn diese Java in der entsprechenden Version unterstützen.

Bitte beachten Sie, dass keinerlei Support für nicht unterstützte Betriebssysteme erbracht wird.

Systemvoraussetzungen EOMS-Worker

Um den EOMS-Worker ordnungsgemäß zu betreiben, sind folgende Voraussetzungen notwendig:

Technische Voraussetzungen:

- Unterstützte Betriebssysteme. Getestete und freigegebene Systeme sind:
  - Windows Server 2003 und 2008
  - Linux (SuseLinux SLES 10)

Logistische Voraussetzungen:

- JAVA JDK 1.8 muss auf dem System verfügbar sein und die Umgebungsvariable **`JAVA_HOME`** muss auf das JAVA JDK-Installationsverzeichnis zeigen.

Das EOMS benötigt in der Version 1.3 Java 8.

Die Nutzung anderer als der hier aufgeführten Betriebssysteme ist möglich, wenn diese Java in der entsprechenden Version unterstützen.

Bitte beachten Sie, dass keinerlei Support für nicht unterstützte Betriebssysteme erbracht wird.

Systemvoraussetzungen EOMS-Client

Um mit dem EOMS-Client arbeiten zu können, sind folgende Voraussetzungen notwendig:

Technische Voraussetzungen:

- Arbeitsplatz-PC mit aktueller Hardware-Ausstattung (mindestens 512 MB freiem Hauptspeicher).
- Bildschirm-Auflösung von mindestens 1200 x 840 Bildpunkten.
- WEB-Browser (getestete Versionen finden Sie im unteren Bereich dieses Artikels).
- Adobe Flash Player als Plugin im WEB-Browser (neuste von Adobe zur Verfügung gestellte Version).
- im Web-Browser müssen Cookies erlaubt sein.
- in einem WEB-Browser-Typ (z.B. Firefox, Internet Explorer) kann der R-S Client zu einer R-S Server-URL **NICHT MEHRFACH** ge werden (dies gilt auch für Registerkarten in WEB-Browsern).

Logistische Voraussetzungen:

- Sie müssen HTTP/HTTPS-Zugriff auf den EOMS-Server haben (kontaktieren Sie bitte Ihren System-Administrator).
- Sie müssen über ein gültigen Nutzer mit einem entsprechenden Passwort verfügen (kontaktieren Sie bitte Ihren System-Administ
- Sie müssen über die Berechtigung für den Zugriff auf mindestens einen Arbeitsbereich verfügen (kontaktieren Sie bitte Ihren System-Administrator).

Derzeit getestete/freigegebene Web-Browser (es wurden immer nur die Haupt-Versionen getestet):

- Internet-Explorer 8 und 9
- Mozilla Firefox 4.0 bis 9.0

Die Nutzung anderer als der hier aufgeführten Web-Browser ist möglich, wenn diese den Adobe Flash Player in der entsprechenden Version unterstützen.

Bitte beachten Sie, dass keinerlei Support für nicht unterstützte Web-Browser erbracht wird.

Installation

Für die Installation des EOMS benutzen Sie bitte die Produkt-CD, welche Sie mit dem Erwerb ihrer Lizenz erhalten. Sie werden mit dieser Produkt-CD weitestgehend durch die einzelnen Installations-Schritte geführt und können danach die durchgeführte Installation auf ihre Bedürfnisse anpassen.

Auch wenn die Installation über die Produkt-CD die favorisierte Installationsmethode ist, beschreiben wir hier zusätzlich noch die manuelle Installation. Diese erfolgt in folgenden Schritten:

1. Installation des JDK
2. Installation des EOMS-Core
3. Installation der EOMS-Worker

Beachten Sie auch die Hinweise zur Systemsicherheit.

JAVA

Alle Komponenten des EOMS benötigen zu ihrer Ausführung ein JAVA SE Development Kit (JDK) der Firma ORACLE. Die vom EOMS unterstützten Versionen des JAVA JDK finden Sie in den System-Voraussetzungen für EOMS aufgelistet.

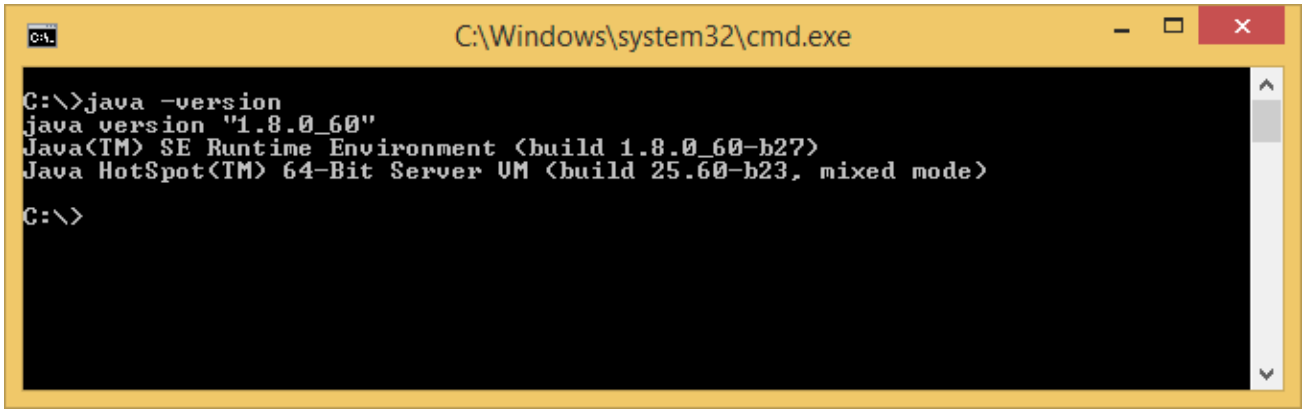
> **Wichtiger Hinweis**
>
> Die Installation/Konfiguration eines JAVA SE Development Kit (JDK) ist eine Installations-Voraussetzung, welche durch den Kunden/Betreiber des EOMS erbracht werden muß. Sprechen Sie hier mit Ihrem Systemadministrator.
>
> Nachfolgende Informationen sollen Sie dabei unterstützen zu prüfen ob JAVA auf einem Server-System installiert ist und in welcher Version JAVA vorliegt.

Prüfung der installierten JAVA-Version

Da JAVA heute eine sehr weit verbreitete Runtime-Umgebung für Applikationen ist kann davon ausgegangen werden das Sie JAVA schon vorinstalliert auf den unterschiedlichen Server-Betriebsystemen existiert. Sie können dies prüfen indem Sie eine Betriebsystem-Shell öffnen (Windows = DOS-Eingabeaufforderung/Command-Shell und UNIX = BASH etc.) und folgenden Befehl eingeben:

```
> java -version
```

Ist JAVA auf dem System installiert, so erhalten Sie für Windows z.B. folgende Ausgabe:



Die Ausschrift bedeutet, das ein JAVA SE Develoment Kit (JDK) in der Version 1.8.0_60 als 64-Bit Version installiert ist und damit die Voraussetzung für die Installation von EOMS auf diesem Server gegeben sind. Wenn Ihnen hier eine Versionsnummer kleiner 1.8.x_xx angezeigt, also z.B. 1.7.0_xx oder kleiner, müssen Sie zuvor Java auf Version 8 aktualisieren, bevor Sie mit der Installation des EOMS fortfahren.

Bezugsquellen für JAVA

Sollte JAVA auf dem Server auf dem Sie EOMS installieren wollen nicht installiert sein, so können eine der vom EOMS unterstützte, kostenfreie Version des JAVA JDK über folgende Quellen beziehen:

Download von der ORACLE Web-Seite

ORACLE Web-Seite zum JAVA-Download

Kopieren von der EOMS Produkt-CD/DVD

Auf der Produkt CD/DVD des EOMS werden unterstützte JAVA JDK für ausgewählte Betriebssystem zur Verfügung gestellt. Sie finden diese Versionen im Ordner *'INSTALL'* und dort im Unterverzeichnis *'JAVA'*.

Wichtiger Hinweis

Für manche Betriebsysteme und spezifische Hardw-Infrastrukturen bieten die jeweiligen Hersteller modifizierte und auf die Infrastruktur angepaßte Versionen eines JAVA SE Development Kit (JDK) an, bitte informieren Sie sich darüber bei dem Hersteller des Servers und/oder Betriebssystems.

Umgebungsvariable JAVA_HOME

Für einzelne Komponenten des EOMS wird die Umgebungsvariablen **JAVA_HOME** benötigt.

Die Variable **JAVA_HOME** zeigt auf ein Verzeichnis in welchem das JDK (Java Development Kit) in der benötigten Version abgelegt ist. Dabei wird nicht das BIN-Verzeichnis in der JAVA-Installation angegeben, sondern das direkt übergeordnete Verzeichnis.

Es gibt mehrere Möglichkeiten, Umgebungsvariablen im Betriebssystem oder für den Nutzer zu setzen, welcher die EOMS Software-Komponenten startet.

Sprechen Sie hier mit Ihrem Systemadministrator.

Durch EOMS Software-Komponenten (z.B. Applikations-Server) wird auch geprüft, ob die Umgebungsvariable **JRE_HOME** exisi tiert.

Ist diese vorhanden (auf manchen LINUX/UNIX-Servern), so wird diese vorrangig **VOR** der Umgebungsvariable **JAVA_HOME** g enutzt.

Existiert die Umgebungsvariable **JRE_HOME**, so sollte diese Umgebungsvariable gelöscht oder zumindest geprüft werden,

ob diese Umgebungsvariable auf eine kompatible **JAVA**-Installation zeigt.

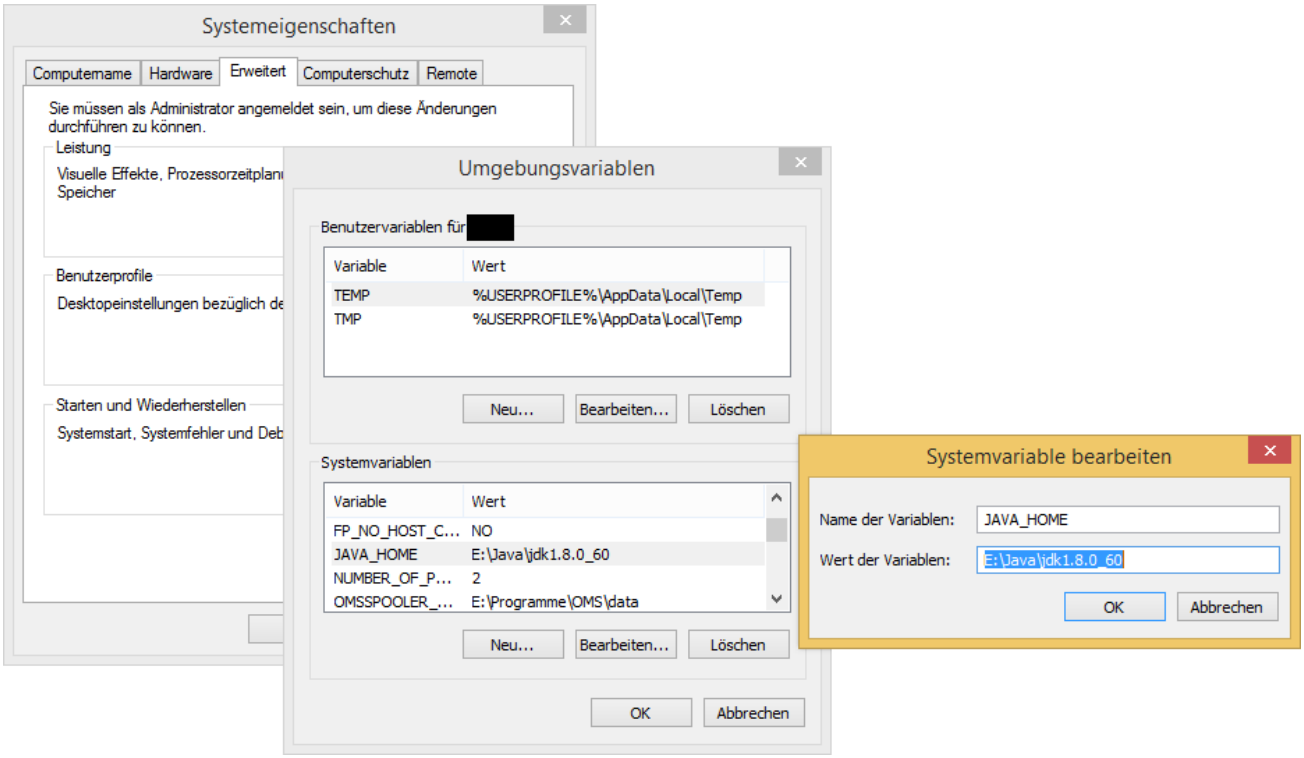
Tipp: Lässt sich aus organisatorischen Gründen die Umgebungsvariable **JRE_HOME** nicht löschen und wird damit auch

keine gültige JAVA-Version referenziert, so können die Start-Scripts der Software-Komponenten den Bedürfnissen angepasst werden.

Wichtiger Hinweis: Verwenden Sie im Verzeichnispfad für die JAVA-Installation keinerlei Leerzeichen!

Beispiel für das Setzen von Windows-Umgebungsvariablen:

Um in einem Windows-System die Umgebungsvariable JAVA_HOME zu setzen, starten Sie die Systemsterung. Dort finden Sie unter der Rubrik SYSTEM unter den 'Erweiterten Systemeinstellungen' die Umgebungsvariablen.



Installation - EOMS-Core

Die Installation des EOMS-Core geschieht in mehreren Schritten:

Installation des Messaging-Servers *[optional]*

Installation des Applikations-Servers

Einrichtung der Datenbank

Messaging-Server

Die Rücklieferung von Status-Informationen vom EOMS-Core an die entsprechenden Auftrags-Systeme kann über einen Messaging-Server erfolgen. Im Folgenden wird die Installation/Konfiguration **Messaging-Servers von Apache - ActiveMQ** beschrieben.

Für die meisten Anwendungsfälle wird empfohlen, direkte http-Kommunikation ("stub") anstatt der hier beschriebenen indirekten Kommunikation zu verwenden. Wenn Sie einen Messaging-Server verwenden, müssen Sie dies in der eoms.invoker .properties des EOMS-Core entsprechend konfigurieren.

Bezugsquellen für ActiveMQ

- Download von der Apache Web-Seite
  - Apache Web-Seite zum ActiveMQ-Download
- Kopieren von der EOMS Produkt-CD/DVD
  - Auf der Produkt CD/DVD des EOMS werden unterstützte ActiveMQ-Versionen für ausgewählte Betriebssystem zur Verfü Sie finden diese Versionen im Ordner *'INSTALL'* und dort im Unterverzeichnis *'Messaging-Server'*.

Installation für ActiveMQ

Die Installation des Messaging-Servers gestaltet sich sehr einfach durch das Kopieren der Installationsdateien in ein Verzeichnis auf dem Server auf welchem das EOMS-Core installiert werden soll. Nach dem Download von ActiveMQ aus dem Internet oder nach dem Kopieren von der Installations-CD/DVD des EOMS ist die Installations-Datei typischerweise komprimiert (ZIP/tar.gz).

Entpacken Sie die Installationsdatei.

ActiveMQ als Betriebssystem-Service integrieren

Je nach Betriebssystem finden Sie im Installationsverzeichnis des Messaging-Servers mehrere Unterverzeichnisse, u.A. das Verzeichnis *'bin'* und unterhalb dieses Verzeichnisses je nach Betriebssystem die 32- oder 64-Bit-Version des Messaging-Servers.

- Windows
- UNIX

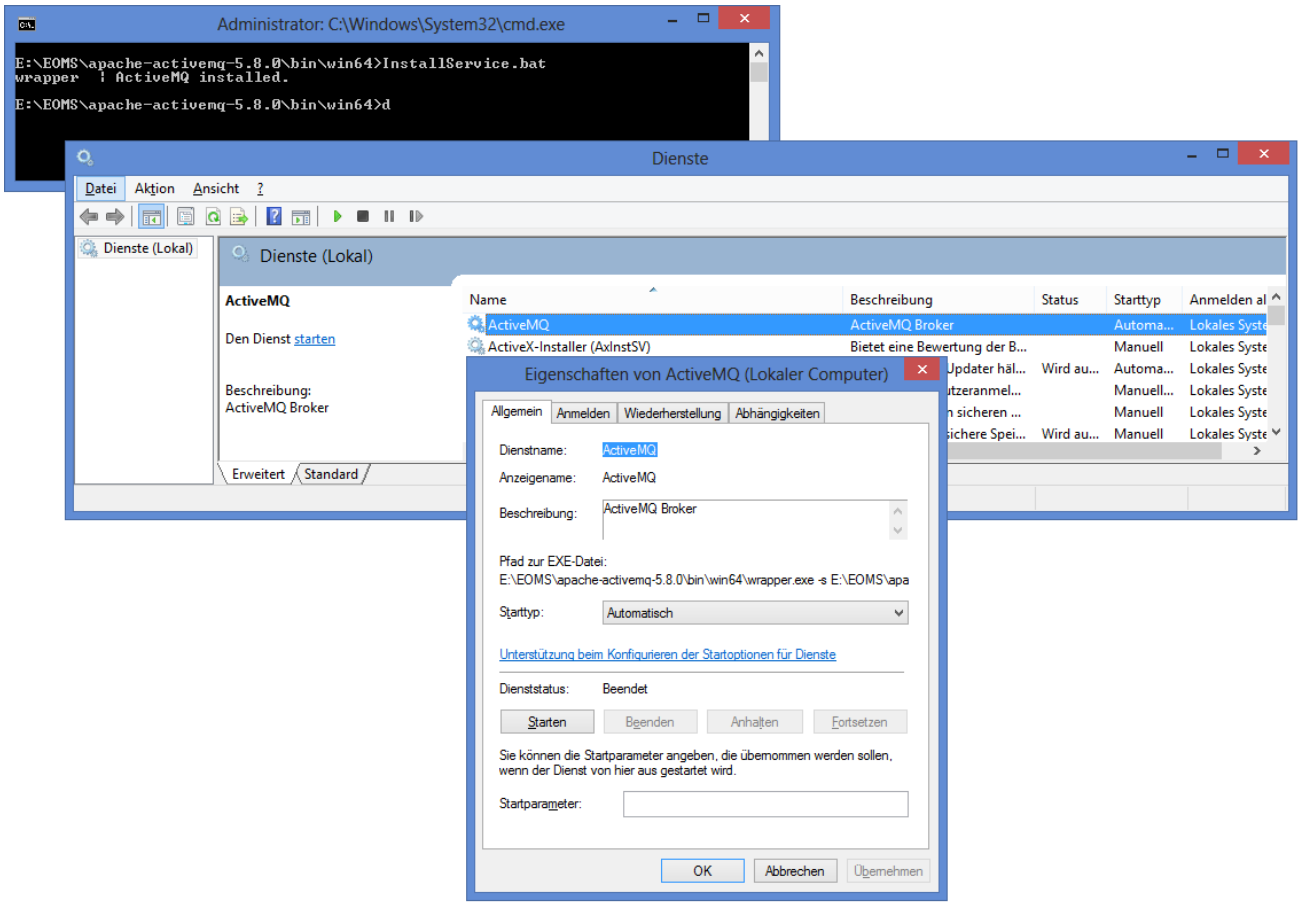
Windows

Öffnen Sie eine Betriebsystem-Shell (DOS-Eingabeaufforderung) und wechseln Sie in das 32-/oder 64-Bit Verzeichnis des Messagin-Servers (je nachdem ob Sie über ein 32- oder 64-Bit Betriebssystem verfügen).

Führen Sie nun folgenden Befehl aus um ActiveMQ als Windows-Service zu installieren:

```
> installService.bat
```

Konnte ActiveMQ erfolgreich als Windows-Service installiert werden sehen Sie in der Dienste-Verwaltung den Service als Starttyp 'automatisch' installiert. Dies bedeutet das ActiveMQ bei Start des Windows-Servers automatisch gestartet und beim Beenden des Windows-Servers auch automatisch beendet wird.



UNIX

Eine detaillierte Beschreibung der Installation und De-Installation des Messaging-Servers Apache ActiveMQ finden Sie hier: ActiveMQ-Dokumentation 'Getting Started'

Änderung der IP-Adresse für die Dienste des ActiveMQ

Nach dem Start des Messaging-Servers ActiveMQ stellt dieser seine Dienst über den Port 61616 auf der IP-Adresse '0.0.0.0'/localhost zur Verfügung. Da ActiveMQ jedoch die Schnittstelle für entfernt arbeitende Auftrags-Systeme ist um den Status von Aufträgen im EOMS-Core abzufragen, muss ActiveMQ im Netzwerk unter einer IP-Adresse verfügbar sein.

Lassen Sie sich von Ihrem Systemadministrator die IP-Adresse des Servers nennen, auf welchem Sie ActiveMQ installiert haben und unter welcher dieser Service im Netzwerk verfügbar gemacht werden soll (Dienste: ipconfig/ifconfig zur Analyse der IP-Adresse des Servers).

Um ActiveMQ an diese IP-Adresse zu binden, öffnen Sie die Konfigurationsdatei *'[INSTALL_DIR]\conf\activemq.xml'* in einem Editor. Suchen Sie dort nach folgendem Eintrag:

```
...
<transportConnectors>
<!-- DOS protection, limit concurrent connections to 1000 and frame
size to 100MB -->
<transportConnector name="openwire"
uri="tcp://0.0.0.0:61616?maximumConnections=1000&amp;wireformat.maxFram
eSize=104857600"/>
<transportConnector name="amqp"
uri="amqp://0.0.0.0:5672?maximumConnections=1000&amp;wireformat.maxFram
eSize=104857600"/>
</transportConnectors>
...
```

Ersetzen Sie die IP-Adresse '0.0.0.0' durch die IP-Adresse oder den HOST-Namen des Servers auf welchem ActiveMQ installiert ist. Beispiel:

```
...
<transportConnectors>
<!-- DOS protection, limit concurrent connections to 1000 and frame
size to 100MB -->
<transportConnector name="openwire"
uri="tcp://192.168.118.49:61616?maximumConnections=1000&amp;wireformat.
maxFrameSize=104857600"/>
<transportConnector name="amqp"
uri="amqp://192.168.118.49:5672?maximumConnections=1000&amp;wireformat.
maxFrameSize=104857600"/>
</transportConnectors>
...
```

Starten Sie nun den ActiveMQ Dienst neu.

Prüfen ob der Messaging-Service ActiveMQ erfolgreich gestartet worden ist

---

Haben Sie ActiveMQ erfolgreich installiert und als System-Service gestartet, so sollte der TCP/IP-Port 61616 verfügbar sein.

Dies können Sie unter Windows oder UNIX durch das Dienstprogramm 'netstat' (Wikipedia NETSTAT) prüfen.

```
> netstat -an
...
TCP 192.168.118.49:61616 0.0.0.0:0 ABHOEREN
...
```

Applikations-Server

Im Folgenden wird die Installation/Konfiguration **Applikations-Server Apache Tomcat** beschrieben.

Das EOMS 1.3 benötigt Tomcat in der Version 8 oder höher.

Bezugsquellen für Tomcat

- Download von der Apache WEB-Seite
  - Apache Tomcat 8.X Download-Seite
- Kopieren von der EOMS Produkt-CD/DVD
  - Auf der Produkt CD/DVD des EOMS werden unterstützte Apache Tomcat-Versionen für ausgewählte Betriebssystem zur gestellt. Sie finden diese Versionen im Ordner *'INSTALL'* und dort im Unterverzeichnis *'Applikations-Server'* für ihre Betriebssystem-Plattform.

Umgebungsvariablen Applikations-Server Apache Tomcat

Zum erfolgreichen Start des Applikations-Servers Apache Tomcat werden die Umgebungsvariablen **`JAVA_HOME`** und **`CATALINA_HOME`** b enötigt.

Die Variable **`CATALINA_HOME`** zeigt auf das Installationsverzeichnis des Applikations-Servers Apache Tomcat.

Ein Beispiel für Windows: *'D:\EOMS\apache-tomcat-8'*. Die Variable **`JAVA_HOME`** zeigt auf auf ein JDK (Java Development Kit) in der benötigten Version.

(Weitere Hinweise finden Sie hier: Umgebungsvariable JAVA_HOME).

Es gibt mehrere Möglichkeiten, Umgebungsvariablen im Betriebssystem oder für den Nutzer zu setzen, welcher den Applikations-Server startet.

Sprechen Sie hier mit Ihrem Systemadministrator.

Durch den Applikations-Server Apache Tomcat wird auch geprüft, ob die Umgebungsvariable **JRE_HOME** exisitiert.

Ist diese vorhanden (auf manchen LINUX/UNIX-Servern), so wird diese vorrangig **VOR** der Umgebungsvariable **JAVA_HOME** g enutzt.

Existiert die Umgebungsvariable **JRE_HOME**, so sollte diese Umgebungsvariable gelöscht oder zumindest geprüft werden,

ob diese Umgebungsvariable auf eine **JAVA**-Installation zeigt, welche kompatibel mit der Version des Applikations-Servers Apache Tomcat ist.

Tipp: Lässt sich aus organisatorischen Gründen die Umgebungsvariable **JRE_HOME** nicht löschen und wird damit auch

keine gültige JAVA-Version referenziert, so kann das Start-Script (catalina.bat|catalina.sh) den Bedürfnissen angepasst werden.

Installation für Apache Tomcat

Die Installation des Applikations-Servers gestaltet sich sehr einfach durch das Kopieren der Installationsdateien in ein Verzeichnis auf dem Server auf welchem das EOMS-Core installiert werden soll. Nach dem Download von Tomcat aus dem Internet oder nach dem kopieren von der Installations-CD/DVD des EOMS ist die Installations-Datei für UNIX-Betriebssysteme typischerweise komprimiert (tar.gz), entpacken Sie die Installationsdatei.

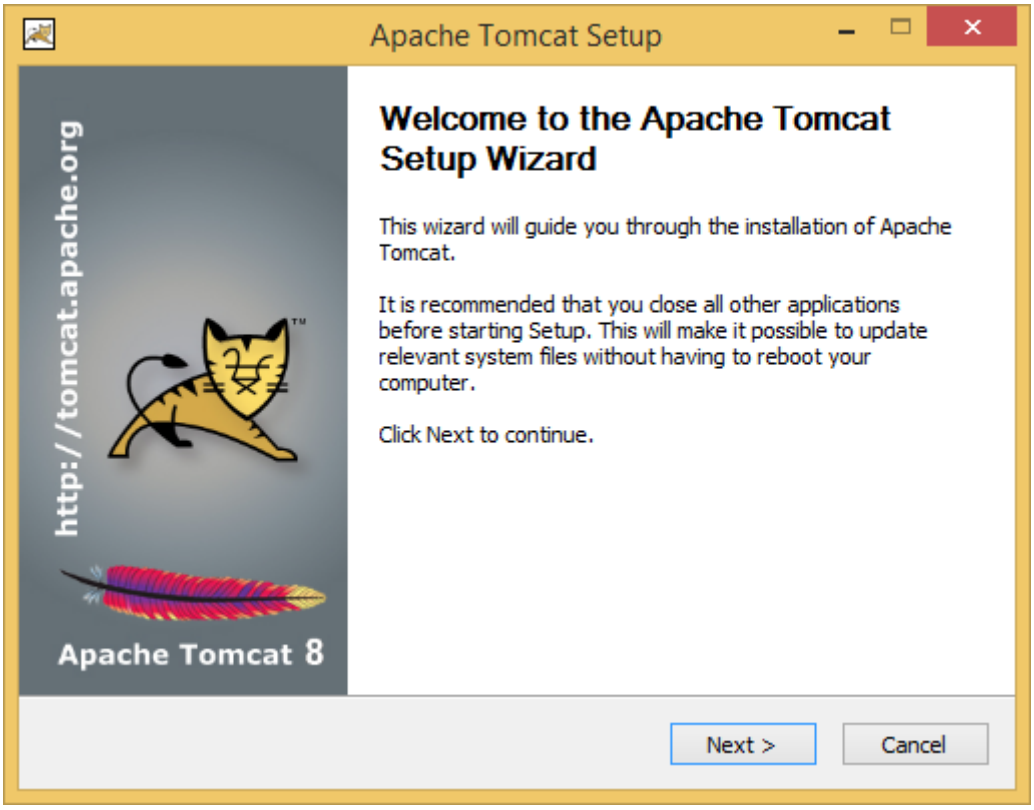
- Windows
- UNIX

Windows

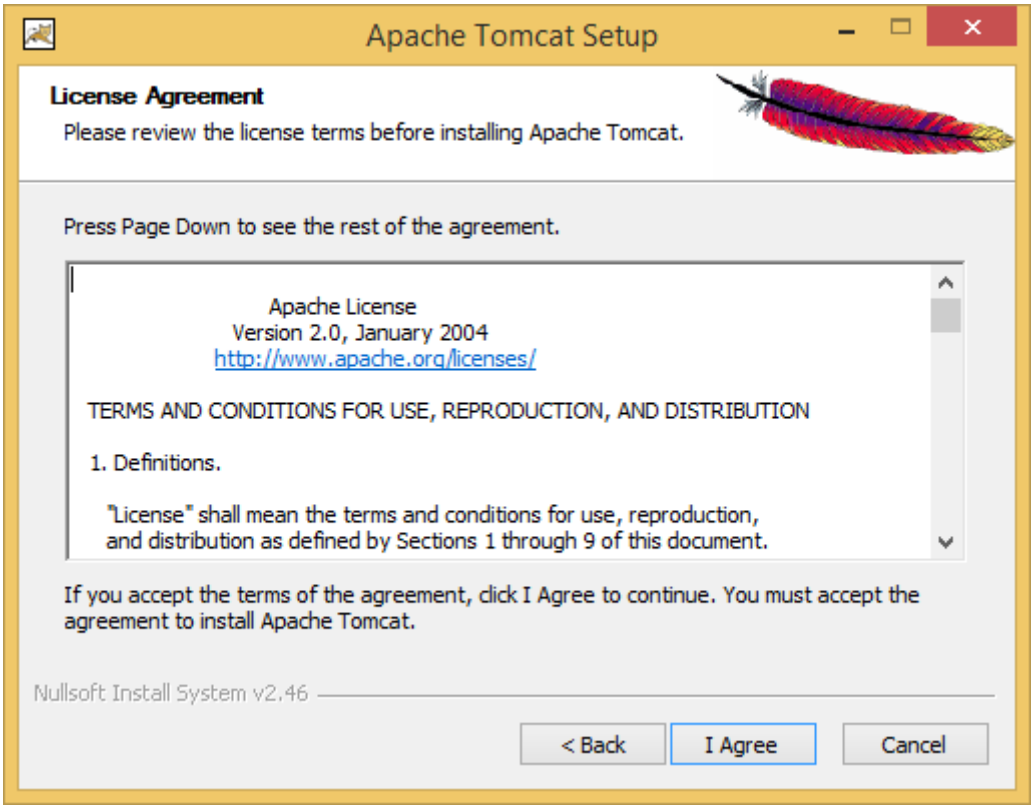
Unter Windows steht für die Installation des Apache Tomcat ein Installations-Routine zur Verfügung, welche den Applikations-Server als 32-/64-Bit Applikation im Windows installiert und als Systemdienst integriert.

Nachfolgend finden Sie eine kurze Beschreibung des Installationsvorganges:

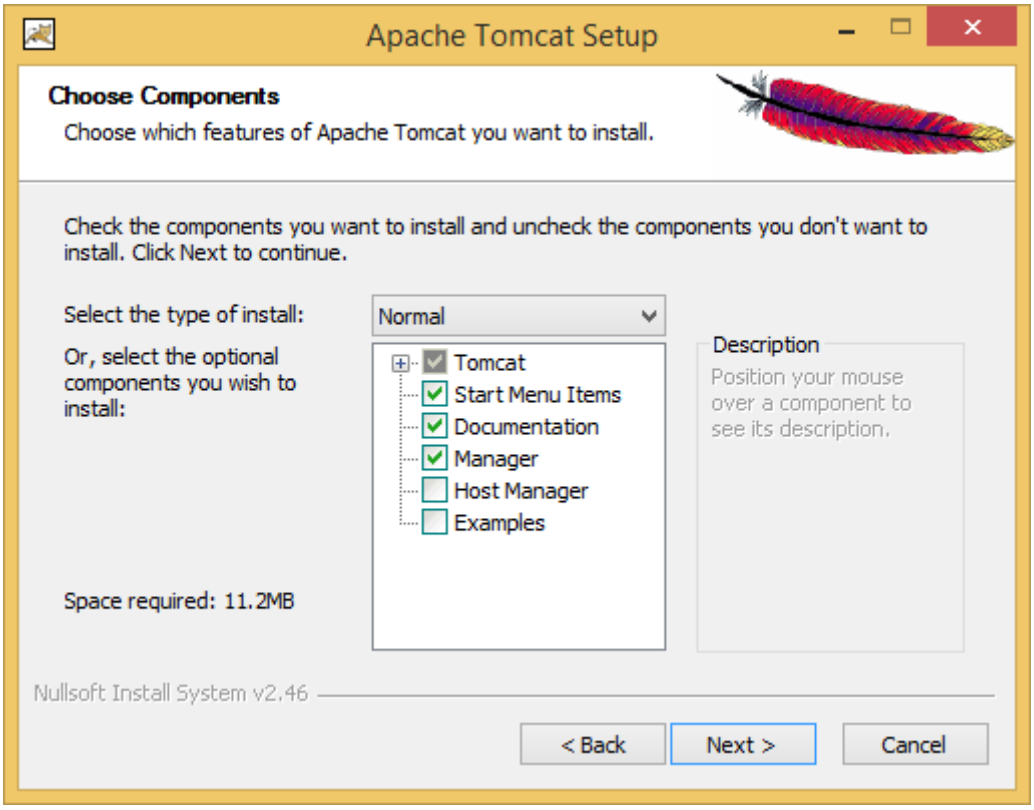
1. Klicken Sie auf die ausführbare Installationsdatei 'apache-tomcat-X.X.XX.exe'.



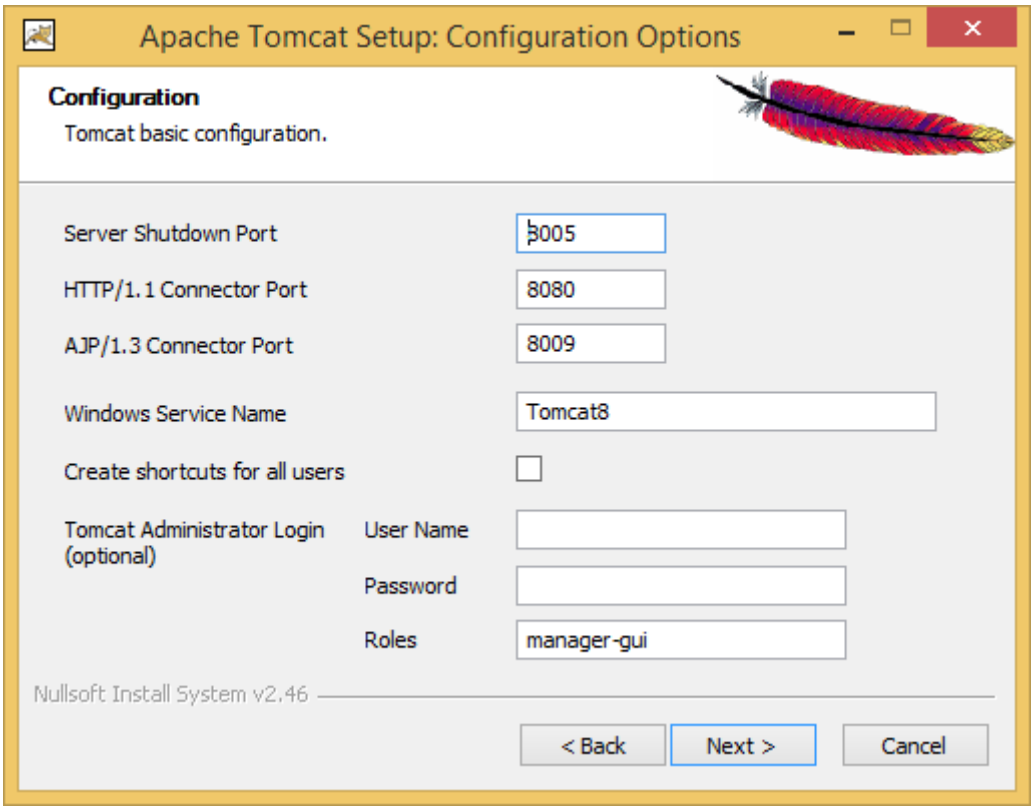
2. Akzeptieren Sie nach dem Durchlesen die Lizenzbedingungen .



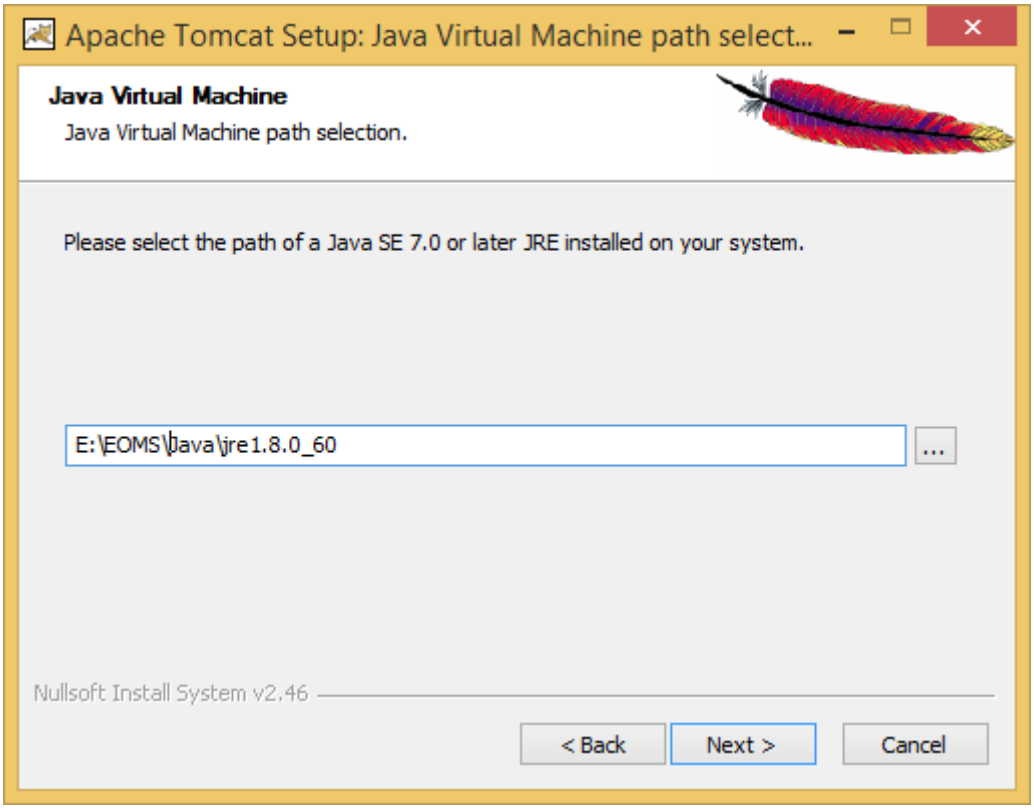
3. Wählen Sie die Installations-Optionen. Die Installation der Option 'Manager' ist aus Sicherheitsgründen nicht zu empfehlen.



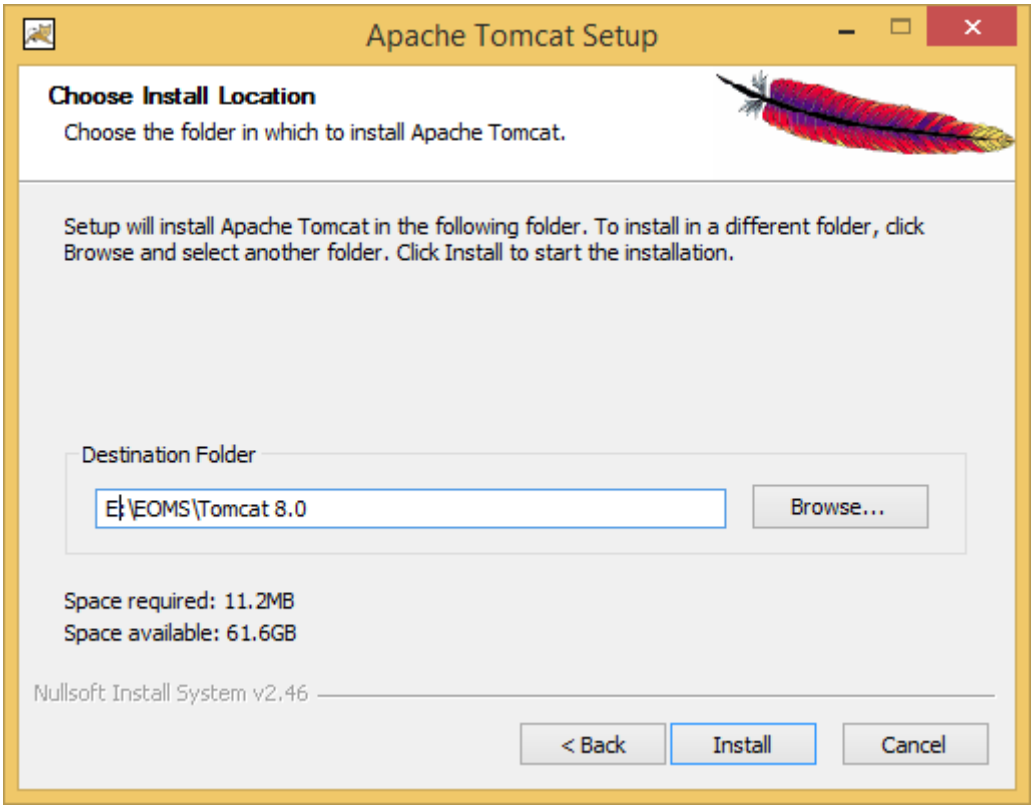
4. In den meisten Fällen können Sie die Standardkonfiguration verwenden, es sei denn, eine andere Anwendung blockiert den Port 8080 oder Sie haben andere Gründe für eine manuelle Konfiguration.
   Ansonsten wird stark davon abgeraten, die Konfiguration zu ändern.



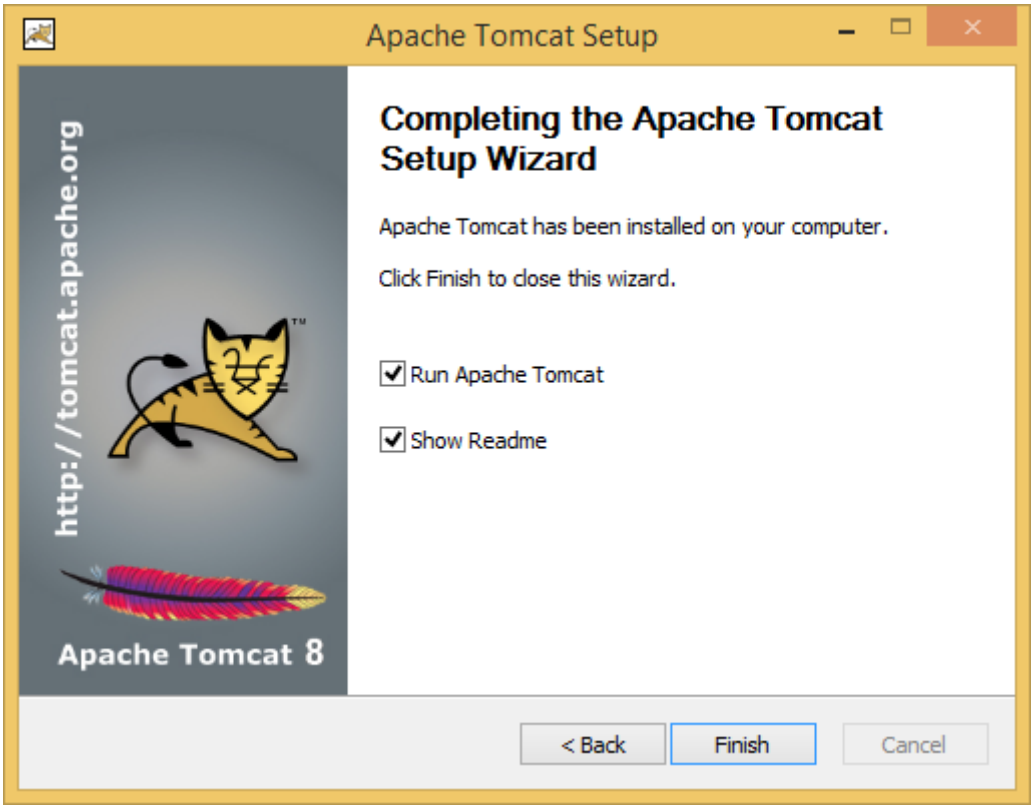
5. Pfad zum JAVA SE JDK (hier den Pfad von JAVA_HOME angeben).



6. Pfad angeben in welchem der Apache Tomcat Server installiert werden soll.



7. Abschlussbildschirm des Installations-Programmes.



UNIX

Eine detaillierte Beschreibung der Installation und De-Installation des Applikations Servers Apache Tomcat finden Sie hier: Apache Tomcat WIKI mit einigen 'HowTo'

Änderung der IP-Adresse für die Dienste des Apache Tomcat

Nach dem Start des Applikations-Servers Tomcat stellt dieser seine Dienst über den Port 8080 auf der IP-Adresse '0.0.0.0'/localhost zur Verfügung. Da Tomcat jedoch seine Dienste entfernt arbeitenden Auftrags-Systemen und EOMS-Workern über das Netzwerk zur Verfügung stellt, muss Tomcat im Netzwerk unter einer IP-Adresse verfügbar sein.

Lassen Sie sich von Ihrem Systemadministrator die IP-Adresse des Servers nennen, auf welchem Sie Tomcat installiert haben und unter welcher dieser Service im Netzwerk verfügbar gemacht werden soll (Dienste: ipconfig/ifconfig zur Analyse der IP-Adresse des Servers).

Um Apache Tomcat an diese IP-Adresse zu binden, öffnen Sie die Konfigurationsdatei *'[INSTALL_DIR]\conf\server.xml'* in einem Editor. Suchen Sie dort nach folgendem Eintrag:

```
...
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
address="localhost"
redirectPort="8443" />
...
```

Ersetzen Sie die IP-Adresse '0.0.0.0' durch die IP-Adresse oder den HOST-Namen des Servers auf welchem ActiveMQ installiert ist.
Beispiel:

```
...
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
address="192.168.118.49"
redirectPort="8443" />
...
```

Starten Sie nun den Tomcat Dienst neu.
Prüfen ob der Applikations-Server Tomcat erfolgreich gestartet worden ist

---

Haben Sie Tomcat erfolgreich installiert und als System-Service gestartet, so sollte der TCP/IP-Port 8080 verfügbar sein.

Dies können Sie unter Windows oder UNIX durch das Dienstprogramm 'netstat' (Wikipedia NETSTAT) prüfen.

```
> netstat -an
...
TCP 192.168.118.49:8080 0.0.0.0:0 ABHOEREN
...
```

Datenbanken

Das EOMS-Core benötigt zur Speicherung seiner Informationen eine Datenbank. Zum Einsatz kommen damit eine Reihe von relationalen SQL-Datenbanken wie folgt:

- Derby (Apache)

  \+ unterstützte/getestete Versionen 10.6.X

- MySQL (Oracle)

  \+ unterstützte/getestete Versionen 5.0.X, 5.6.X

- MSSQL-Server (Microsoft)
  \+ unterstützte/getestete Versionen 2008

Bei der Installation/Auslieferung des EOMS ist die Derby-Datenbank vorinstalliert und kann für Test-/Evaluationszwecke genutzt werden.

Diese Derby-Datenbank wird mit dem Applikations-Container (Apache Tomcat) zusammen gestartet und gestoppt und benötigt damit keinen administrativen Eingriff.

Für den produktiven Betrieb wird die Derby-Datenbank jedoch nur eingeschränkt empfohlen und supportet.

Das R-S setzt einen der o.g. SQL-Datenbank-Server voraus, welcher in produktiven Kundenumgebungen in der Verantwortung durch den Kunden installiert/konfiguriert und betrieben (incl. Backup) wird. Ein Datenbank-Server sollte in produktiven Umgebungen des EOMS-Core immer dediziert, im besten Fall auf einem anderen Server-System als auf dem R-S Server betrieben werden.

Konfiguration der Datenbank

Beachten Sie, dass Sie eventuell in der ***eoms.invoker.properties***-Datei die Voreinstellungen an Ihre Datenbank anpassen müssen.

Datenbank - Derby

Nutzen Sie die Derby-SQL-Datenbank als Datenbank-Server (Auslieferungszustand des EOMS), so finden Sie eine vorinstallierte Datenbank nach einer fehlerfreien Konfiguration vor, welche Sie sofort nutzen können. Dieser Artikel beschreibt die Einrichtung einer Derby-Datenbank für das EOMS.

Zum erfolgreichen Start des Kommando-Zeilen-Client "ij" des Derby-Datenbank-Servers wird eine Umgebungsvariable **DERBY_HOME** benötigt.

Diese Variable zeigt auf auf das Verzeichnis, in welchem der Kommando-Zeilen-Client "ij" installiert ist.

Standardmäßig befindet sich dieses Verzeichnis unterhalb des Installation-Verzeichnisses des Applikations-Servers Apache Tomcat

an folgender Stelle: *"[EOMS]\bin\eoms_helper\derby_admin".* Es gibt mehrere Möglichkeiten, Umgebungsvariablen im Betriebssystem

oder für den Nutzer zu setzen, welcher den Applikations-Server startet. Sprechen Sie hier mit Ihrem Systemadministrator.

1. Bezug von Derby

Falls auf Ihrem System noch kein Apache Derby vorinstalliert ist, ist der 1. Schritt die Installation und Einrichtung eines Derby-Servers. Beziehen können Sie derby hier.

Wählen Sie die für Sie passende Distribution. Die Installation von Derby geschieht einfach durch Entpacken in das gewünschte Verzeichnis. Vergessen Sie danach nicht, **DERBY_HOME** zu setzen (siehe oben). Eine ausführliche Installationsbeschreibung finden Sie in der Apache Derby Dokumentation.

2. Starten des Servers und des Kommandozeilen-Clients

Starten Sie den Datenbank-Server von derby, indem Sie im Verzeichnis {DERBY_HOME}/lib eine Eingabeaufforderung öffnen und dort folgenden Befehl ausführen:

```
> java -jar derbyrun.jar server start
```

Warten Sie, bis der Server hochgefahren ist und öffnen Sie dann den Kommandozeilen-Client **ij**, indem Sie im Verzeichnis {DERBY_HOME}/bin eine Eingabeaufforderung öffnen und dort folgenden Befehl ausführen:

```
> ij
```

Wenn Sie den Kommando-Zeilen-Prompt des **ij** angezeigt bekommen, können Sie als nächsten Schritt die Datenbank anlegen:

Führen Sie folgenden Befehl aus, um eine neue EOMS-Datenbank anzulegen:

```
ij > connect 'jdbc:derby://<host>:<port>/eoms;create=true' user 'eoms'
password 'eoms';
```

Hinweis: Dabei ist <host> die IP-Adresse oder der Hostname des Derby-SQL-Datenbank-Servers (z.B. localhost oder 192.168.128.1) und <port>

der TCP/IP-Port des Derby-SQL-Datenbank-Servers (Standard: 1527).

Es wird dringend empfohlen, hier nicht das Standardpasswort zu verwenden. Wenn Sie andere Werte als 'eoms' verwenden, müssen Sie dies in der Konfigurationsdatei des EOMS-Core anpassen.

Falls bereits eine EOMS-Datenbank vorhanden ist, Sie diese aber neu initialisieren wollen (Vorsicht: Alle Daten gehen dabei verloren!), führen Sie folgenden Befehl aus:

```
ij > connect 'jdbc:derby://<ip>:<port>/eoms;create=false' user 'eoms'
password 'eoms';
```

Hinweis: Für 'localhost' können Sie die IP-Adresse oder Host-Namen des Derby-SQL-Datenbank-Servers angeben (z.B. localhost oder 192.168.128.1) und für <port> geben Sie

den TCP/IP-Port des Derby-SQL-Datenbank-Servers an (Standard: 1527).

Um nun die neu angelegte / neu initialisierte Datenbank für das EOMS zu konfigurieren, führen Sie das mitgelieferte SQL-Skript aus:

```
ij > run 'PATH_TO_SCRIPT/eoms.derby.X_Y.sql';
```

Siehe: http://db.apache.org/derby/manuals/#docs_10.0

Wobei eoms.derby.X_Y.sql das beim EOMS mitgelieferte Datenbankskript ist und X_Y die EOMS-Versionsnummer. Wenn bei Ihnen kein Skript mitgeliefert wurde, können Sie das aktuelle Skript für derby hier herunterladen.

Damit das Core eine Verbindung zur Datenbank herstellen kann, müssen Sie in der eoms.invoker.properties die Datenbankverbindung entsprechend konfigurieren. Mehr dazu finden Sie hier.

Datenbank - MySQL

Für den produktiven Einsatz ist derby nur eingeschränkt geeignet. Deshalb wird empfohlen, hier MySQL (oder Alternativ: MSSQL) einzusetzen. Dieser Artikel beschreibt die Einrichtung der EOMS-Datenbank auf einem MySQL-Server.

1. Bezug von MySQL

Falls auf Ihrem System noch kein MySQL installiert ist, ist der 1. Schritt die Installation und Einrichtung eines MySQL-Servers. MySQL gibt es von verschiedenen Anbietern, die verbreitetste und empfohlene MySQL-Distribution ist MySQL von Oracle. Prinzipiell sind Sie bei der Wahl der Distribution aber frei.

Die MySQL Community Edition von Oracle können Sie hier beziehen. Achten Sie darauf, das richtige Paket für Ihr Betriebssystem und Ihre Systemarchitektur auszuwählen. Installieren Sie Server und Client, alle anderen Erweiterungen werden nicht benötigt.

Merken Sie sich unbedingt den Benutzernamen und das Passwort, das Sie bei der Installation angeben.

2. Start des MySQL-Servers und des Kommandozeilen-Clients

Starten Sie den Datenbank-Server von MySQL, indem Sie im Verzeichnis {MYSQL_SERVER}/bin eine Eingabeaufforderung öffnen und dort folgenden Befehl ausführen:

```
> mysqld
```

Starten Sie dann den Kommandozeilen-Client, indem Sie im Verzeichnis {MYSQL_SERVER}/bin eine neue Eingabeaufforderung öffnen und dort folgenden Befehl ausführen:

```
> mysql --host <host> --port <port> -u <user> -p
> <password>
```

Dabei ist <host> die IP-Adresse oder der Hostname des MySQL-Datenbank-Servers (z.B. 192.168.128.1, wenn der MySQL Server auf dem selben System läuft (Host = localhost) kann dieser Parameter weggelassen werden) und <port> der TCP/IP-Port des MySQL-Datenbank-Servers (Standard: 3306, wenn der Standardport verwendet wird kann dieser Parameter weggelassen werden), <user> der Benutzername, den Sie bei der Installation angegeben haben und <password> das dazugehörige Passwort. Achten Sie darauf, die Befehle getrennt einzugeben.

3. Anlegen der Datenbank

Führen Sie folgenden Befehl aus, um eine neue EOMS-Datenbank anzulegen:

```
mysql > CREATE DATABASE IF NOT EXISTS eoms;
```

Um nun die neu angelegte / neu initialisierte Datenbank für das EOMS zu konfigurieren, führen Sie das mitgelieferte SQL-Skript aus:

```
mysql > use eoms;
mysql > source 'PATH_TO_SCRIPT/sql/eoms.mysql.X_Y.sql';
```

Wobei eoms.mysql.X_Y.sql das beim EOMS mitgelieferte Datenbankskript ist und X_Y die EOMS-Versionsnummer. Wenn bei Ihnen kein Skript mitgeliefert wurde, können Sie das aktuelle Skript für mysql hier herunterladen.

4. Konfiguration des EOMS-Core

Damit das Core eine Verbindung zur Datenbank herstellen kann, müssen Sie in der eoms.invoker.properties die Datenbankverbindung entsprechend konfigurieren. Mehr dazu finden Sie hier.

Datenbank - MSSQL

Für den produktiven Einsatz ist derby nur eingeschränkt geeignet. Deshalb wird empfohlen, MySQL oder MSSQL einzusetzen. Dieser Artikel beschreibt die Einrichtung der EOMS-Datenbank auf einem MSSQL-Server.

1. Bezug von MSSQL

Falls auf Ihrem System noch kein MSSQL installiert ist, ist der 1. Schritt die Installation und Einrichtung eines MSSQL-Servers. MSSQL können Sie direkt von Microsoft beziehen (unter Umständen wird hier ein Microsoft-Konto benötigt). Installieren Sie MSSQL Server mit Hilfe des mitgelieferten Installers.

⚠ Falls Sie SQL Server Express verwenden, müssen Sie eventuell noch SQL Server Management Studio Express herunterladen, um per Kommandoclient auf den SQL Server verbinden zu können.

⚠ Merken Sie sich unbedingt den Benutzernamen und das Passwort, das Sie bei der Installation angeben.

2. Start des MSSQL-Servers

Sie können den MSSQL-Server entweder über den mitgelieferten Konfigurations-Manager oder direkt per Konsole starten.

A. Starten über den SQL Server Konfigurations Manager (empfohlen)

Öffnen Sie den SQL Server Konfigurations Manager (z.B. über Startmenü - Microsoft SQL Server 2014 - Konfigurationstools - SQL Server Konfigurations Manager).

Wählen Sie dann im Konfigurationsmanager in der linken Übersicht **SQL Server Dienste**. Mit Rechtsklick - Starten auf SQL Server (MSSQL) in der rechten Übersichtstabelle starten sie den SQL Server. Analog können Sie auch andere Aktionen auf dem Server ausführen, z.B. Stoppen oder Neustart.

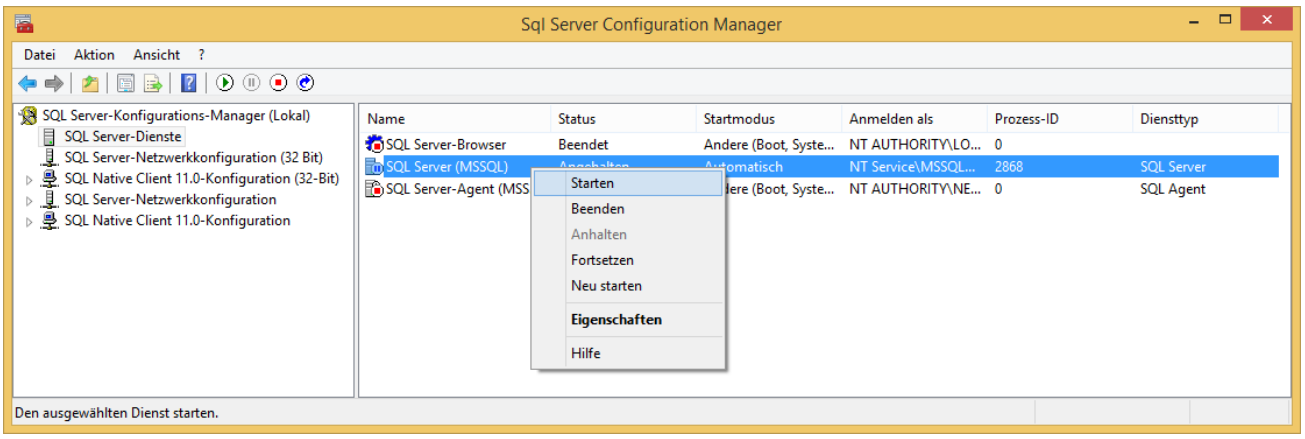


Abbildung A: SQL Server Konfigurations Manager

Der MSSQL-Server sollte nun gestartet und bereit für eingehende Verbindungen sein.

B. Starten über die Kommandozeile (nicht empfohlen)

> ⚠ Diese Methode wird für den produktiven Einsatz nicht empfohlen. Sie sollte nur zur Fehlerbehandlung eingesetzt werden, falls der Server aus irgendeinem Grund nicht mit Methode A gestartet werden konnte.

Starten Sie den Datenbank-Server von MSSQL, indem Sie im Verzeichnis {MSSQL_SERVER}/Binn eine Eingabeaufforderung öffnen und dort folgenden Befehl ausführen:

```
> sqlservr.exe
```

3. Verbindung zum SQL-Server per Kommandozeilen-Client herstellen

Achten Sie darauf, dass sqlcmd.exe verfügbar ist. Führen Sie dann folgende Befehle in einer Eingabeaufforderung aus:

```
> sqlcmd -S <host>,<port> -U <user> -P <password>
```

Dabei ist <host> die IP-Adresse oder der Hostname des MSSQL-Datenbank-Servers (z.B. 192.168.128.1, wenn der MSSQL Server auf dem selben System läuft (Host = localhost) kann dieser Parameter weggelassen werden) und <port> der TCP/IP-Port des MSSQL-Datenbank-Servers (Standard: 1919, wenn der Standardport verwendet wird kann dieser Parameter weggelassen werden), <user> der Benutzername, den Sie bei der Installation angegeben haben und <password> das dazugehörige Passwort. Achten Sie darauf, die Befehle getrennt einzugeben.

4. Anlegen der Datenbank

Führen Sie folgenden Befehl aus, um eine neue EOMS-Datenbank anzulegen:

```
sqlcmd > CREATE DATABASE eoms;
```

Verlassen Sie sqlcmd. Öffnen Sie eine neue Eingabeaufforderung.

Um nun die neu angelegte / neu initialisierte Datenbank für das EOMS zu konfigurieren, führen Sie das mitgelieferte SQL-Skript aus:

```
> sqlcmd -S <host>,<port> -U <user> -P <password> -i
PATH_TO_SCRIPT/sql/eoms.mssql.X_Y.sql
```

Wobei eoms.mssql.X_Y.sql das beim EOMS mitgelieferte Datenbankskript ist und X_Y die EOMS-Versionsnummer. Wenn bei Ihnen kein Skript mitgeliefert wurde, können Sie das aktuelle Skript für mysql hier herunterladen.

5. Konfiguration des EOMS-Core

Damit das Core eine Verbindung zur Datenbank herstellen kann, müssen Sie in der eoms.invoker.properties die Datenbankverbindung entsprechend konfigurieren. Mehr dazu finden Sie hier.

Installation - EOMS-Worker

Im Gegensatz zum EOMS-Core benötigt der Worker keine vorherige Installation einer Laufzeitumgebung, sondern es genügt, die Workerdateien in das gewünschte Verzeichnis zu kopieren und dann die eoms.invoker.client.properties so anzupassen, dass die Worker eine Verbindung zum EOMS-Core aufbauen können. Dazu müssen in erster Linie die Verbindungsdaten dort angepasst werden (Übertragungsart, Hostadresse, Port). Konfigurieren Sie die eoms.invoker.client.properties dazu so wie hier beschrieben. Eine umfassende Beschreibung der Konfigurationsdatei finden Sie hier.

Kurzuebersicht Konfiguration

Hier soll eine Kurzübersicht gegeben werden, welche Minimalkonfiguration in Core und Worker für ein lauffähiges EOMS nötig sind. Dabei wird nur auf die Herstellung einer Verbindung zwischen Core, Worker und Auftrags-System eingegangen und nicht auf die Konfiguration der Systeme selbst (es wird mit der Standardkonfiguration gearbeitet). Entnehmen Sie dies den Konfigurationen über Core und Worker.

> Diese Übersicht ersetzt nicht die umfangreicheren Kapitel über Konfiguration im Core und im Worker, sondern soll nur einen Schnelleinstieg in die Grundkonfiguration des EOMS bieten.

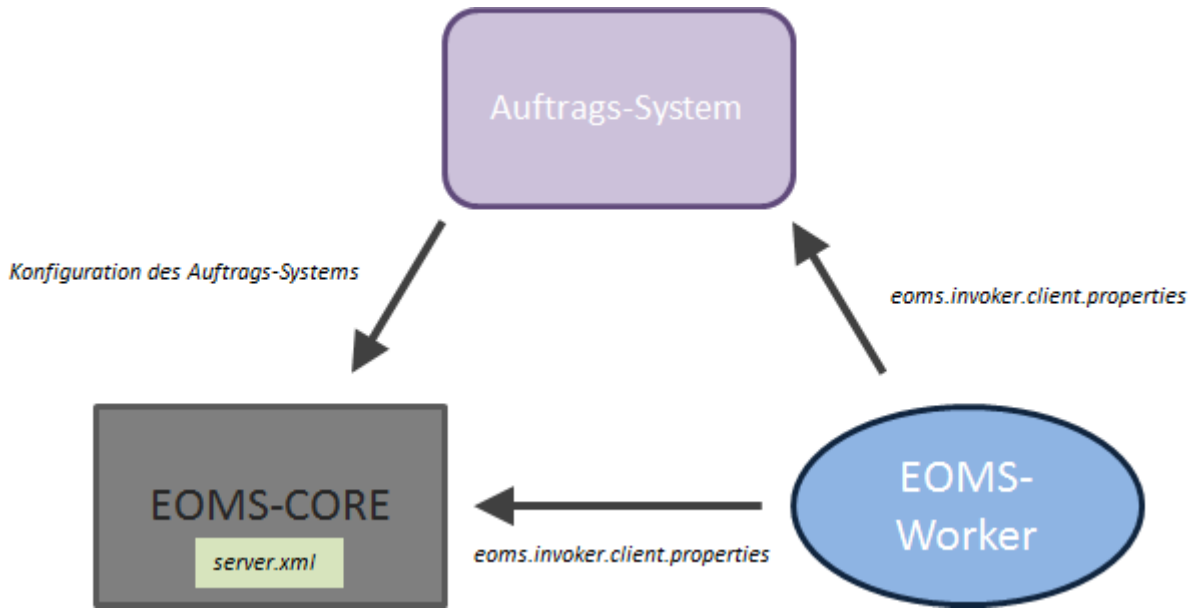


Abbildung A: Vereinfachtes Schema für den Verbindungsaufbau zwischen Core, Worker und Auftrags-System

Wie in *Abb. A* zu sehen sind die relevanten Konfigurationsdaten (Verbindungsdaten) im EOMS nur in der ***server.xml*** für den Core (Server) und in der ***eoms.invoker.client.properties*** (Client) festgelegt. Die Konfiguration des Auftrags-System hängt von dessen Typ ab, hier wird nur beispielhaft stark verkürzt die Anbindung an den Spooler beschrieben . Eine vollständige Anleitung zur Konfiguration von Auftrags-Systemen finden Sie hier.

Schritt 1

Der 1. Schritt bei der Einrichtung des EOMS besteht in der Konfiguration des EOMS-Core als Host durch die server.xml. Eine Anleitung dazu finden Sie auch hier und hier.

Generell muss lediglich sichergestellt werden, dass das EOMS-Core im Netzwerk erreichbar ist. Dazu muss in der server.xml nur der Wert des **address**-Attributs im Connector von "localhost" zu der IP-Adresse des Rechners geändert werden:

```
 ...
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
address="192.168.2.105" <!-- Von localhost zu (Beispiel-)IP
192.168.2.105 -->
redirectPort="8443" />
...
```

Das EOMS-Core ist jetzt (nach einem Neustart) unter 192.168.2.105:8080 erreichbar.

Diese Änderung ist nur dann nötig, wenn der EOMS-Server von außen erreichbar sein soll, was bei den meisten Anwendungsszenarien der Fall ist.

Schritt 2

Der 2. Schritt besteht in der Konfiguration des eoms.invoker.client.properties-Datei des EOMS-Workers. Hier muss sowohl die Verbindung zum EOMS-Core als auch die zum Auftrags-System sichergestellt werden.

Für das EOMS-Core sind dazu in der eoms.invoker.client.properties (des Workers) folgende Einträge relevant:

- eoms.invoker.host (auf IP setzen, also hier 192.168.2.105).
- eoms.invoker.port (auf Port setzen, also hier 8080).

und, falls ⚠ JMS verwendet wird (dann muss aber auch sichergestellt werden, dass der JMS-Server aktiv ist):

- eoms.invoker.jms.host (auf IP des Hosts des JMS-Servers setzen).
- eoms.invoker.jms.port (auf Port des JMS-Servers setzen).

Neben dem EOMS-Core müssen auch die Verbindungsdaten zum Auftrags-System eingetragen werden. Welche Eigenschaften hier relevant sind hängt vom Workertyp und dem spezifischen Auftrags-System ab. Beachten Sie dazu die Konfigurationsseite für Worker.

Für den Spooler als Auftrags-System muss folgende Konfiguration vorgenommen werden (für die Kommunikation über http):

- eoms.resource.spooler.host (auf die IP-Adresse des Spooler-Hosts setzen, also auf die IP-Adresse des Rechners, auf dem der S läuft).
- eoms.resource.spooler.port (auf den http-Port des Spoolers setzen, Standard: 62616).

Für die Kommunikation mit dem Spooler über FileShare (nicht empfohlen):

- eoms.resource.spooler.base-dir (auf das Job-Verzeichnis des Spoolers setzen, Standard: <PATH_TO_SPOOLER>/data/SpoolIn)
- eoms.resource.spooler.temp-dir (auf das temporäre Job-Verzeichnis des Spoolers setzen, Standard: <PATH_TO_SPOOLER>/data/SpoolIn/temp).

Dies richtet einen Standard Ressourcen-Fetcher ein. Sie können aber beliebige weitere Fetcher anlegen, siehe dazu eoms.invoker.client.properties.

Schritt 3

Der 3. Schritt ist die Einrichtung des Auftrags-System. Beim Spooler genügt es, unter Prozessteuerung ein neues Programm mit folgenden Werten anzulegen:

Programm = *eoms*

Parameter = *eoms.invoker.host=<HOST_IP> eoms.invoker.port=<HOST_PORT> eoms.invoker.user=<USER> eoms.invoker.password=<PASSWORD> eoms.invoker.spooler=XXX eoms.invoker.jms.host=polling eoms.invoker.jms.port=polling eoms.invoker.transfer=http eoms.process=copy copy.output=@SourceName.dat*

- *wobei <HOST_IP> die IP-Adresse des EOMS-Core ist (hier: 192.168.2.105) und <HOST_PORT> der Port, auf das EOMS-Core g wird (hier: 8080). Diese Aufrufzeile gilt für die Kommunikation über http ohne JMS. Für andere Varianten siehe Anbindung Auftrags-Systeme.*
- *wobei eoms.invoker.spooler=XXX nur nötig ist, wenn der EOMS-Worker so eingerichtet sind, dass sie mehrere Fetcher (Kommun zu mehreren Spoolern) besitzen. Dann ist XXX der Name des Fetchers in der eoms.invoker.client.properties des Workers, der angesprochen werden soll.*
- *wobei <user> und <password> die Anmeldedaten für den EOMS-Core sind (Standard: eoms - eoms).*

Danach müssen Sie in der Prozesssteuerung einen neuen Task anlegen (z.B. CHECK EOMS). In diesem neuen Task fügen Sie ein TaskItem hinzu, welches das neue EOMS-Programm verwendet. Vergessen Sie nicht, den neuen Task zu den Inputtasks hinzuzufügen (damit der Task auch ausgeführt wird). Mehr Informationen dazu finden Sie in der Spooler-Dokumentation.

Testen Sie nun, ob zwischen den Einzelsystemen erfolgreich Verbindungen aufgebaut werden können. Konsultieren Sie bei Problemen die Konfigurationsseiten Core und Worker und die Anbindung an Auftrags-Systeme oder kontaktieren Sie gegebenenfalls unseren Suppo rt.

Systemsicherheit

Ist der Applikations-Server, die SQL-Datenbank und EOMS-Core korrekt installiert, sollten noch einige Aspekte der Systemsicherheit erörtert werden.

Apache Tomcat Applikations-Server

Nach der Installation des Apache Tomcat Server ist zu empfehlen, folgende Verzeichnisse aus dem Installationsverzeichnis des Tomcat Applikations-Servers zu löschen:

- /webapps/examples
- /webapps/docs
- /webapps/manager
- /webapps/host-manager
- /conf/catalina/localhost (die Dateien 'host-manager.xml' und 'manager.xml')
- /work/Catalina/localhost (wird dynamisch neu vom Tomcat Applikations-Server erstellt)

Sollten Sie die Manager-Applikation des Tomcat Applikations-Servers nutzen wollen, so sichern Sie diese nach den Vorgaben aus der Administrator-Dokumentation des Tomcat Appliaktions-Servers.

Apache Derby Datenbank

Nach der Standard-Installation sind keine System-Nutzer auf der Derby-SQL-Datenbank aktiviert, somit kann sich jeder User ohne Angabe von Nutzer-ID und Passwort auf die Derby Datenbank verbinden.

Ist dies nicht gewollt, so gibt es zwei Möglichkeiten, den Zugang zu erschweren oder zu autorisieren.

Bindung des Host-Prozesses der Derby Datenbank an die IP-Adresse LOCALHOST und einen anderen Port als im Standard

Im Auslieferungszustand des EOMS-Core ist die Derby Datenbank an die lokale IP-Adresse (LOCALHOST) des Servers installiert, auf welchem das EOMS-Core installiert ist.

Dies ist bezüglich der Systemsicherheit die effektivste Methode um den direkten Zugriff auf die Derby Datenbank von entfernten Computern zu unterbinden.

Desweiteren ist zu empfehlen, den Standard-TCP/IP-Port der Datenbank auf einen anderen Wert als 1527 (Standard-Port der Derby Datenbank) zu setzen.

Den im Auslieferungszustand genutzten TCP/IP-Port der Derby Datenbank entnehmen Sie aus Sicherheitsgründen der Administrator-Dokumentation.

Nutzung der Nutzer-Autorisierung der Derby Datenbank

Bevor der Derby Datenbank-Server über authentifizierte Nutzer genutzt werden kann, müssen diese auf der Derby Datenbank angelegt werden.

Nutzen Sie dazu den von Apache mitgelieferten Command-Line-Client 'ij' oder grafische SQL-Clients, welche in der Lage sind, sich auf die Derby SQL-Datenbank zu verbinden.

Befehl/Beispiel zur Anlage eines neuen System-Nutzers auf der Derby Datenbank

```
CALL SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY('derby.user.XXX', 'YYYY')
```

Wobei 'XXX' der Name des Nutzers und 'YYY' das Passwort des Nutzers in Klartext ist.

Befehl/Beispiel zum Setzen der notwendigen Rechte für den User 'XXX'

```
CALL
SYSCS_UTIL.SYSCS_SET_DATABASE_PROPERTY('derby.database.fullAccessUsers'
,'XXX')
```

Mit der Datei 'derby.properties' im Datenbankverzeichnis der Derby Datenbank wird nun festgelegt, dass der Derby Datenbank-Server mit einer User-Authentifizierung arbeiten soll:

```
derby.connection.requireAuthentication=TRUE
derby.authentication.provider=BUILTIN
derby.user.XXX=YYYY
```

Wobei 'XXX' der Name des Nutzers und 'YYY' das Passwort des Nutzers in Klartext ist.

Es ist seitens des Betriebssystems darauf zu achten, dass die Konfigurations-Datei des Derby Servers nicht von unberechtigten Usern gelesen werden kann.

Lesen Sie die Dokumentation des Derby Datenbank-Servers, um weitere 'Authentication-Provider' wie LDAP oder Betriebssystem zu nutzen.

Änderungen bezüglich der IP-Adresse/Port der Derby Datenbank, sowie des standardmäßigen System-Nutzers müssen in den entsprechend korrespondierenden Konfigurations-Dateien des EOMS-Core nachgepflegt werden.

Grundsätzlich sind die Vorgaben der Produkt-Hersteller bezüglich Systemsicherheit oder die Sicherheitsanforderungen in den Kundenumgebungen umzusetzen.

Verwendete Software

Nachfolgend wird aufgelistet, welche externe Software das EOMS verwendet / benötigt:

- Messaging Server - ActiveMQ ab Version 5.3
- Apache Tomcat Version 8.X *(optional: JBOSS 5.X).*
- Apache Derby *(optional MS SQL oder MySQL)*
- Adobe Flash Player (immer die aktuell von Adobe bereitgestellte Version)
- Java JDK 1.8.

Konfiguration und Struktur des EOMS-Core

Die Konfiguration des EOMS-Core erfolgt ausschließlich über das Ändern von Kofigurationsdateien, insbesondere von **eoms.invoker.properties**.

Konfiguriert wird der Core hauptsächlich in den Dateien

- **conf/eoms.invoker.authentication** für die Benutzerverwaltung.
- **conf/eoms.invoker.properties** für die Konfiguration des Core.
- **conf/server.xml** für die Konfiguration des Apache Tomcat (sollte, abgesehen von der IP-Angabe, nicht verändert werden).

Für das EOMS 1.3 wird Tomcat 8 vorausgesetzt. Die Verzeichnisstruktur dieser Distribution unterscheidet sich eventuell von der vorhergehender Versionen.

Um Ihnen einen Überblick über die Verzeichnisstruktur des EOMS-Core zu geben, sind im Folgenden alle Unterordner von %EOMS-CORE_HOME% aufgeführt.

- /bin
- /conf
- /data
- /lib
- /logs
- /temp
- /webapps
- /work

/bin

- Enthält die Start/Stopp-Skripte von Tomcat bzw. tomcatX.exe/tomcatXw.exe *(X = Versionsnummer)* zum Starten / Stoppen von T

/conf

- Enthält die Konfigurationsdateien von Tomcat und des EOMS-Core. Wichtige Konfigurationsdateien sind (blau hinterlegte Dateier werden separat behandelt):

eoms.invoker.authentication	Legt Nutzer, Passwörter und Benutzerrollen fest.
eoms.invoker.properties	Konfigurationsdatei des EOMS-Core.
catalina.properties	Konfigurationsdatei von Tomcat.
server.xml	Server-Konfiguration von Tomcat.
logging.properties	Steuert das Logging von Tomcat.

/data

- Enthält die Datenbank des EOMS-Core. Je nach verwendetem Datenbanksystem unterscheidet sich der Aufbau der Datenbankdateistruktur. Datenbanken liegen in nach dem Datenbanksystem benannten Unterverzeichnissen von /data (z.B. /data Abhängig vom verwendeten Datenbankserver kann dieses Verzeichnis auch leer sein, die Datenbank wird dann nicht lokal abgele

/lib

- Enthält die Programmbibliotheken / Klassendateien von Tomcat, Datenbankservern und EOMS-Core.

/logs

- Enthält die Logging-Dateien von Tomcat und EOMS-Core. Konsultieren Sie diese Logdateien, falls unerwartete Fehler aufgetrete Wichtige Log-Dateien sind:

root.log	Loggingdatei des Tomcat-Servers.
eoms.log	Loggingdatei des EOMS-Servers.

/temp

- Temporärer Ordner.

/webapps

- Enthält das EOMS-Core Web Application Archive (.war), den Kern der Anwendung.

/work

- Enthält die aus der EOMS-Core.war Datei generierten Servlet Sourcecode Dateien und daraus kompilierte Java Class Files.

eoms.invoker.authentication

Es wird dringendst empfohlen, die Standardbenutzer und Passwörter nach der Installation zu ändern!

Seit Version 1.3 werden Passwörter als PBKDF2 Hashes gespeichert.

In der eoms.invoker.authentication-Datei wird festgelegt, welche Benutzer im System existieren, welche Rollen diese Benutzer haben (welche Rechte) und die dazugehörigen Passwörter. Jeder Eintrag (jede Zeile) entspricht einem Benutzerdatensatz. Die eoms.invoker.authentication-Datei enthält nur Benutzerdatensätze der folgenden Form:

```
USERNAME=PASSWORD_AS_HASH,ROLE_1,ROLE_2,ROLE_n
```

Wobei ***ROLE*** eine der vordefinierten Benutzerrollen ist und die Rechte des Benutzers im System angibt (Was darf der Nutzer).

Es existieren folgende Rollen:

Rolle	Berechtigung
ROLE_ROOT	EOMS-Locks verwalten, EOMS-Prozesse erstellen, aktualisieren, abfragen und terminieren (darf alles). Wird z.B. vom Spooler und Workern verwendet.
ROLE_ADMIN	EOMS-Prozesse terminieren und die Prozessverarbeitung anhalten / fortsetzen. Wird z.B. von der grafischen Oberfläche Flex-GUI verwendet.
ROLE_OPERATOR, ROLE_A, ROLE_B, ROLE_1, ROLE_2	Nur zu Demonstrationszwecken. Wird momentan im System nicht verwendet.

Passwort generieren

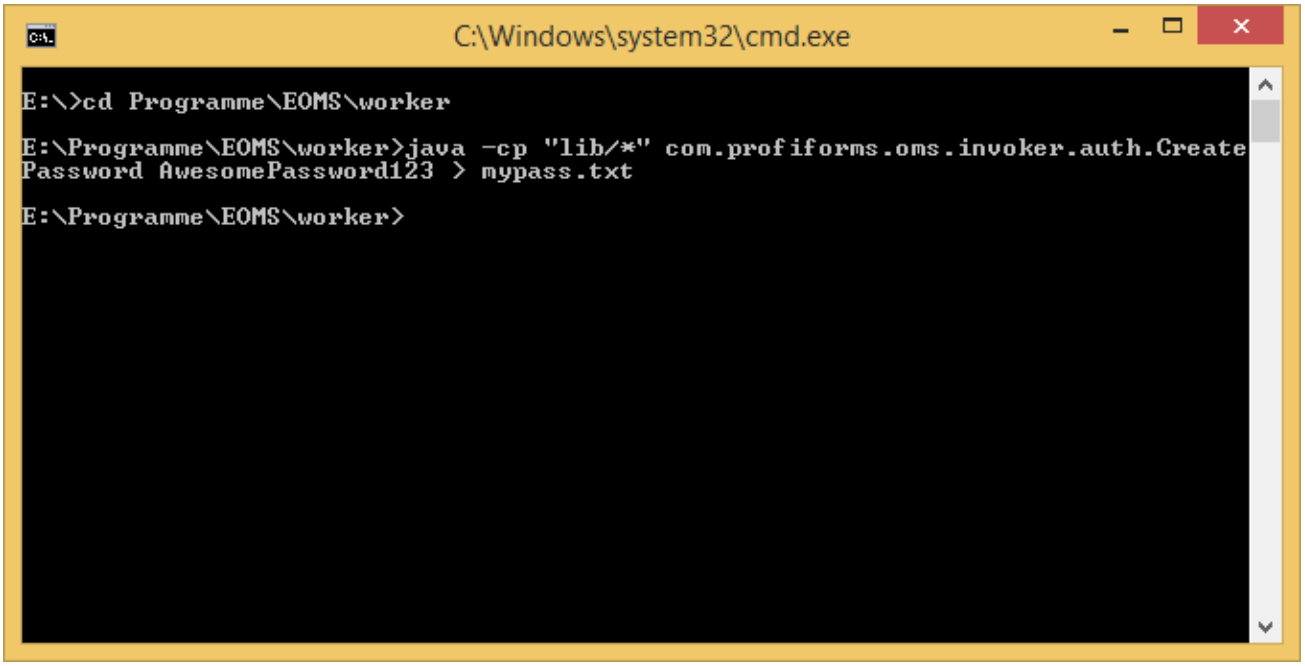
Da Passwörter als Hashes gespeichert werden, müssen Sie Ihr Wunschpasswort zuvor als Hash generieren lassen, bevor Sie den Nutzer in der eoms.invoker.authentication eintragen.

Dafür steht Ihnen ein leicht zu bedienendes Kommandozeilentool zur Verfügung, dass Sie mit folgendem Befehl starten können:

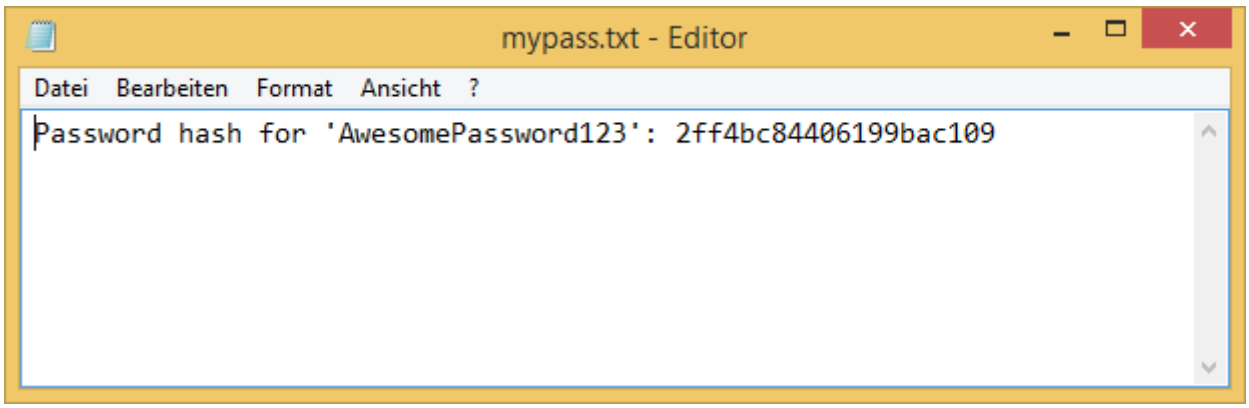
```
Java -cp „lib/*" com.profiforms.oms.invoker.auth.CreatePassword
<password>
```

Wobei Sie als <password> das Passwort angeben, für das der Hashwert ausgegeben werden soll.

Beachten Sie, dass Sie diesen Befehl im *%WORKER_HOME%*-Verzeichnis ausführen müssen! (starten Sie die Eingabeaufforderung direkt im Homeverzeichnis oder wechseln Sie dorthin mit dem Befehl ***cd***). Falls Sie den Hashwert in eine Datei ausgeben wollen, können Sie nach dem Befehl noch **> *filename*** anhängen:



In der mypass.txt im *%WORKER_HOME%*-Verzeichnis steht dann der Hashwert für "AwesomePassword123":



Diesen Hashwert müssen Sie dann in der eoms.invoker.authentication als Password-Wert eintragen.

Beispieleintrag:

```
simon=2ff4bc84406199bac109,ROLE_ROOT
```

Hier wird ein Benutzer mit dem Benutzernamen "simon" und dem Passwort "AwesomePassword123" angelegt. Der Benutzer bekommt mit **ROLE_ADMIN** das Recht zum Terminieren von Prozessen und zum Stoppen/Starten der Prozessverarbeitung zugewiesen. Er darf keine EOMS-Prozesse erstellen, aktualisieren oder terminieren und keine EOMS-Locks verwalten.

Ist der Benutzer in der eoms.invoker.authentication eingetragen, können Sie sich als dieser Benutzer anmelden.

eoms.invoker.properties

Dieser Artikel beschreibt die Konfiguration des EOMS-Core über Eintragungen in ***conf/eoms.invoker.properties***.

Einträge in der **eoms.invoker.properties** überschreiben die Standardwerte des EOMS. Prinzipiell sind also alle Direktiven optional. Fehlt eine Direktive, wird automatisch der intern festgelegte Standardwert angenommen.

Das EOMS-Core funktioniert nur ordnungsgemäß, wenn die ***eoms.invoker.properties*** richtig konfiguriert wurde.

Öffnen Sie sie dazu mit einem gewöhnlich Texteditor.

Die Datei enthält generelle Einstellungen, die Datenbankkonfiguration und die Verzeichniskonfiguration.

Wichtig ist vor allem die Datenbankkonfiguration, die Sie eventuell vor dem ersten Start an Ihre Datenbank anpassen müssen.

Allgemeine Konfiguration

Eigenschaft	Standardwert
eoms.invoker.server-name	—
eoms.invoker.process.pessimistic-locking-exception.retries	3
eoms.invoker.process.pessimistic-locking-exception.sleep	1000

- ***eoms.invoker.server-name***
  - Legt den Servernamen fest (beliebig wählbar und optional).
- **eoms.invoker.process.pessimistic-locking-exception.retries**

- Legt fest, wie viele Zugriffsversuche auf eine Ressource gemacht werden sollen, die sich in einem Deadlock befindet (eir wird durch einen anderen Prozess verwendet und ist dadurch gesperrt, die Ressource wird aber auch nicht freigegeben, verwendende Prozess selbst auf eine Ressource / ein Ereignis wartet).

- ***eoms.invoker.process.pessimistic-locking-exception.sleep***
  - Legt fest, wie lange bei einem Deadlock nach einem Zugriffsversuch gewartet werden soll, bis ein neuer Versuch gemacl

Prozesspoolkonfiguration

Die Konfiguration des Prozess-Pools kann unter Umständen starke Auswirkungen auf die Performance und Effizienz Ihres EOMS haben.

Beachten Sie hierzu unbedingt auch die Beschreibung Prozess-Pool.

Eigenschaft	Standardwert
eoms.invoker.process-pool.size	100
eoms.invoker.process-pool.type	cluster
eoms.invoker.process-pool.cluster.nodes	20
eoms.invoker.terminated-process-collector.hours	24

- ***eoms.invoker.process-pool.size***
    - Legt fest, wie viele Prozesse eine ⚠ Parzelle fasst. Der gesamte Prozess-Pool fasst also size * nodes Prozesse, in der Standardkonfiguration 2000.

- **eoms.invoker.process-pool.type**
    - Legt den Typ des Prozesspools fest (cluster | default):
        - **cluster**: Für jeden Prozesstyp wird eine Parzelle im Prozesspool reserviert. Ein Prozess wird nur zum Prozesspo entsprechende Parzelle einen freien Platz hat, sonst muss der Prozess so lange warten, bis ein Prozess dieses und die Parzelle wieder Prozesse aufnehmen kann.
        - **default**: Es werden so lange Prozesse zum Pool hinzugefügt, bis dieser voll ist, danach müssen alle Prozesse w wird ein beliebiger Prozess in den Pool hinzugefügt. Es wird nicht nach Prozesstypen unterschieden. Sinnvoll, w verschiedene Prozess-Typen verarbeiten.

- ***eoms.invoker.process-pool.cluster.nodes***
    - Legt die Anzahl an Parzellen fest. Eine Parzelle fasst nur Prozesse von genau einem Prozess-Typ. Wenn nodes auf 20 g können also maximal 20 verschiedene Prozesstypen im Prozess-Pool gelagert werden.

- **eoms.invoker.terminated-process-collector.hours**
    - Legt fest, wie viele Stunden terminierte Jobs gespeichert bleiben sollen, bevor sie aus der Datenbank entfernt werden.

Die optimale Konfiguration des Prozess-Pools hängt stark von der Struktur Ihrer Prozessverteilung und von den Kapazitäten Ihres Systems ab. Prinzipielle Empfehlungen:

- Bei wenigen Prozessen und wenigen unterschiedlichen Prozess-Typen: Standardwerte.
- Bei vielen Prozessen, aber wenigen unterschiedlichen Prozess-Typen (< 20): size-Attribut eventuell erhöhen, nodes-Attribut Stan oder verringern.
- Bei wenigen Prozessen, aber vielen unterschiedlichen Prozess-Typen (> 20): size-Attribut eventuell verringern, nodes-Attribut erh
- Bei vielen Prozessen und vielen unterschiedlichen Prozess-Typen (> 20): size-Attribut erhöhen, nodes-Attribut erhöhen.

ⓘ Falls Sie Performanceprobleme haben hilft es eventuell, die Werte von size und nodes zu verringern, um weniger Kapazitäten bereitzustellen.

Spoolerkonfiguration

ⓘ Neu in Version 1.3!

⚠ Benötigt Spooler-Version 3.8.1 oder höher!

Für eine schnellere Reaktionszeit des Spoolers bei beendeten Prozessen kann der Spooler per Callback vom EOMS benachrichtigt werden. Dabei wird der Spooler-Status nach beendeten Prozessen über Callbacks abgefragt. Dies muss aber mit folgenden Direktiven entsprechend eingerichtet werden:

Eigenschaftswert	Standardwert
eoms.invoker.process-terminate-callback-dispatcher.min-threads	10
eoms.invoker.process-terminate-callback-dispatcher.max-threads	150
eoms.invoker.process-terminate-callback-dispatcher.queue-capacity	500
eoms.invoker.process-terminate-callback-dispatcher.keep-alive	60000

- ***eoms.invoker.process-terminate-callback-dispatcher.min-threads***
  - Minimale Anzahl an gleichzeitigen Dispatcher-Threads für Callback.
- ***eoms.invoker.process-terminate-callback-dispatcher.max-threads***
  - Maximale Anzahl an gleichzeitigen Dispatcher-Threads für Callback.
- ***eoms.invoker.process-terminate-callback-dispatcher.queue-capacity***
  - Größe der Callback-Warteschlange; Maximale Anzahl terminierter Jobs in der Warteschlange.
- ***eoms.invoker.process-terminate-callback-dispatcher.keep-alive***
  - Zeit in ms, wie lange ein Thread ohne ausgeführten Dispatch existieren darf, bevor er beendet wird.

Datenbankkonfiguration

Momentan enthält die ***eoms.invoker.properties*** Voreinstellungen zu den 3 unterstützten Datenbanksystemen **derby**, **mssql**, und **mysql** .

In Version 1.3 wird postgreSQL nicht mehr als Datenbanksystem unterstützt. Standard ist derby.

Eigenschaft	Standardwert	Mögliche Werte
eoms.invoker.dao	derby	derby mssql mysql

eoms.invoker.dao Legt das verwendete Datenbanksystem fest. Wenn Sie keine Derby-Datenbank verwenden müssen Sie den Wert auf den richtigen Typ ändern (Stellen Sie auch dann, wenn Sie derby verwenden, sicher, dass die Eigenschaft auf derby gesetzt ist).

Neben dem Datenbanksystem müssen Sie auch den Treiber und die URL der Datenbank angeben. Für die 3 unterstützten Datenbanksysteme sind bereits auskommentierte *(beginnend mit #)* Standardwerte eingetragen:

```
...
# mssql, mysql, derby
eoms.invoker.dao = derby

####db settings

##derby
#eoms.invoker.db.driver=org.apache.derby.jdbc.ClientDriver
#eoms.invoker.db.url=jdbc:derby://localhost:1527/eoms;create=false

##mssql
#eoms.invoker.db.driver=com.microsoft.sqlserver.jdbc.SQLServerDriver
#eoms.invoker.db.url=jdbc:sqlserver://localhost:1433;databaseName=eoms

##mysql
#eoms.invoker.db.driver=com.mysql.jdbc.Driver
#eoms.invoker.db.url=jdbc:mysql://localhost:3306/eoms
...
```

Entfernen Sie die Raute vor den 2 Zeilen der von Ihnen verwendeten Datenbank. Stellen Sie dabei sicher, dass alle anderen Zeilen durch eine beginnende Raute auskommentiert sind. (In der obigen Konfiguration müssten Zeile 12 und 13 auskommentiert werden, da in ***eoms.invoker.dao*** derby als verwendete Datenbank festgelegt ist). eoms.invoker.db.driver und eoms.invoker.db.url dürfen nur für einen Typ angegeben (auskommentiert) werden!

Die vorgegebene Treiberangabe und die Connection-URL sind nur Standardwerte des jeweiligen Datenbanksystems. Passen Sie die Werte gegebenenfalls an, falls Sie beim Einrichten ihrer Datenbank andere Werte verwendet haben. Natürlich müssen Sie auch "localhost" durch die entsprechende IP-Adresse des Hostrechners ersetzen, auf dem der Datenbankserver läuft, falls dieser nicht auf dem gleichen System wie das EOMS-Core gehostet wird.

Zuletzt müssen Sie noch den Datenbankbenutzer und das zugehörige Passwort angeben, mit dem sich das EOMS an der Datenbank anmelden kann:

Eigenschaft	Standardwert
eoms.invoker.db.user	eoms
eoms.invoker.db.password	eoms

Die Werte von ***eoms.invoker.db.user*** und ***eoms.invoker.db.password*** müssen mit den Werten, die Sie beim Erstellen der Datenbank angegeben haben, übereinstimmen.

 **Beispiel:** Hier wird beschrieben, wie Sie die Datenbank anlegen, wenn Sie derby als Datenbanksystem verwenden. Dabei lautet der Befehl zum Erstellen der Datenbank (siehe Originalseite):

```
ij> connect 'jdbc:derby://localhost:XXXX/eoms;create=true' user 'eoms'
password 'eoms';
```

Sie können statt 'eoms' auch einen anderen Nutzernamen und Passwort bei der Erstellung verwenden, müssen diese Werte dann aber hier anpassen. Für andere Datenbanksysteme erfolgt die Vergabe des Nutzernamens und des Passworts eventuell anders, siehe hier für MySQL und MSSQL.

Neben dieser grundlegenden Datenbankkonfiguration, die unbedingt korrekt vorgenommen werden muss, gibt es noch weitere Anpassungsmöglichkeiten:

Eigenschaft	Standardwert
eoms.invoker.db.connect-retries	3
eoms.invoker.db.connect-timeout	5000

- ***eoms.invoker.db.connect-retries***
  - Legt fest, wie oft ein fehlgeschlagener Verbindungsversuch zur Datenbank wiederholt werden soll, bevor abgebrochen w

- ***eoms.invoker.db.connect-timeout***
  - Legt fest, wie lange *(in ms)* gewartet werden soll, bis nach einem fehlgeschlagenen Verbindungsversuch zur Datenban Aufbau versucht werden soll (nur bis ***eoms.invoker.db.connect-retries*** erreicht).

Konfiguration der Verbindung zum Messaging-Server

Das EOMS-Core verfügt über zwei Methoden Status-Informationen über entgegengenommene Aufträge an den Auftraggeber zurückzumelden.

Methode 1: Der Auftraggeber überträgt nur den abzuarbeitenden Auftrag an das EOMS-Core. Alle weiteren Informaionen über den Auftragsstatus erhält der Auftraggeber über einen Messaging-Server. Dazu stellt das EOMS-Core Auftragsinformationen über einen Messaging-Server zur Verfügung, welche asynchron vom Auftraggeber gelesen und verarbeitet werden können.

Methode 2: Der Auftraggeber überträgt den abzuarbeitenden Auftrag an das EOMS-Core. Danach ruft der Auftraggeber zyklisch Informationen über diesen Auftrag vom EOMS-Core ab. Diese Methode ist einfacher zu konfigurieren und benötigt keine weitere System-Komponente wie ein Messaging-Server. Der Nachteil besteht jedoch darin, dass durch eine sehr hohe Anzahl von Auftragsanfragen direkt an des EOMS-Core, das EOMS-Core stark belastet werden kann.

Mit folgenden Eigenschaften wird festgelegt, ob und wie mit einem Messaging-Server kommuniziert werden soll:

Eigenschaft	Standardwert	Mögliche Werte
eoms.invoker.messaging	stub	stub apollo
oms.invoker.messaging.apollo.uri		tcp://localhost:63616

- ***oms.invoker.messaging***
    - Gibt an, ob und mit welchem Messaging-Server kommuniziert werden soll.
    - Wert: stub - die Kommunikation mit einem Messaging-Server ist ausgeschaltet.
    - Wert: apollo - Es wir über einen Apache Apollo-Server (JMS) kommuniziert. In diesem Fall muss auch ***oms.invoker.mes uri*** gesetzt sein.

- ***oms.invoker.messaging.apollo.uri***
    - Gibt die Kommunikations-URL des Messaging-Servers an (diese kann im Messaging-Server konfiguriert werden).

Verzeichnisdienstkonfiguration

> Die Verzeichnisdienstkonfiguration ist momentan ohne Funktion, wird aber eventuell in zukünftigen Releases für die Kommunikation mit Active Directory benötigt werden.

Mit folgenden Eigenschaften werden der Standort und die Anmeldedaten für den EOMS-Verzeichnisdienst festgelegt. Im Regelfall brauchen Sie diese Werte nicht ändern.

Eigenschaft	Standardwert
oms.invoker.ds.url	localhost
oms.invoker.ds.user	eoms
oms.invoker.ds.password	eoms

- ***oms.invoker.ds.url***
    - Gibt den Host an, auf dem der Verzeichnisdienst läuft.

- ***oms.invoker.ds.user***
    - Gibt den Benutzername zur Anmeldung am Verzeichnisdienst an.

- ***oms.invoker.ds.password***
    - Gibt das Passwort zur Anmeldung am Verzeichnisdienst an.

server.xml

Dieser Artikel deckt nur die Einrichtung eines Connectors des Apache Tomcat Servers ab. Vollständige Informationen finden Sie auf der Apache Tomcat Homepage.

Ändern Sie die ***conf/server.xml*** wirklich nur wenn nötig!

Grundsätzlich sollten Sie keine Änderungen an der Konfiguration des Tomcat Servers vornehmen. Einziger Ausnahmefall ist hier die IP-Angabe und Port-Angabe im Connector, damit das EOMS-Core auch im Netzwerk erreichbar ist. Solche Änderungen werden in der server.xml über sogenannte Connectors vorgenommen.

Öffnen Sie dazu die ***conf/server.xml***-Datei und suchen Sie folgenden Eintrag:

```
...
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
...
```

Das EOMS-Core ist wie hier konfiguriert (diese Konfiguration ist Standard) nur auf dem Hostrechner über http://localhost:8080/omsinvoker erreichbar.

Der Standardeintrag stellt einen Connector an Port 8080 auf localhost bereit. D.h. standardmäßig ist das EOMS-Core nur auf dem Hostrechner über den Port 8080 aufrufbar. Fügen Sie das Attribut **address** hinzu, um das EOMS-Core auch im Netzwerk zur Verfügung zu stellen *(Vergleiche hierzu Anpassungen bei der Installation).* **address** muss dabei die IP-Adresse des Hostrechners zugewiesen werden *(z.B. mit ifconfig/ipconfig herauszufinden)*. Ändern Sie den Wert von **port**, um EOMS-Core auf einem anderen Port zur Verfügung zu stellen. Beachten Sie dann unbedingt, dass sämtliche Konfigurationen im EOMS auf den neuen Port angepasst werden müssen. Dazu gehört auch die Konfiguration des Workers.

```
...
<Connector address="21.120.213.231" port="5678" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
...
```

Das EOMS-Core ist nun unter http://21.120.213.231:5678/omsinvoker erreichbar (auch aus dem Netzwerk/Internet).

Konfiguration und Struktur der EOMS-Worker

Die Konfiguration des EOMS-Workers erfolgt ausschließlich über das Ändern von Kofigurationsdateien. Diese finden sich in /config und in /config/spring.

Konfiguriert werden Worker hauptsächlich in den Dateien

- **conf/*.properties** für allgemeine Workereinstellungen und die List of Tasks
- **conf/eoms.invoker.client.properties** für Verbindungskonfiguration, Verzeichniskonfiguration, etc.
- ***.rcml** für die RCML-Spezifikation

*Wobei * der Workertyp ist (Von diesen Konfigurationsdateien gibt es pro Workertyp 1 Konfigurationsdatei, die nur für diesen Worker gilt).*

Die Verzeichnisstruktur hat sich im Vergleich zu früheren Versionen (v.a. im Vergleich zu 1.1) des EOMS geändert:

- Es gibt keine AKI-Worker mehr, daher wurden die folgenden Dateien entfernt:
  - start-aki-worker.cmd
  - config/aki.properties
  - config/aki.rcml
  - config/spring/eoms.invoker.client.aki.xml
- Die config.properties wurde in build-workers.properties umbenannt.
- Die eoms.invoker.client.properties-Datei wurde von /config/spring nach /config verschoben.
- Folgende Dateien wurden in /config/spring entfernt:Einige Bibliotheken in /lib wurden hinzugefügt / geändert.
  - eoms.invoker.client.properties.xml
  - eoms.invoker.client.xml

Um Ihnen einen Überblick über die komplette Verzeichnisstruktur des EOMS-Workers zu geben, sind im Folgenden alle Unterordner von %EOMS-WORKER_HOME% aufgeführt (Die Verzeichnisstruktur ist für alle Workertypen gleich):

- /~home
- /config
- /spring
- /lib
- /logs
- /WORK
- /WORKER

/~home

- Das Homeverzeichnis enthält unter anderem die Startskripts.

build-workers.properties	Enthält keine eigenen Konfigurationswerte, sondern nur Angaben zur Konfiguration.
start-oms-worker.cmd	Startet einen **OMS-Worker**.
start-eomsinput-worker.cmd	Startet einen **EOMS-Input-Worker**.
history.txt	Enthält die *What's New* für jeden bisher erschienenen Build.
dependencies.txt	Enthält Angaben zu den Mindestanforderungen an andere Systeme.

/config

- Enthält die Konfigurationsdateien für den Worker. Dabei besitzt jeder Workertyp eine eigene .properties-Datei und eine eigene .rcml-Datei.

eoms.invoker.client.properties	Enthält Verbindungs- und Verzeichnisangaben.
eoms-input.properties	Enthält die Konfiguration für EOMS-Input-Worker.
worker.properties	Enthält die Konfiguration für OMS-Worker.
eoms-input.rcml	Enthält die RCML-Spezifikation für EOMS-Input-Worker.
worker.rcml	Enthält die RCML-Spezifikation für OMS-Worker.
log4j.xml	Enthält die Log4J-Konfiguration.

/spring

- Enthält Konfigurationsdateien des Spring-Frameworks, das von EOMS-Worker als Plattform genutzt wird.

eoms.invoker.client.eoms-input.xml	Enthält Eigenschaftswerte für EOMS-Input-Worker.
eoms-input.remote.xml	Konfigurationsdatei für EOMS-Input-Worker.

/lib

- Enthält die Bibliotheken und Klassendateien des Workers, des Spring Frameworks, etc. Hier sollten keine Veränderungen vorgen werden.

/logs

- Enthält die Logging-Dateien des Workers. Beachten Sie, dass hier die Logdateien sämtlicher gestarteter Worker abgelegt werden
  Konsultieren Sie diese Logdateien, falls unerwartete Fehler aufgetreten sind.

/WORK

- Enthält die Zwischenspeicherungen der Jobdaten. Für jeden Job wird ein eigener Ordner angelegt.

/WORKER

- Enthält die vom Auftrags-System empfangenen Ressourcen. Kann durch die **eoms.resource.spooler.fetcher.work-dir**-Eigensch
  geändert werden. Achten Sie darauf, diese temporären Dateien in regelmäßigem Abstand zu löschen.

*.properties

Die **/config/*.properties**-Datei legt unter anderem die List Of Tasks des Workertyps fest. Die Liste enthält alle ausführbaren Prozesse auf diesem Worker. Diese Prozesse müssen dann aber auch in der ***.rcml** des Workertyps definiert sein.

Workerkonfiguration

Im oberen Teil der *.properties werden allgemeine Konfigurationswerte des Workers festgelegt.

Eigenschaftsname	Beschreibung	Standardwerte
eoms.worker.name	Name des Workertyps, für den diese Konfigurationsdatei bestimmt ist.	—
eoms.worker.consumers-number	Die maximale Anzahl an gleichzeitigen Prozessen pro Worker. Für OMS-Worker wird empfohlen, diesen Wert maximal auf die Anzahl Kerne der CPU zu setzen.	• EOMS-Input: 10 • OMS: 1
eoms.worker.sleep-time	Die Zeit in ms, die ein Worker wartet, bis er am EOMS-Core nach einem neuen Job anfragt, nachdem er erfolglos angefragt hat.	1000

List Of Tasks

Der 2. Teil der *.properties (nach "*#list of processes accepted by consumer*") enthält die List of Tasks.

Diese besteht aus beliebig vielen Paaren von:

eoms.processN	Gibt den Namen eines Prozesses an, der auf diesem Worker ausgeführt werden kann.
eoms.processN.group	Gibt die Gruppe des Prozesses an.

Wobei N = Fortlaufend nummeriert.

Jeder erlaubte Prozess muss hier aufgeführt und in der *.rcml definiert sein.

Beispiel: Die Standard-List Of Tasks für OMS-Worker:

```
...
# list of processes accepted by consumer
eoms.process = copy
eoms.process.group = *

eoms.process1 = rw
eoms.process1.group = *

eoms.process2 = *
eoms.process2.group = *
```

Dieser Standardworker kann also die Prozesse "copy", "rw" und "*" ausführen. "*" ist die sog. Wildcard und bedeutet, dieser Worker akzeptiert alle Prozesse die ihm aufgetragen werden. Dazu muss in der *.rcml ein entsprechender Eintrag vorhanden sein, der für alle Prozessnamen ausser "copy" und "rw" gilt. mehr dazu in RCML Kompendium.

Die Werte, die Sie eoms.processN zuweisen, müssen exakt mit der id des entsprechenden Prozesses in der RCML-Daten für diesen Workertyp übereinstimmen.

Zuweisung von Prozessen an Konsumenten

Neu in Version 1.3!

Sie können jetzt auch für einzelne Konsumenten angeben, welche Tasks an diese Konsumenten geliefert werden sollen. Eine Beispielkonfiguration dazu könnte so aussehen:

```
...

# Anzahl an Konsumenten für den Worker
eoms.worker.consumers-number = 10

# Diese Prozesse werden von allen(!) Konsumenten akzeptiert:
eoms.process.[*] = pdf
eoms.process.[*].group = *

# Diese Prozesse auch:
eoms.process1.[*] = write
eoms.process1.[*].group = *

# Diese Prozesse aber nur von den Konsumenten 1,2,3 und 5:
eoms.process.[1-3,5] = copy
eoms.process.[1-3,5].group = copy-group

# Diese Prozesse nur von den Konsumenten 4-10.
# Da es nur 10 Konsumenten gibt, wird die Angabe für Konsumenten #11-15
ignoriert:

eoms.process1.[4-15] = print
eoms.process1.[4-15].group = *

...
```

Sie müssen diese Notation allerdings nicht verwenden, sondern können Ihre Zuweisungen auch ganz gewohnt ohne Konsumenteneinschränkung, wie in List of Tasks beschrieben, vornehmen.

eoms.invoker.client.properties

Die eoms.invoker.client.properties ist die Hauptkonfigurationsdatei des Workers. Sie enthält vor allem die Verbindungsdaten zum EOMS-Core (bzw. bei indirekter Kommunikation zum JMS), die Verzeichnisdaten für den Kontakt zum Spooler, für die EOMS-Input-Worker FTP- und SFTP-Konfiguration und sichere SSL-Verbindungen.

1. Allgemeine Konfiguration

Eigenschaftswert	Beschreibung	Standardwert
eoms.invoker.worker-service	Neu in Version 1.3 Gibt an, über welche Schnittstelle der Worker Jobs empfangen und versenden soll: • **REST**: Über ein REST-Interface. • **REMOTE**: Über das Standard-Interface. Falls Problemen in der Kommunikation auftreten, sollten Sie hier REST verwenden.	remote (rest / remote)
eoms.invoker.worker-registry	ehemals 'eoms.invoker.registry' Gibt an, welche Schnittstelle der Worker ansprechen soll, um sich am Core zu registrieren: • **REST**: Über ein REST-Interface, • **REMOTE**: Über das Standard-Interface. Falls Problemen in der Kommunikation auftreten, sollten Sie hier REST verwenden.	remote (rest / remote)

2. Verbindungskonfiguration zum EOMS-Core

Der 2. Teil der eoms.invoker.client.properties enthält die Daten für den Verbindungsaufbau zum EOMS-Core:

Eigenschaftswert	Beschreibung	Standardwert
eoms.invoker.protocol	Das Übertragungsprotokoll, über dass das EOMS-Core einen Verbindungsaufbau ermöglicht (http / https).	http
eoms.invoker.host	Die IP-Adresse, auf dem das EOMS-Core gehostet wird, z.B. **82.176.126.42** oder **localhost**.	localhost (gleicher Rechner)
eoms.invoker.port	Der Port, auf dem das EOMS-Core gehostet wird, z.b. 8080 oder 8443.	8080
eoms.invoker.user	Gibt den Benutzer an, mit dem sich der Worker anmelden soll.	user
eoms.invoker.password	Gibt das Passwort an, mit dem sich der Worker anmelden soll.	password
eoms.invoker.read-timeout	Maximale Zeit in ms, die zwischen 2 Leseoperationen vom Verbindungssocket vergehen darf.	60000
eoms.invoker.connection-timeout	Zeit in ms, wie lange der Core keine Operation auf dem Verbindungssocket ausführen darf, bevor die Verbindung beendet wird.	120000
eoms.invoker.jms.host	Die IP-Adresse, auf dem der Messaging-Server gehostet wird, z.B. **82.176.126.42** oder **localhost**. Nur nötig, wenn auch ein Messaging-Server genutzt wird (indirekte Kommunikation).	localhost
eoms.invoker.jms.port	Der Port, auf dem der Messaging-Server gehostet wird, z.B. **63616**.	63616
eoms.invoker.messaging	Gibt an, ob die Kommunikation direkt oder über JMS (Apollo) stattfinden soll.	stub

Durch bloße Angabe von eoms.invoker.jms.host und eoms.invoker.jms.port wird noch nicht festgelegt, dass auch JMS verwendet werden soll. Setzen Sie dazu die eoms.invoker.messaging-Eigenschaft.

3. Resource-Fetcher Konfiguration

Damit der OMS-Worker Jobs vom Auftragssystem abholen kann, müssen entsprechende Resource-Fetcher eingerichtet sein, in denen Sie angeben, von wo und wie der Worker diese Daten empfangen kann.

> In Version 1.3 ist es möglich, mehrere Auftragssysteme (Spooler) gleichzeitig an einen OMS-Worker anzubinden. In Versionen älter als Version 1.2.3 oder war dies nicht möglich, ein Worker konnte nur Aufträge von einem Spooler empfangen.

Eigenschaftswert	Beschreibung	Standardwert
eoms.resource.spooler.fetcher.work-dir	Legt fest, in welchem Verzeichnis die Ressourcedaten, die vom Spooler empfangen werden, zwischengespeichert werden sollen.	./WORKER/fetcher-work
eoms.invoker.worker.base-dir	Obsolet in Version 1.3! Arbeitsverzeichnis des Workers.	worker

Sie können prinzipiell seit Version 1.2.4 beliebig viele Ressource-Fetcher anlegen. Die Deklaration für einen Fetcher besteht dabei aus folgenden Direktiven:

Eigenschaftswert	Beschreibung
eoms.resource.spooler.XXX.base-dir	Legt das Verzeichnis fest, unter dem sich der Worker am Spooler die Jobdaten abholen kann. Nur relevant, wenn FileShare und kein http (http empfohlen!) als Übertragungsmethode verwendet wird.
eoms.resource.spooler.XXX.temp-dir	Legt das temporäre Verzeichnis des Spoolers für Jobdaten fest. Nur relevant, wenn FileShare und kein http (http empfohlen!) als Übertragungsmethode verwendet wird.
eoms.resource.spooler.XXX.protocol	Legt das Übertragungsprotokoll zwischen EOMS und Spooler fest (http / https).
eoms.resource.spooler.XXX.host	Die Hostadresse, auf dem das Auftragssystem (Spooler) erreichbar ist.
eoms.resource.spooler.XXX.port	Der Port des Hostrechners, auf dem das Auftragssystem (Spooler) lauscht (Standard Spooler: 62616).

Dabei ist **XXX** der Name des Ressource-Fetchers ist. Dieser ist beliebig wählbar, Sie müssen aber für mehrere Fetcher unterschiedliche Namen wählen.

Des Weiteren ist es erlaubt (und auch meistens wünschenswert), einen Standardfetcher einzurichten. Dieser wird ohne Namen deklariert, also z.B. eoms.resource.spooler.base-dir statt eoms.resource.spooler.XXX.base-dir.

Sie können nur einen Standardfetcher einrichten. Dieser ist dann immer der initial ausgewählte Fetcher, falls Sie nicht explizit einen anderen Fetcher verwenden.

Eine Konfiguration mit Standardfetcher + Fetcher 'spool2' könnte also z.B. so aussehen:

```
...

## Ressource Fetcher


# Standardfetcher
eoms.resource.spooler.protocol = http
eoms.resource.spooler.host = 192.168.128.84
eoms.resource.spooler.port = 62616

# HTTPS-Fetcher auf 2. Spooler
eoms.resource.spooler.spool2.protocol = https
eoms.resource.spooler.spool2.host = 192.168.128.85
eoms.resource.spooler.spool2.port = 62616

...
```

Neben der Konfiguration im EOMS-Worker müssen Sie auch die jeweiligen Spooler so konfigurieren, dass sie den richtigen Fetcher ansprechen. Falls Sie nicht den Standard-Fetcher ansprechen wollen, müssen Sie in der Aufrufzeile im Spooler (siehe hier) folgenden Parameter hinzu:

```
eoms.invoker.spooler=XXX
```

Wobei XXX = Name des Fetchers

also hier z.B. **eoms.invoker.spooler=spool2**.

Sie können einen HTTP- und einen FileShare-Fetcher mit dem gleichen Namen anlegen. Welcher der Fetcher verwendet werden soll wird automatisch über die Übermittlungsmethode im Spooler erkannt.

4. EOMS-Input Konfiguration

Eigenschaftswert	Beschreibung	Standardwert
eoms.input.service.protocol	Das Übertragungsprotokoll, über das der EOMS-Input Worker eine Verbindung zum EOMS-Core aufbaut.	http
eoms.input.service.host	Die IP-Adresse, auf dem der EOMS-Input Server (Empfänger) gehostet wird.	localhost
eoms.input.service.port	Der Port, auf dem der EOMS-Input Server (Empfänger) gehostet wird.	8080
eoms.input.user	Gibt den Benutzer an, mit dem sich der Worker am EOMS-Input Server anmelden soll.	rw
eoms.input.password	Gibt das Passwort an, mit dem sich der Worker am EOMS-Input Server anmelden soll.	rw

5. FTP- und SFTP-Konfiguration

Eigenschaftswert	Beschreibung	Standardwert
eoms.input.producer.workdir	Arbeitsverzeichnis.	. (aktuelles Verzeichnis)
eoms.input.producer.ftp.host	IP-Adresse des FTP-Hosts für die EOMS-Input Einlieferung.	localhost (gleicher Rechner)
eoms.input.producer.ftp.port	Port des FTP-Dienstes für die EOMS-Input Einlieferung.	21
eoms.input.producer.ftp.user	Benutzername, mit dem sich der Worker am FTP-Server anmelden soll.	erw
eoms.input.producer.ftp.password	Passwort, mit dem sich der Worker am FTP-Server anmelden soll.	erw
eoms.input.producer.sftp.host	IP-Adresse des SFTP-Hosts für die EOMS-Input Einlieferung.	—
eoms.input.producer.sftp.port	Port des SFTP-Dienstes für die EOMS-Input Einlieferung.	22
eoms.input.producer.sftp.user	Benutzername, mit dem sich der Worker am SFTP-Server anmelden soll.	—
eoms.input.producer.sftp.password	Passwort, mit dem sich der Worker am SFTP-Server anmelden soll.	—
eoms.input.producer.sftp.timeout	Zeit in ms, wie lange gewartet werden soll, bis nach einem fehlgeschlagenen SFTP-Verbindungsversuch ein erneuter Verbindungsaufbau versucht wird. Neu in Version 1.3	30000

eoms.input.producer.sftp.retries	Anzahl an maximalen SFTP-Verbindungsversuch-Wiederholungen, bis kein weiterer Verbindungsaufbauversuch unternommen wird. Neu in Version 1.3	3

6. SSL-Konfiguration

Nutzen Sie folgende Attribute, um das EOMS für eine sichere SSL-Verbindung einzurichten. Nur wenn der EOMS-Core per https erreichbar ist, können Sie eine SSL-Verbindung einrichten!

Eigenschaftswert	Beschreibung	Standardwert
eoms.invoker.init-ssl-environment	Nur wenn hier true gesetzt wird erlaubt der Worker SSL-Verbindungen (HTTPS).	true
eoms.input.ssl.validation	Für echte SSL-Verbindungen muss hier **true** gesetzt sein.	false
eoms.input.ssl.truststore-file	Gibt den Pfad zur Datei an, die das SSL-Zertifikat speichert. Kann z.B. eine .pem- oder .crt-Datei sein.	—
eoms.input.ssl.truststore-password	Gibt das Passwort für das **eoms.input.ssl.truststore-file** an.	—
eoms.input.ssl.ssl-protocols	Gibt die verwendeten (erlaubten) Protokolle an.	SSLv3,TLSv1

*.rcml

Die *.rcml-Datei enthält die RCML-Spezifikation für den jeweiligen Worker. Hier werden die Prozesse, die der Worker ausführen kann, mit Hilfe von RCML definiert. Dabei sind Ihnen fast unbegrenzte Möglichkeiten gegeben. Für jeden Workertyp existiert eine eigene RCML, die Sie nach belieben anpassen können. Der Inhalt der .rcml-Spezifikationsdateien wird hier nicht behandelt, sondern im Kapitel über RCML: worker.rcml und eoms-input.rcml (Die AKI-RCML wird in dieser Dokumentation aufgrund eingestellter Unterstützung nicht mehr behandelt).

Einstieg in RCML

RCML Kompendium

Das wichtigste Element eines Workers ist seine RCML-Datei. In dieser RCML-Datei stehen die Prozessdefinitionen, die dem Worker "sagen, was er zu tun hat". Diese RCML-Datei enthält, wie der Name schon sagt, RCML-Markup. Über dieses Markup ist es Ihnen möglich, den Worker völlig frei zu konfigurieren. Somit können Sie Worker für fast jeden Einsatzzweck einrichten.

Jeder Workertyp enthält greift auf seine eigene RCML zurück. Das heißt: Die RCML-Prozessdefinitionen für den OMS-Worker befinden sich in /config/oms.rcml und die für den EOMS-Input-Worker in /config/eoms-input.rcml. Dies ermöglicht es Ihnen, die verschiedenen Workertyp völlig unabhängig voneinader zu Konfigurieren. Ein Prozess, der in der oms.rcml definiert wurde, wird von einem EOMS-Input-Worker überhaupt nicht beachtet.

Jeder Worker kommt mit vorgefertigten Standard-RCML's, die grundlegende Prozesse wie ReportWriter oder das Senden an den EOMS-Input-Server enthalten. Sie können diese Standard-RCML's umändern oder auch komplett verwerfen und von Grund auf Ihre eigene RCML schreiben. In diesem Kapitel lernen Sie alles nötige über RCML, um selbst Prozesse definieren zu können, die dann vom Worker ausgeführt werden.

Dieses Kapitel umfasst folgende Kapitel:

1. Ein **Tutorial**, das Ihnen den Einstieg in RCML erleichtern soll,
2. **How-To's**, die typische Anwendungsfälle zeigen,
3. Kommentierte Beschreibungen der **Standard-RCML's,** die im Lieferumfang enthalten sind,
4. Eine komplette **RCML-Elemente-Referenz,** die alle existierenden Elemente auflistet und detailliert beschreibt.

Einfuehrung in RCML

In diesem Tutorial lernen Sie die wichtigsten Grundlagen zur Verwendung von RCML im EOMS. Dieses Tutorial ist vor allem für Anfänger geeignet, die noch nicht mit RCML in Berührung gekommen sind. Wenn Sie schon mit RCML gearbeitet haben, könnte Sie Teil 5: RCML für Fortgeschrittene interessieren.
Was ist RCML?

RCML (Rich Client Markup Language) ist eine an XML angelehnte Auszeichnungssprache, die auch das Einbinden von Skriptsprachen wie JavaScript unterstützt. Das EOMS verwendet RCML für Worker-Prozessdefinitionen. In diesen Prozessdefinitionen (.rcml-Dateien im config-Verzeichnis des Workers) legen Sie mit Hilfe von RCML fest, was ein Worker überhaupt tun soll, wenn er einen Job empfängt. Mit RCML beschreiben Sie also den Ablauf eines Prozesses. Wird ein Job eingeliefert, der für diesen Prozess vorgesehen ist, wird exakt das ausgeführt, was Sie in der entsprechenden .rcml definiert haben.
Wo finde ich die RCML-Dateien?

Die RCML-Prozessdefinitionen für einen Worker finden Sie in dessen /config-Verzeichnis: worker.rcml definiert die Prozesse für einen OMS-Worker (gestartet durch start-oms-worker.cmd) und eoms-input.rcml die Prozesse für einen EOMS-Input-Worker (gestartet durch start-eoms-worker.cmd). In diesen Dateien müssen Sie mit RCML arbeiten. Jeden Workertyp interessiert nur seine eigene RCML-Datei. Das ermöglicht es Ihnen, verschiedene Konfigurationen für die unterschiedlichen Workertypen zu definieren. Schreiben Sie ihre RCML einfach direkt in die .rcml des gewünschten Workertyps. Die anderen Workertypen werden dadurch nicht tangiert.
Wie funktioniert die Ausführung des RCML-Markups?

RCML-Dateien enthalten nur Markup-Code, der vor der Ausführung zuerst Interpretiert werden muss. Diese Übersetzung durch den Worker wird bei jedem einkommenden Job erneut vorgenommen. Dabei kompiliert der Interpreter jedes Element einzeln, entsprechend der Position im Dokument. Das bietet den Vorteil, das RCML-Konfigurationen im laufenden Betrieb geändert werden können, ohne den Worker neustarten zu müssen. Die Änderung wird sofort bei der nächsten Jobausführung erkannt.

Das Tutorial ist in 6 Teile gegliedert:

Als Anfänger sollten Sie unbedingt mit Teil 1 beginnen!

1. Die Syntax der RCML

RCML ist im Prinzip eine auf XML basierende Auszeichnungssprache, die syntaktisch auch aufgebaut ist wie XML. Das macht RCML besonders einfach und schnell zu erlernen. Bevor wir uns mit den semantischen Elementen der Sprache beschäftigen, zeigt dieser Artikel zunächst einmal, wie RCML überhaupt aufgebaut ist und wie Sie RCML notieren.

1. Elemente

Eine RCML-Datei (wie auch XML) besteht ausschließlich aus Elementen *("Tags")*. Welche Elemente es überhaupt gibt ist genau festgelegt, momentan stehen Ihnen ca. 33 Elemente zur Verfügung. Sie notieren Elemente durch den Elementnamen, eingeschlossen in Vergleichszeichen:

```
<Elementname>
```

Damit "öffnen" Sie das Element, d.h. alles, was nachfolgend notiert wird, ist der **Inhalt** des Elements:

```
<Elementname> Hallo, Ich bin Inhalt des Elements.
```

Alles, was Sie jetzt notieren, ist Inhalt des Elements. Natürlich müssen Sie das Element auch wieder "schließen", sonst wäre ja der restliche Code Inhalt des Elements. Geschlossen wird das Element, wie es geöffnet wurde, aber mit einem **/** vor dem Elementnamen:

```
<Elementname> Hallo, Ich bin Inhalt des Elements. </Elementname>
```

Nicht alle Elemente dürfen überhaupt Inhalt enthalten. Für solche Elemente können Sie statt

```
<Elementname></Elementname>
```

auch verwenden:

```
<Elementname />
```

Es gibt auch Elemente, die andere Elemente als Inhalt enthalten dürfen:

```
<Element1>
<Element2>
Hallo, Ich bin der Inhalt des Kindelements!
</Element2>
</Element1>
```

Element2 ist jetzt das *Kindelement* (auch Subelement) von Element1 und Element1 nennt man *Elternelement* von Element2. Element2 hat einen Text als Inhalt. Elemente, die andere Elemente beinhalten dürfen, nennt man *Top-Level-Elemente,* während Elemente, die keine anderen Elemente oder nur Text beinhalten dürfen, sog. *Standalone-Element* sind.

Prinzipiell ist ein RCML-Dokument nichts anderes als Elemente, die wiederrum Kindelemente haben. Die Verschachtelungstiefe ist dabei nicht begrenzt.

Natürlich schreiben Sie nicht einfach irgendwelche Elemente in RCML auf, sondern es existiert eine vorgegebene Palette an Elementen, die Sie notieren können.Hier finden Sie eine Beschreibung aller Elemente. Es führt aber auch nicht zu einem Fehler, wenn Sie ein Element notieren, dass es garnicht gibt:

```
<workdir id="myworkdir" home="./WORK" /> <!-- Real existierendes
RCML-Element: Setzt ein Arbeitsverzeichnis. -->
<MeinElement author="profiforms" /> <!-- Dieses Element gibt es nicht.
Es wird einfach ignoriert. -->
```

Der Interpreter ignoriert Ihr "eigenes" Element allerdings. Es spricht aber nichts dagegen, dass Sie eigene Elemente notieren, um Ihre RCML besser zu strukturieren. Das gleiche darf auch bei Attributen gemacht werden (siehe unten).

<!-- --> markiert 3. Kommentare.

2. Attribute

Attribute beschreiben ein Element näher. Ein Attribut besteht aus dem Attributnamen und dem Attributwert. Attribute werden nach dem Elementnamen notiert, wobei der Attributwert in doppelte Anführungszeichen eingeschlossen wird. Die Zuweisung des Wertes an den Namen erfolgt durch **=**. Mehrere Attribute werden durch Leerzeichen getrennt.

```
<Elementname Attribut1="Wert1" Attribut2="Wert2" />
```

also z.B.:

```
<workdir id="my_workdir" home="./WORK" />
```

Für jedes Element in RCML ist festgelegt, welche Attribute dieses Element hat. Dabei wird unterschieden zwischen obligatorischen und optionalen Attributen. Obligatorische Attribute müssen zwingend notiert werden, sonst stoppt die Ausführung mit einem Fehler.

Beispiel:

Das Element ***<workdir>*** besitzt 3 Attribute: ***id***, ***home*** und ***clear-on-shutdown***. Davon muss nur ***home*** zwingend gesetzt werden (obligatorisch). Folgendes wäre also gültiges RCML:

```
<workdir home="./WORK" />
```

Ebenso gültig wäre z.b.:

```
<workdir id="my_workdir" home="./WORK" clear-on-shutdown="true" />
```

Nicht gültig wäre hingegen, da das obligatorische ***home***-Attribut nicht gesetzt wurde:

```
<workdir id="my_workdir" clear-on-shutdown="true" />
```

Auf die Semantik des Beispiels wird hier noch nicht eingegangen, es soll nur die Verwendung von Attributen veranschaulicht werden.

Sie können jedem Element auch eigene, beliebig benannte Attribute übergeben, die dann allerdings keinen semantische Bedeutung haben. RCML behandelt Attribute, die es nicht erkennt, als wären sie garnicht vorhanden. Dies ermöglicht es Ihnen, zur eigenen Verwendung Attribute zu setzen, z.B., um Ihr RCML-Dokument besser zu strukturieren oder um Elemente zu kennzeichnen:

```
<workdir home="./WORK" author="profiforms"
description="Arbeitsverzeichnis nur für Spooljobs" />
```

Obwohl ***<workdir>*** die Attribute *author* und *description* nicht kennt ist dies trotzdem gülties RCML.

Eine besondere Bedeutung unter den Attributen besitzt id. Über dieses Attribut weisen Sie einem Element eine eindeutige Identifikationsbezeichnung zu, die im gesamten RCML-Dokument nur einmal vorkommen darf. Über diese id können Sie das Element z.B. in anderen Elementen ansprechen.

3. Kommentare

Kommentare sind sinnvoll, um Ihr RCML-Dokument übersichtlich und strukturiert zu halten. Kommentare werden vom Interpreter ignoriert, weshalb Sie innerhalb eines Kommentars notieren können, was Sie wollen. Kommentare orientieren sich an der XML-Syntax und werden durch <!-- eingeleitet und durch --> beendet:

```
<!-- Ich bin ein Kommentar -->
```

Sie können Kommentare überall im Dokument notieren. Alles was zwischen **<!--** und **-->** steht, wird vom Interpreter ignoriert. Mehrzeilige Kommentare sind dabei auch kein Problem.

Kommentare eigenen sich auch, um Elemente auszukommentieren. Elemente werden dabei als Kommentar gekennzeichnet und werden dadurch ignoriert, bis man die Kommentierung wieder entfernt. Sinnvoll ist dies z.B. zu Testzwecken oder weil Sie die Elemente in der aktuellen Konfiguration nicht verwenden wollen, es aber ermöglichen wollen, den Originalzustand durch einfaches Entfernen der Kommentierung wiederherzustellen:

```
<workdir id="myworkdir" home="./WORK" author="profiforms"
description="Arbeitsverzeichnis nur für Spooljobs" />
<!-- <workdir id="myworkdir" home="C:/WORK" author="profiforms"
description="Alternatives Arbeitsverzeichnis nur für Spooljobs" /> -->
```

Hier ist das 2. ***<workdir>*** auskommentiert, wird also vom Interpreter ignoriert und damit auch nicht als Element erkannt. Beachten Sie, dass in diesem Beispiel niemals beide <workdir>-Elemente existieren dürfen (eins muss immer auskommentiert sein), da sie die gleiche ***id*** besitzen (wie wir oben bereits beschrieben haben darf eine id nur einmal vorkommen). Wir könnten natürlich auch unterschiedliche id's vergeben, doch falls wir an einer anderen Stelle auf das Element mit der id "myworkdir" verweisen, würde dieses Element nicht mehr gefunden, nachdem wir das 1. ***<workdir>*** auskommentiert und stattdessen das 2. ***<workdir>*** verwendet hätten. Um jetzt das 1. ***<workdir>*** inaktiv und das 2. ***<workdir>*** aktiv, müssen Sie nur die Kommentartags entfernen und dafür das 1. ***<workdir>*** auskommentieren:

```
<!-- <workdir id="myworkdir" home="./WORK" author="profiforms"
description="Arbeitsverzeichnis nur für Spooljobs" /> -->
<workdir id="myworkdir" home="C:/WORK" author="profiforms"
description="Alternatives Arbeitsverzeichnis nur für Spooljobs" />
```

Sicherlich ist dies ein einfaches und nicht immer sinnvolles Beispiel, doch uns geht es ja erst einmal nur um die Syntax. Bei größeren Codeteilen ist es oft sinnvoller, diese auszukommentieren, falls Sie sie nicht verwenden wollen, statt sie komplett zu löschen. So steht ihnen der Code auch später noch zur Verfügung, hat aber keinen Einfluss auf den restlichen Code.

Damit sind die grundlegenden Sprachelemente von RCML bereits erklärt - Im Grunde sind RCML-Dokumente nur eine Verschachtelung von Elementen, die durch Attribute genauer definiert werden. Für einen etwas tieferen Blick in RCML fahren Sie mit 2. RCML für Fortgeschrittene fort.

2. Die Elementhierarchie

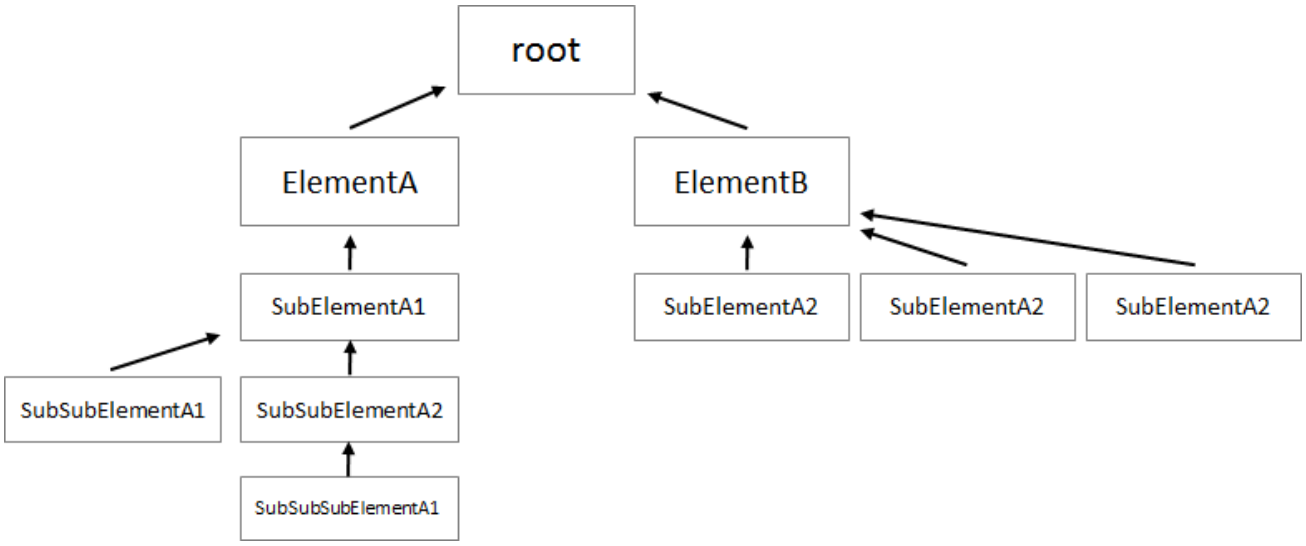
Wie bereits in Teil 1 des Tutorials angedeutet bestehen RCML-Dokumente ausschließlich aus Elementen, die wiederum Attribute, Kindelemente und Inhalt besitzen können. Wir wollen uns in diesem Teil etwas näher mit der Hierarchie eines RCML-Dokuments befassen und wie Elemente notiert werden können. Wie genau RCML-Dokumente dann aufgebaut sein müssen wird erst im 4. Teil beha ndelt.

Betrachten wir dazu folgendes Beispiel (Abstraktion, keine gültigen RCML-Elemente):

```
<root>
<ElementA>
<SubElementA1>
<SubSubElementA1 />
<SubSubElementA2>
<SubSubSubElementA1 />
</SubSubElementA2>
</SubElementA1>
</ElementA>

<ElementB>
<SubElementB1 />
<SubElementB2 />
<SubElementB3>
<SubSubElementB1 />
</SubElementB3>
</ElementB>
</root>
```

Dies erscheint zunächst verwirrend, ist aber hierarchisch gesehen einfach aufgebaut. Wenn wir die RCML-Datei auf der höchsten Ebene betrachten, sehen wir zunächst nur 1 einziges Element: <root>. Syntaktisch gesehen besteht unser RCML-Dokument also nur aus einem einzigen Element. Alle anderen Elemente, die hier notiert sind, sind Kindelemente (Subelemente) von <root>, werden also als dessen Inhalt notiert. Diese Verschachtelung lässt sich beliebig tief anwenden. Wenn wir unser Beispiel komplett "aufklappen" ergibt sich folgende Struktur:



Pfeile signalisieren dabei die Beziehung zwischen Kindelement und Elternelement. Man sieht: <root> ist das Elternelement aller Elemente des Dokuments. root selbst hat 2 direkte Kindelemente: <ElementA> und <ElementB>. Diese wiederum haben ebenfalls Kindelemente, usw.

Elemente können sowohl direkte als auch indirekte Elternelemente sowie direkte als auch indirekte Kindelemente haben. <SubSubElementA1> beispielsweise hat als direktes Elternelement <SubElementA> und als indirekte Elternelemente <ElementA> sowie <root>. <SubElementA1> hat als direkte Kindelemente <SubSubElementA1> sowie <SubSubElementA2> und als indirektes Kindelement <SubSubSubElementA1>. Indirekte Elternelemente nennt man auch Vorfahren, indirekte Kindelemente Nachfahren.

Verschachtelungsebenen:

1. Ebene	2. Ebene	3. Ebene	4. Ebene	5. Ebene
root	ElementA	SubElementA1	SubSubElementA1	
			SubSubElementA2	SubSubSubElementA1
	ElementB	SubElementB1		
		SubElementB2		
		SubElementB3	SubSubElementB1	

Jetzt fragen Sie sich vielleicht, warum man Elemente überhaupt verschachtelt und wozu Eltern-/Kind-Beziehung überhaupt dienen. Betrachten wir dazu folgendes simples Beispiel, über das sich die Modellplatte von Autoherstellern darstellen ließe.

```
<marke>
<audi>
<sportcoupes>
<rs5 />
<rs7 />
</sportcoupes>
<limousinen>
<a4 />
<a5 />
<a6 />
<a7 />
<a8 />
</limousinen>
</audi>

<porsche>
<suv>
<cayenne />
<macan />
</suv>
<sport>
<boxster />
<cayman />
<_911 />
</sport>
</porsche
</marke>
```

Sie sehen also: Der Sinn von Subelementen ist es, das Elternelement genauer zu beschreiben oder zu spezifizieren. In RCML werden Subelemente ähnlich eingesetzt, allerdings darf man hier nicht so statisch denken wie in diesem Beispiel. Vielmehr dienen Subelemente als Elemente, die das Elternelement "logisch umsetzen". Beispiel:

```
<exec ... >
<param />
<commandline>...</commandline>
<result />
</exec>
```

<exec> dient dazu, Befehle auf Betriebssystemebene auszuführen. Wir wollen darauf hier nicht näher eingehen, sondern nur auf den Sinn der Verschachtelung: Das <exec>-Element selbst führt keine Befehle aus, sondern signalisiert RCML nur, dass hier jetzt eine Befehlsausführung folgen soll. Wie dieser Befehl aussehen soll (<commandline>), welche Parameter er besitzt (<param>) und was getan werden soll, falls Fehler auftreten (<result>) wird erst innerhalb des Elements durch Subelemente beschrieben.

Nur wenige RCML-Elemente dürfen überhaupt Subelemente enthalten. Diese sind ***<rcml>, <process>, <docxworld-fetch-production-environment>, <exec>, <if>, < else>, <switch>*** und ***<case>.*** Diese Elemente nennt man deshalb auch Top-Level-Elemente. Es gibt aber auch Elemente, die zwar keine Subelemente enthalten dürfen, dafür aber Inhalt (Text). Diese sind ***<commandline>, <docxworld-contract>, <link-name>, <binary-bundles-home>, <production-bundles-home>, <runtime-home>*** und ***<merge-home>.*** Hier dient der Inhalt als eine Art Wert des Elements. So übergibt man z.b. ***<commandline>,*** das einen Befehl ausführt, eben diesen Befehls als Textinhalt oder ***<binary-bundles-home>,*** das den Pfad zum Homeverzeichnis für Binärpakete aus dem Redaktions-System setzt, eben den Pfad als Textinhalt.

Zum Schluss sollen noch Attribute etwas näher betrachtet werden. Attribute beschreiben ein Element näher und setzen dessen Eigenschaften. Wie bereits in Teil 1 beschrieben besitzt jedes Element unterschiedliche Attribute (wobei manche Attribute bei mehreren Elementen vorkommen), Sie können aber auch immer ihre eigenen Attribute angeben, die dann allerdings keine Wirkung auf das Element haben, sondern nur Ihnen als Hilfe dienen. Attribute gehören immer zu genau einem Element, nämlich zu dem, in dem sie gesetzt wurden. Eltern- oder Kindattribute gibt es nicht.

Eine Sonderstellung nimmt das Attribut **id** ein, das für alle Elemente gesetzt werden kann. Über dieses Attribut kann von überall aus auf das Element zugegriffen und verwiesen werden. Beispiel:

```
<workdir id="myworkdir" home="./WORK" />
...
<delete file="myworkdir" />
```

Hier wird ein **<workdir>** (Arbeitsverzeichnis) mit der id "myworkdir" angelegt und nachdem damit gearbeitet wurde (...) wird es durch ***<delete>*** wieder gelöscht. Die Referenz darauf wird mit der id hergestellt.

Die id wird nicht nur für solche Fälle, in denen Verweise über Attributwerte generiert werden, benutzt, sondern z.B. auch bei Variablenbindungen. Beispiele dazu finden Sie in Teil 4 im Abschnitt über Variablenbindungen.

3. Datentypen

Sie haben im 1. Teil dieses Tutorials bereits Attribute kennengelernt und gesehen, wie man sie initialisiert:

```
<workdir home="./WORK" clear-on-shutdown="true" />
```

Hier wird das Attribute ***home*** für dieses ***<workdir>*** auf ***./WORK*** gesetzt. Für das Attribut ***clear-on-shutdown*** setzen wir ***true***. Offensichtlich erwarten die Attribute unterschiedliche Werte: Einmal eine Pfadangabe und einmal true. Attribute unterscheiden sich also offensichtlich hinsichtlich des Typs von Wert, den sie erwarten.

Jedes Attribut erwartet einen der folgenden Datentypen als Wert:

Datentyp	Wertebereich	Beispiel
INTEGER / LONG	Ganzzahl	125987, -2158
STRING	Zeichenkette (beliebige Zeichen)	"C:/WORK", "Hallo, ich bin ein String!"
BOOLEAN	Wahrheitswert: true (1) oder false (0)	true, 1, false, 0
FETCHEDRESOURCE	Die id eines <fetchresource>, auf dessen Ressource verwiesen werden soll.	—
FILEOBJECT	Die id einer Ressource, die eine Datei / ein Verzeichnis repräsentiert.	—

Diese Datentypen sind nicht nur für Attribute von Bedeutung. Generell können und werden alle Werte in RCML einem dieser Datentypen zugeordnet. Deshalb geben auch Variablenbindungen (die Sie in Teil 4 noch Kennenlernen werden) immer einen Wert von einem dieser Datentypen zurück. Dies ist wichtig, falls Sie z.B. eine Variablenbindungen in einem Attributwert verwenden möchten.

Prinzipiell sind alle Datentyp zu einem String konvertierbar. Das bedeutet, dass Sie einem Attribut, dass einen String erwartet, selbstverständlich auch "3" oder "true" zuweisen können, solange es Sinn macht. Die Konvertierung in die andere Richtung funktioniert natürlich nicht: "Hallo" lässt sich nicht zu einer Zahl konvertieren und "Welt" ist kein Wahrheitswert true oder false.

Wie bereits erwähnt spielen Datentypen vor allem bei Attributen und bei Variablenbindungen eine wichtige Rolle. Der Wert, den Sie einem Attribut zuweisen, muss mit dem erwarteten Datentyp konform sein und eine Variablenbindung gibt immer einen festgelegten Datentyp zurück. Vor allem wenn Sie den Rückgabewert einer Variablenbindung als Attributwert setzen wollen, müssen Sie darauf achten, dass die Variablenbindung auch den Datentyp zurückgibt, der vom Attribut erwartet wird. In der RCML-Elemente-Referenz finden Sie eine Auflistung aller RCML-Elemente mit dazugehöriger Beschreibung. Dort finden Sie auch zu jedem RCML-Element Angaben zu den Attributen und welche Datentypen die Attribute erwarten sowie eine Auflistung der Variablenbindungen und ihrer Rückgabetypen.

4. Einfuehrung in die RCML-Elemente

Nachdem Sie nun mit den wichtigsten Grundlagen der RCML vertraut sind, sollen Ihnen hier nun die wichtigsten Elemente, die in der Worker-RCML existieren, vorgestellt werden.

Wie Sie bereits im 1. Teil dieses Tutorials gelernt haben, besteht ein RCML-Dokument ausschließlich aus Elemente, die wiederum Kindelemente enthalten. Genaugenommen besteht ein RCML-Dokument nur aus einem einzigen Elemente, dem ***<rcml>***-Element. Dieses wiederum darf nur ***<process>*** und ***<script>***-Elemente enthalten. Alle anderen Elemente müssen in einem dieser ***<process>***-Elemente notiert werden. Diese Struktur wird Ihnen aber noch ausführlich in Teil 6 vorgestellt.

In diesem Teil werden nur die wichtigsten RCML-Elemente oberflächlich vorgestellt, damit Sie eine Idee davon bekommen, was mit RCML möglich ist. Dieser Artikel ersetzt nicht die umfangreiche RCML-Elemente-Referenz, in der jedes Element inklusive Beschreibung, Attributen, Variablenbindungen und Beispielen aufgeführt ist.

Momentan existieren folgende RCML-Elemente (inkl. *<rcml>*):

- <process>
    - <delete>
    - <destination>
    - <docxworld-fetch-production-environment>
        - <docxworld-contract>
        - <link-name>
        - <binary-bundles-home>
        - <production-bundles-home>
        - <merge-home>
        - <runtime-home>
    - <eoms-input-query-data>
    - <eoms-input-query-status>
    - <eoms-input-submit>
    - <error>
    - <exec>
        - <param>
        - <commandline>
        - <result>
    - <fetch-production-bundle>
    - <fetch-production-environment>
    - <fetchresource>
    - <if> - <else>
    - <message>
    - <releaseresource>
    - <rw-response>
    - <sleep>
    - <switch>
        - <case>
    - <update-variable>
    - <upload>
    - <while>
    - <workdir>
- <script>

Dies sind die Elemente, mit denen Sie in Ihrem RCML-Dokument arbeiten können. Wie bereits angesprochen ist die Struktur einer RCML-Datei auf den ersten 2. Hierarchieebenen vorgegeben: Das RCML-Dokument besteht aus einem ***<rcml>***-Element, das beliebig viele ***<process>***- und ***<script>***-Elemente beinhaltet. Jedes ***<process>***-Element wiederum darf alle Elemente (auch ***<script>***, aber nicht ***<rcml>***) enthalten (und zwar unbegrenzt viele und alle Elemente unbegrenzt oft). Genaueres erfahren Sie aber noch im 6. Teil. Es ist wichtig, dass Sie diese Struktur verstehen und beachte, wir wollen uns hier aber nun erst einmal nur mit den Elementen und ihren Aufgaben befassen.

In den Beispiel dieses Artikels werden keine gültigen RCML-Dokumente, sondern nur Ausschnitte für diese Elemente gezeigt, die die Funktionsweise verdeutlichen sollen.

<rcml>

Das Wurzelelement jedes RCML-Dokuments. Da das Element selbst keinen Semantische Bedeutung hat, wird es auch nich näher beschrieben. Beachten Sie nur, dass jedes RCML-Dokument (bis auf die XML-Deklaration) nur aus dem <rcml>-Element besteht und Sie Ihr komplettes RCML-Markup **innerhalb** des <rcml>-Elements notieren müssen.

<process>

Wie bereits erwähnt kommt ***<process>*** eine besondere Bedeutung zu: Es bildet sozusagen den "Container" für alle anderen Elemente. Wie der Name schon sagt definiert ***<process>*** einen Prozess, der vom Worker ausgeführt werden kann. Für jeden Job wird genau 1 Prozess ausgeführt: Dazu wird die id des ***<process>*** (das als Attribut gesetzt wird) mit der angeforderten Prozess-ID des Jobs verglichen. Der passende Prozess wird dann für diesen Job ausgeführt (alle anderen ***<process>*** dieser RCML werden ignoriert und für diesen Job nicht ausgeführt). Sie schreiben eine Prozessdefinition also als Inhalt eines ***<process>***-Elements:

```
<process id="MyProcess">
<!-- Hier steht, was getan werden soll, wenn für den Job "MyProcess"
ausgeführt werden soll. -->
</process>
```

<workdir>

Mit ***<workdir>*** können Sie ein Verzeichnis als Arbeitsverzeichnis setzen. Andere Elemente können dieses Arbeitsverzeichnis dann nutzen. Dazu übergeben Sie ***<workdir>*** über das ***home***-Attribut das Verzeichnis und über das ***id***-Attribut einen eindeutigen Namen, über den andere Elemente es verwenden können. Existiert das Verzeichnis noch nicht, wird es angelegt:

```
<workdir id="myworkdir" home="./WORK" />
<!-- Hier wird etwas gemacht... -->
<exec workdir="myworkdir >
...
```

Hier verwendet z.B. ein ***<exec>***-Element die ***<workdir>*** als Arbeitsverzeichnis. Das bedeutet, dieses Element benutzt dann dieses Verzeichnis als Stammverzeichnis.

<delete>

So wie Sie z.B. mit ***<workdir>*** ein Verzeichnis angelegt haben, so können Sie solche Verzeichnisse auch wieder löschen. Verwenden Sie dazu das ***<delete>***-Element:

```
<workdir id="myworkdir" home="./WORK" />
<!-- Hier wird etwas gemacht... -->
<delete file="myworkdir" />
```

<exec>

Wenn Sie Befehle auf Betriebssytemebene (Kommandozeilenbefehle) ausführen wollen, steht Ihnen dazu ***<exec>*** zur Verfügung:

```
<workdir id="myworkdir" home="./WORK"/>
<exec id="ExampleExec" workdir="myworkdir>
<param name="cmd" value="cmd copy C:\testfile.dat C:\new\testfile.dat"
/>
<commandline processor="velocity">$cmd</commandline>
<result return-code="0" />
</exec>
```

Wie Sie sehen, werden innerhalb von ***<exec>*** noch weitere Kindelemente notiert: ***<exec>*** selbst führt noch keinen Befehl aus, sondern ermöglicht nur eine Befehlsausführung. Der eigentliche Befehl wird erst durch ***<commandlnie>*** gesetzt. Über ***<param>***-Elemente können Sie beliebig viele Parameter setzen, die Sie in Ihrem Befehl als normale Variablen (aber über **$**) ansprechen können. Mit ***<result >*** können Sie überprüfen, ob bei der Befehlsausführung Fehler aufgetreten sind und dann die Ausführung stoppen (Ausführung wird gestoppt, wenn result-code nicht 0 ist (0 = kein Fehler). Alle Programmausführungen, also alle Bearbeitungen der Input-Dateien, laufen über ein ***<exec>***.

<fetchresource> und <releaseresource>

<fetchresource> stellt die Verbindung zum Auftrags-System her und dient dazu, Daten (Dateien / Verzeichnisse) vom Auftrags-System (speziell Spooler) zum empfangen und in der RCML verfügbar zu machen. In der Regel empfängt ein Prozess also erst einmal über ***<fetchresource>*** die Daten vom Auftrags-System, die in dem Prozess dann verarbeitet werden sollen. Über ***<releaseresource>*** kann die Ressource wieder freigegeben werden, wenn sie nicht mehr benötigt wird:

```
 <fetchresource id="inputFile"
resource="${process['eoms.process.input']} " />
<!-- Hier wird mit inputFile gearbeitet. -->
<releaseresource file="inputFile" />
<!-- Hier kann nicht mehr mit inputFile gearbeitet werden. -->
```

Wahrscheinlich ist Ihnen "process['eoms.process.input']" und auch das umschließende **${ }** unbekannt. Dabei handelt es sich um ein sogenanntes Feld und einen berechneten Wert. Felder und berechnete Werte werden in Teil 5 vorgestellt.

<upload>

Nachdem Sie Ihre Bearbeitungen erfolgreich abgeschlossen haben, müssen Sie wahrscheinlich die Ergebnissdaten zurück an das Auftrags-System (oder an ein anderes Auftrags-System, an das EOMS-Input, etc.) schicken. Dazu dient ***<upload>***:

```
<workdir id="workdir" home="./WORK" />

<fetchresource id="inputFile"
resource="${process['eoms.process.input']} " />
<!-- Hier wird mit inputFile gearbeitet. Die Resultate werden in der
<workdir> abgelegt. -->
...
<!-- Wenn alle Bearbeitungen abgeschlossen sind, laden wir die
komplette workdir hoch. -->
<upload file="workdir" destination="${process['eoms.process.output']}"
/>
```

<message>

Mit ***<message>*** können Sie eine Nachricht an das EOMS-Core senden. Diese Nachrichten können dort von Auftrags-Systemen oder direkt vom Benutzer ausgelesen werden. Benutzen Sie Messages, um den Ablauf eines Prozesses aufzuzeichnen. Dies erleichtert das Debugging enorm.

```
<message text="Starte Download..." />
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />
<message text="Download abgeschlossen..." />
```

<update-variable>

Mit ***<update-variable>*** können Sie Variablen mit Wert setzen, die dann im gesamten RCML verwendbar sind. Sie können damit sowohl Variablen neu anlegen als auch ändern:

```
<update-variable name="myvar" value="Hallo!" />

<message text="${myvar}" /> <!-- Ausgabe an das EOMS-Core: Hallo! -->

<update-variable name="myvar" value="Hallo, EOMS-Core! />

<message text="${myvar}" /> <!-- Ausgabe an das EOMS-Core: Hallo,
EOMS-Core! -->
```

<if> und <else>

Mit ***<if>*** und ***<else>*** können Sie bedingte Ausführungen formulieren. Sie übergeben ***<if>*** dabei eine Bedingung (Attribut ***condition***). Nur wenn diese Bedingung wahr ist, wird der Inhalt des ***<if>*** ausgeführt, ansonsten nur der Inhalt des ***<else>***. ***<if>*** kann auch alleine ohne ***<else>*** stehen (aber nicht umgekehrt):

```
<if condition="${(1 + 1) == 2}" >
<message text="1 + 1 ist 2!"
</if>
<else>
<message text="Hier lief etwas gewaltig schief!" />
</else>
```

Eine umfangreichere Alternative zu ***<if>*** **-** ***<else>*** bildet das hier nicht behandelte ***<switch>*** **-** ***<case>***.

Weitere Elemente

Die hier vorgestellten Elemente bilden nur einen kleinen Teil des gesamten RCML-Umfangs. Insgesamt gibt es über 30 verschiedene RCML-Elemente. Dazu gehören z.B. noch weitere Standardelemente, Kontrollstrukturen und Elemente zur Interaktion mit dem Redaktions-System und dem EOMS-Input-Server. Sie finden eine vollständige Referenz mit allen existierenden Elementen inklusive Beschreibung, Attributen und Variablenbindungen sowie Beispielen hier. Beachten Sie auch die HowTo-Sektion.

5. RCML fuer Fortgeschrittene

In diesem Artikel soll nun auf die erweiterten Sprachmerkmale von RCML eingegangen werden. Es wird vorausgesetzt, dass Sie bereits mit den Grundlagen von RCML vertraut sind.

1. Skriptsprachen in RCML

Wie bereits in der Einführung in RCML erwähnt ist es möglich, JavaScript innerhalb von RCML einzubetten und zu verwenden. Dies bietet nahezu unbegrenzte Möglichkeiten, da Sie auf alle Sprachfeatures von JavaScript, inklusive Library's, zugreifen können.

Die Einbettung von Skriptcode wird durch das ***<script>***-Element ermöglicht. Der Scriptcode wird dabei einfach als Inhalt notiert:

```
<script language="JavaScript">
<![CDATA[

function getAuthor(){
return "profiforms";
}
]]>
</script>
```

Auf Variablen und Funktionen kann dann in RCML ganz einfach über den Namen zugegriffen werden:

```
<message text="${getAuthor()}" />
```

Es wird dringend empfohlen, den JavaScript-Code innerhalb von **<![CDATA[** und **]]>** zu notieren, um auszuschließen, dass der Parser den Code fälschlicherweise als XML erkennt.

Wichtig ist, dass Sie, wenn sie auf Skriptelemente zugreifen wollen, den Aufruf als sogenannten "berechneten Ausdruck" formulieren müssen, d.h. der Aufruf muss zwischen **${** und **}** stehen. Genauer wird darauf unter 2. Berechnete Werte eingegan gen.

Auf JavaScript als Sprache wird hier nicht weiter eingegangen, da selbst eine Einführung in das Thema zu umfangreich werden würde. Hier finden Sie ein gutes Tutorial.

2. Berechnete Werte

Neben statischen Zuweisungen wie id="1" ist es auch möglich, dynamische Werte, die erst bei Ausführung berechnet werden, zu verwenden. Oft ist dies sogar notwendig, z.B., wenn Sie auf Funktionen und Variablen aus einem ***<script>***-Bereich zugreifen wollen. Der Inhalt des berechneten Werts wird dann nicht als reiner Text behandelt, sondern der Inhalt wird auf Funktions- und Variablenaufrufe überprüft und diese gegebenenfalls ausgeführt und eingesetzt. Um Werte dynamisch berechnen zu lassen, umschließen Sie diese mit **${** und **}**. Der Inhalt von **${ }** wird dann dynamisch berechnet. Beispiel:

```
<script language="JavaScript">
<![CDATA[

function getAuthor(){
return "profiforms";
}
]]>
</script>

<message text="getAuthor()" /> <!-- Messageausgabe: getAuthor()-->
<message text="${getAuthor()}" /> <!-- Messageausgabe: profiforms -->
```

Wie Sie sehen, muss der Ausdruck bei Attributwerten dennoch mit doppelten Anführungszeichen umschlossen werden. Innerhalb des Berechneten Ausdrucks werden alle Verweise aufgelöst und durch die zurückgegebenen Werte ersetzt.

Nicht nur Skriptaufrufe müssen als Berechneter Ausdruck behandelt werden, sondern alle Verweise, die nicht als reiner Text interpretiert werden sollen. Dazu zählt auch der Aufruf von lokalen Variabeln aus dem RCML. Zur Verdeutlichung benutzen wir das Element ***<update -variable>***, das dazu dient, eine Variable (ein Objekt, das wir über seinen Namen ansprechen können und das einen bestimmten Wert hat) anzulegen.

```
<update-variable name="NeueVariable" value="Hallo" /> <!-- Dieses
Element legt eine Variable mit dem Namen "NeueVariable" und dem Wert
"Hallo" an, auf die im gesamten Code zugegriffen werden kann. -->

<message text="NeueVariable" /> <!-- Messageausgabe: NeueVariable -->
<message text="${NeueVariable}" /> <!-- Messageausgabe: Hallo -->
```

Sie sehen also, dass auch auf RCML-Objekte mit **${ }** zugegriffen werden muss, wenn diese nicht als reiner Text interpretiert werden sollen. Dies gilt auch, wenn Sie Elemente über die id ansprechen wollen! Wenn Sie reinen Text innerhalb von **${ }** platzieren wollen, stellen Sie diesen in einfache Anführungszeichen.

Sie können innerhalb von Berechneten Ausdrücken auch Werte verketten, indem Sie den Verkettungsoperator + benutzen:

```
<update-variable name="NeueVariable" value="Hallo" /> <!-- Dieses
Element legt eine Variable mit dem Namen "NeueVariable" und dem Wert
"Hallo" an, auf die im gesamten Code zugegriffen werden kann. -->

<message text="${'Der Wert von NeueVariable ist: ' + NeueVariable +
'.'}" /> <!-- Messageausgabe: Der Wert von NeueVariable ist: Hallo. -->
```

3. Zugriff auf Variablen

In RCML werden Sie mit einer Vielzahl von Variablen arbeiten: In JavaScript erstellte Variablen und Funktionen, mit ***<update-variable>*** g esetzte Variablen, vordefinierte Variablen aus Feldern (siehe 7.) usw. Der Zugriff auf solche Variablen ist denkbar simpel:

```
<update-variable name="test_var" value="1" />
...
<message text="${text_var}" />
```

Alle Objekte (egal ob durch <update-variable> gesetzt oder auf andere Art) werden einfach direkt über ihren Namen angesprochen. Die einzige Besonderheit ist, dass, um RCML zu signalisieren, dass es sich um einen Objektaufruf und nicht um reinen Text handelt, der Name als berechneter Ausdruck gekennzeichnet wird.

4. Sichtbarkeitsbereich in RCML

Von anderen Programmiersprachen sind Sie es eventuell gewohnt, dass es unterschiedliche Sichtbarkeitsbereiche für Variablen und Funktionen gibt. In RCML gibt es nur den globalen Sichtbarkeitsbereich, d.h. Objekte sind immer überall ansprechbar. Sie können also z.B. mit ***<update-variable>*** eine Variable innerhalb eines ***<if>***-Elements anlegen, ohne dass die Sichtbarkeit auf den <if>-Block beschränkt bleibt.

```
<if condition="true" />
<update-variable name="myvar" value="10" />
</if>

<message text="${myvar}" /> <!-- Messageausgabe: 10 -->
```

In klassischen Programmiersprachen wäre "myvar" außerhalb des ***<if></if>***-Elements unbekannt. In RCML können Sie von überall auf "myvar" zugreifen.

Diese Sichtbarkeit gilt auch für ***<script>***-Inhalte. Das bedeutet, Sie können überall im gesamten Dokument auf Skriptelemente zugreifen, egal, wo sie notiert sind.

5. Logische/Boolsche Ausdrücke & Kontrollstrukturen

In vielen Fällen müssen Sie Vergleiche zwischen Werten anstellen oder bestimmen, ob eine Aussage wahr oder unwahr ist (z.B. bei ***<if>***, ***<switch>*** oder ***<while>***). Dazu stehen Ihnen in RCML folgende Operatoren zur Verfügung:

==	Der Ausdruck ist wahr, wenn beide Operanten gleich sind.
!=	Der Ausdruck ist wahr, wenn beide Operanten ungleich sind.
>	Der Ausdruck ist wahr, wenn der linke Operant größer als der Rechte ist.
>=	Der Ausdruck ist wahr, wenn der linke Operant größer oder gleich dem Rechten ist.
<	Der Ausdruck ist wahr, wenn der linke Operant kleiner als der Rechte ist.
<=	Der Ausdruck ist wahr, wenn der linke Operant kleiner oder gleich dem Rechten ist.

Zur Verdeutlichung folgendes Beispiel:

```
<if condition="${ 2 == 3 }"> <!-- Gibt false zurück, also wird ...
nicht ausgeführt -->
...
</if>
<if condition="${ 2 != 3 }"> <!-- Gibt true zurück, also wird ...
ausgeführt. -->
...
</if>
<if condition="${ 3 > 3 }"> <!-- Gibt false zurück, also wird ... nicht
ausgeführt -->
...
</if>
<if condition="${ 3 >= 3 }"> <!-- Gibt true zurück, also wird ...
ausgeführt. -->
...
</if>
<if condition="${ 3 < 3 }"> <!-- Gibt false zurück, also wird ... nicht
ausgeführt -->
...
</if>
<if condition="${ 2 <= 3 }"> <!-- Gibt true zurück, also wird ...
ausgeführt. -->
...
</if>
```

Da es sich bei Vergleichen natürlich auch um Berechnete Werte handelt, müssen diese in ${ } eingeschlossen werden!

Neben boolschen Operatoren (Vergleichsoperatoren) gibt es noch logische Operatoren. Diese dienen zur Verkettung von boolschen Ausdrücken. Beispiel:

```
a == b <!-- Wahr, wenn a gleich b ist. -->
a != b <!-- Wahr, wenn a ungleich b ist. -->
(a == b) && (c != d) <!-- Wahr, wenn a gleich b UND c ungleich d ist.
-->
```

Mit && werden hier also 2 boolsche Ausdrücke verkettet. Der Gesamtausdruck ist nur wahr, wenn beide Einzelausdrücke wahr sind.

Es gibt folgende logische Operatoren in RCML:

&&	Ausdruck ist wahr, wenn beide Einzelausdrücke wahr sind.
||	Ausdruck ist wahr, wenn mindestens einer der Einzelausdrücke wahr ist.
!	Negiert den Ausdruck: Ein wahrer Ausdruck wird unwahr und ein unwahrer Ausdruck wahr.

Folgendes Beispiel zur Verdeutlichung:

```
<update-variable name="a" value="2" />
<update-variable name="b" value="3" />
<update-variable name="c" value="4" />
<update-variable name="d" value="4" />

<if condition="${a < b && c == d}"> <!-- Die Bedingung ist true, da
beide Teilausdrücke wahr sind. -->
...
</if>
<if condition="${a == c || d > b}"> <!-- Die Bedingung ist true, da
einer der Teilausdrücke (der 2.) wahr ist. -->
...
</if>
<if condition="${!(c == d)}"> <!-- Die Bedingung ist false, da zwar der
Ausdruck c == d wahr ist, durch ! aber negiert wird und damit unwahr
wird. -->
...
</if>
```

Eine besondere Anweisung bildet der "bedingte Ausdruck". Dieser folgt folgendem Aufbau:

```
(BEDINGUNG) ? RETURN_A : RETURN_B
```

Dabei ist BEDINGUNG die boolsche Bedingung, die überprüft werden soll, RETURN_A der Wert, der zurückgegeben werden soll, falls BEDINGUNG wahr ist und RETURN_B der Wert, der zurückgegeben werden soll, falls die BEDINGUNG unwahr ist. Dazu folgendes einfaches Beispiel:

```
...

<update-variable name="errorvar" value="false" />

<if condition="${myTestExec.getReturnCode() != 0}" />
<update-variable name="errorvar" value="true" />
</if>

<message text="${errorvar==true ? 'Fehler aufgetreten' : 'Kein Fehler
aufgetreten.'}" />
```

In diesem Beispiel wird die Variable "errorvar" auf "false" gesetzt. Danach wird mit einem ***<if>*** überprüft, ob bei der Ausführung eines ***<exec>*** (der in ... vorkommt) fehlerfrei war. War sie das nicht (ReturnCode ist nicht 0) wird "errorvar" auf true gesetzt. Zum Schluss wird abhängig von errorvar die passende Nachricht an das EOMS-Core geschickt.

 Alle in diesem Abschnitt vorgestellten Operatoren müssen innerhalb von **${ }** notiert sein.

6. Variablenbindungen

Einige Elemente besitzen sogenannte Variablenbindungen, die es ermöglichen, auf Eigenschaften (nicht auf Attribute!) des Elements zuzugreifen und diese abzufragen. Variablenbindungen sind Bindungen an ein RCML-Element, über die man auf Eigenschaften des Elements zugreifen kann. Syntaktisch sind Variablenbindungen Funktionen mit Rückgabewert. Sie folgend folgendem Aufbau:

```
ElementID.Variablenbindung()
```

Es gibt auch Variablenbindungen, die einen Parameter (Argument) erwarten:

```
ElementID.Variablenbindung(DATENTYP Parameter)
```

Die Referenz auf das Element, auf das Sie die Bindung anwenden wollen, wird dabei über die ***id*** des Elements hergestellt. Variablenbindungen geben immer einen Wert zurück, den Sie z.B. als Elementinhalt verwenden oder einem Attribut zuweisen können. Beispiel:

```
<workdir id="myworkdir" home="C:/temp/worker/work />
<message text="${'Workdir-Verzeichnis: ' + workdir.getAbsolutePath()"
/>
<!-- Message, die an EOMS-Core gesendet wird: "Workdir-Verzeichnis:
C:/temp/worker/work" -->
```

Da es sich bei Variablenbindungen natürlich auch um dynamische, zur Laufzeit generierte Werte handelt, müssen auch sie in **${}** eingeschlossen werden.

Es gibt auch Variablenbindungen, die wiederum Objekte zurückgegeben, die selbst wieder Variablenbindungen besitzen. Diese können dann einfach verkettet werden.

Beispiel:

```
<fetchresource id="fetched_resource" resource="${
process['eoms.process.input'] }" />

<!-- Über .getFile() erhalten wir das FILEOBJECT, also ein Objekt, das
die Datei repräsentiert, die durch
<fetchresource> empfangen wurde. FILEOBJECT selbst hat wiederum die
Bindung .getAbsolutePath(), die den Pfad
zur Datei / Verzeichnis zurückgibt. -->
<message text="${ fetchted_resource.getFile().getAbsolutePath() }" />
```

Nur folgende Elemente besitzen Variablenbindungen: *<docxworld-fetch-production-environment>*, *<eoms-input-submit>*, *<eoms-input-query-status>*, *<eoms-input-query-data>*, *<exec>*, *<fetchresource>*, *<rw-response>* und *<workdir>*. Die Elemente haben unterschiedliche Variablenbindungen. Welche Bindungen ein Element besitzt steht in der jeweiligen Elementbeschreibung.

7. Zugriff auf vordefinierte Felder

Sie können in RCML auf vordefinierte Felder zugreifen, die verschiedene Informationen enthalten. Diese Felder werden durch das System gefüllt und erlauben den Zugriff auf die unterschiedlichsten Eigenschaften. Zugegriffen wird ein Feld folgendermaßen:

```
FELDNAME['eigenschaft']
```

wobei *FELDNAME* das Feld ist, auf das Sie zugreifen wollen und *eigenschaft* die Eigenschaft, deren Wert Sie abfragen möchten.

Beispiel:

```
<message text="${ properties['os.name'] }" />
```

Hier greifen Sie auf das Feld **properties** zu und erfragen den Namen des Betriebssystems. Der Rückgabewert der Feldabfrage wäre also z.B. "Windows 7".

Es existieren folgende, vordefinierte Felder, auf die Sie zugreifen können und die unterschiedliche Typen von Informationen enthalten:

Feldname	Enthält
process	Prozessspezifische Eigenschaften, also Eigenschaften, die sich auf den aktuellen Prozess beziehen.
properties	Allgemeine Eigenschaften, die sich auf die Workerumgebung beziehen.

Der Zugriff auf Feldnamen unterscheidet sich in der Anwendung nicht vom Zugriff z.B. auf Variablenbindungen: Da es sich um dynamische, zur Laufzeit berechnete Werte handelt, muss auch hier der Aufruf in **${ }** eingeschlossen werden.

Hier eine (unvollständige) Auflistung der Eigenschaften, die die einzelnen Felder enthalten. Beachten Sie unbedingt, dass nicht immer unbedingt alle Eigenschaften auch mit Werten gefüllt sind, da diese Werte natürlich zuvor z.B. durch das Auftrags-System eingetragen werden müssen! So macht es z.B. keinen Sinn, auf oms.spooler.path_appendix zuzugreifen, wenn gar kein Spooler verwendet wird. Ist eine Eigenschaft nicht gesetzt, wird einfach ein leerer Wert zurückgegeben.

process

Eigenschaft	Zurückgegebener Wert
eoms.client	Referenz auf den Client des Jobs.
eoms.procedure	Die Referenz der Daten in docxworld (docxworld-Vertrag / docxworld-Link).
eoms.process.output	Die Output-Adresse, wohin Ergebnisdaten geschickt werden sollen.
eoms.process.input	Die Input-Adresse, von wo Eingangsdaten abgeholt werden können.
oms.spooler.X	Spooler-Informationen (X steht für die Eigenschaften, z.B. path_appendix).
[selbstdefinierte Eigenschaften]	Sie können beim Aufruf durch ein Auftrags-System eigene Eigenschaften mitgeben, die Sie dann im process-Feld abrufen können (siehe unten).

Das Feld **process** kann im Gegensatz zu **properties** auch eigene Werte enthalten: Sie können diese im Auftrags-System definieren, und sie stehen dann in RCML zur Verfügung. Lesen Sie dazu diesen Artikel.

properties

Das **properties**-Feld enthält eine ganze Reihe statischer Informationen über Betriebssystem und Laufzeitumgebung. Sie können über **properties** einen Großteil der hier aufgelisteten Eigenschaften abfragen.

6. Der Aufbau einer RCML-Datei

Nachdem Sie nun mit den Sprachmerkmalen von RCML vertraut sind wird es Zeit, direkt in den Aufbau einer typischen Worker-RCML zu springen.

RCML-Dokumente sind grundsätzlich sehr einfach aufgebaut. Wie in den vorherigen Kapiteln dieses Tutorials bereits des Öfteren erwähnt enthält ein RCML-Dokument im Prinzip nichts anderes als ein Element (das sogenannte "root"-Element), das wiederum Kindelemente hat, die wiederum eigene Kindelemente haben, usw.

Wie bereits in Teil 1 dieses Tutorials angesprochen wurde ist RCML eine auf XML basierende Auszeichnungssprache. Deshalb enthält ein RCML-Dokument neben dem root-Element eine einzige weitere Anweisung, die sogenannte XML-Deklaration:

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

Mit dieser Deklaration sagen wir dem Interpreter, dass es sich um XML-Syntax handelt. Diese Anweisung muss vor dem root-Element stehen und ist deshalb ganz am Anfang des RCML-Dokuments zu notieren!

Unter der XML-Deklaration folgt das root-Element. Das root-Element hat in RCML den Elementnamen "rcml":

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

Damit haben wir bereits ein gültiges RCML-Dokument. Natürlich fehlt hier noch die Programmlogik. Deshalb kommen jetzt das wichtigste RCML-Elemente ins Spiel: ***<process>.*** Dieses Element definiert einen Prozess für den Worker und **muss** direktes Kindelement von <rcml> sein:

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

Sie können beliebig viele ***<process>***-Elemente im ***<rcml>-Element*** notieren. Beachten Sie aber, dass ***<process>*** in der Hierarchie unbedingt direkt unterhalb von ***<rcml>*** stehen muss. Folgendes ist also nicht erlaubt:

```
<?xml version="1.0" encoding="UTF-8"?>
<rcml>
<exec>
<process>
</process>
</exec>
</rcml>
```

Sie müssen nicht nur darauf achten, dass ***<process>*** direktes Kind von <rcml> ist, sondern auch, dass ***<rcml>*** nur 2 verschiedene Kindelemente erlaubt: Neben ***<process>*** ist das noch ***<script>.*** Folgendes ist also auch erlaubt:

```
<?xml version="1.0" encoding="UTF-8"?>
<rcml>
<process>
<!-- Do something... -->
</process>
<script>
<!-- Hier steht mein Scriptcode. -->
</script>
</rcml>
```

Nicht erlaubt wäre hingegen:

```
<?xml version="1.0" encoding="UTF-8"?>
<rcml>
<process>
<!-- Do something... -->
</process>
<exec>
<!-- Hier wird etwas ausgeführt. -->
</exec>
</rcml>
```

Sie können so viele ***<process>***- und ***<script>***-Elemente wie Sie wollen innerhalb von ***<rcml>*** notieren.

Eine Standard-RCML mit 3 Prozessen sieht also zunächst einmal so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<rcml>
<process id="copy_process">
</process>
<process id="rw_process">
</process>
<process id="eoms_process">
</process>
</rcml>
```

Nun können Sie beginnen, Programmlogik hinzuzufügen. Die Elemente eines Prozesses werden als dessen Kindelemente notiert. In einem ***<process>***-Element dürfen jetzt alle RCML-Elemente eingefügt werden (⚠ abgesehen von ***<rcml>, <process>*).** Eine einfache Implementierung des copy_process könnte z.B. so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<rcml>
<process id="copy_process">
<workdir id="workdir" home="./WORK"></workdir>

<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />

<exec id="ExampleExec" workdir="workdir">
<param name="oldfile" value="${inputFile.getFile().getAbsolutePath()}"
/>-->
<param name="newfile"
value="${workdir+'/copy/'+inputFile.getFile().getName()'+'.copy'}" />
<commandline>cmd /C move $oldfile $newfile</commandline>
</exec>

<upload file="workdir" destination="${process['eoms.process.output']}"
include="${FileName + '.txt'}" />
</process>
<process id="rw_process">
</process>
<process id="eoms_process">
</process>
</rcml>
```

In der Gestaltung der Prozesse sind Sie völlig frei. Um ein Gefühl für den RCML-Aufbau zu bekommen empfehlen wir, dass Sie sich die Standard-RCML-Dateien, die mit dem Worker mitgeliefert werden, anschauen. Die kommentierte Fassung dazu finden Sie hier: OMS und EOMS-Input.

Die Standard-RCMLs

Hier finden Sie die Schritt-für-Schritt Erklärungen der Standard-RCML's, die mit dem EOMS-Worker ausgeliefert werden.

Die Standard-RCML's gliedern sich in folgende Unterkapitel:

RCML fuer den Worker

In diesem Abschnitt wird die RCML für den Worker (worker.rcml) in ihrem Auslieferungszustand behandelt. Die worker.rcml können Sie hier herunterladen.
Die Standard RCML-Prozessdefinition für Worker

Die *worker.rcml* enthält standardmäßig 3 Prozesse (***<process>***). Dies fällt auf, wenn man die Datei in einem XML-Editor bis auf die ***<process>***-Knoten reduziert.

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

+ <process id="rs" name="rs process"> ...</process>
+ <process id="copy" name="Data copy process">...</process>
+ <process id="ReportWriter" name="ReportWriter process">...</process>

</rcml>
```

Der Worker unterstützt also im Auslieferungszustand 3 verschiedene Prozesse:

- **rs**
- **copy**
- **ReportWriter**

In diesem Artikel werden die einzelnen Prozesse nun nacheinander behandelt.

rs

Der Prozess rs implementiert die Anbindung des Workers an das Redaktions-System. Über diesen Prozess ist es möglich, auf Binär- und Produktions-Pakete aus dem Redaktions-System zuzugreifen und diese per ReportWriter mit Daten aus dem Spooler anzureichern, bevor die Daten dann weitergeschickt werden (oder zurück an den Spooler).

```
<process id="rs" name="rs process">
<message text="${'starting worker: java:' +
properties['java.version'] + ' on ' + properties['os.name']}" />

<workdir id="workdir" home="./WORK" clear-on-shutdown="false" />

<script language="JavaScript">
<![CDATA[

var binaryBundleHome = 'binary-bundles';
var productionBundleHome = 'production-bundles';

var workerSystemPath = 'env-home';
```

```
function buildPath4ClientProcedure(aClient, aProcedure) {
var result = workerSystemPath + aClient + '/' + aProcedure;
return result;
}

]]>
</script>

<fetchresource id="inputFile" clear-on-shutdown="false"
resource="${process['eoms.process.input']}" />
<message text="${'Start ReportWriter RSI ' +
inputFile.getFile().getName() + ' ...'} " />
<message text="${'### Working with CLIENT/PROCEDURE: ' +
process['eoms.client'] + '/' + process['eoms.procedure']} " />
<docxworld-fetch-production-environment id="rsProductionEnvironment"
runtime-environment="shared">
<docxworld-contract>${process['eoms.procedure']}</docxworld-contract>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->
<binary-bundles-home>${binaryBundleHome}</binary-bundles-home>
<production-bundles-home>${productionBundleHome}</production-bundles-
home>
<runtime-home>${workdir.getAbsolutePath()}</runtime-home>
<merge-home>${buildPath4ClientProcedure(process['eoms.client'],
process['eoms.procedure'])}</merge-home>
</docxworld-fetch-production-environment>
<switch>
<case condition="${rsProductionEnvironment.getResultCode()=='0'}">
<message text="#docxworld-fetch-result=FETCHED" />
<message text="${'#
ProductionBundleResultCode='+rsProductionEnvironment.getProductionBun
dleResultCode()}" />
<message text="${'#
BinaryBundleResultCode='+rsProductionEnvironment.getBinaryBundleResul
tCode()}" />
<message text="${'#
ProductionBundleID='+rsProductionEnvironment.getProductionBundleID()}
" />
<message text="${'#
SchemaName='+rsProductionEnvironment.getSchemaName()}" />
<message text="${'#
SchemaVersion='+rsProductionEnvironment.getSchemaVersion()}" />
<message text="${'#
BinaryBundleID='+rsProductionEnvironment.getBinaryBundleID()}" />
<message text="${'#
BinaryBundleQuery='+rsProductionEnvironment.getBinaryBundleQuery()}"
/>
<message text="${'#
CommandLine='+rsProductionEnvironment.getCommandLine(inputFile.getFil
e())}" />

<update-variable name="docxworld_fetched_bundle"
value="${rsProductionEnvironment.getProductionBundleID()}" />
<update-variable name="docxworld_environment_dir"
value="${rsProductionEnvironment.getProductionEnvironmentHome()}" />
<exec id="rwProcess" process="rwProcess" workdir="workdir">
<param name="cmd"
```

```
value="${rsProductionEnvironment.getCommandLine(inputFile.getFile())}
" />
<commandline processor="velocity">$cmd</commandline>
</exec>
<switch>
<case condition="${rwProcess.testReturnCode('0')}">
<message text="${'####ReturnCode OK:'+rwProcess.getReturnCode()}" />
<upload file="workdir"
destination="${process['eoms.process.output']}" />
</case>
<case condition="true">
<message text="${'####Fehler ReturnCode:'+rwProcess.getReturnCode()}"
/>
<upload file="workdir"
destination="${process['eoms.process.output']}" />
</case>
</switch>
</case>
<case condition="true">
<message text="#docxworld-fetch-result=ERROR" />
<message text="${'#
ProductionBundleResultCode='+rsProductionEnvironment.getProductionBun
dleResultCode()}" />
<message text="${'#
BinaryBundleResultCode='+rsProductionEnvironment.getBinaryBundleResul
tCode()}" />
<message text="${'#
ProductionBundleID='+rsProductionEnvironment.getProductionBundleID()}
" />
<message text="${'#
SchemaName='+rsProductionEnvironment.getSchemaName()}" />
<message text="${'#
SchemaVersion='+rsProductionEnvironment.getSchemaVersion()}" />
<message text="${'#
BinaryBundleID='+rsProductionEnvironment.getBinaryBundleID()}" />
<message text="${'#
BinaryBundleQuery='+rsProductionEnvironment.getBinaryBundleQuery()}"
/>
<message text="${'#
CommandLine='+rsProductionEnvironment.getCommandLine(inputFile.getFil
e())}" />

<error message="${ ' ReturnCode: ' +
rsProductionEnvironment.getResultCode() + ' ' +
rsProductionEnvironment.getMessage()}" />
</case>
</switch>
<message text="ReportWriter R-S beendet ..." />
<sleep time="1" />
<releaseresource resource="inputFile" />
```

```
<!-- <delete file="workdir" /> -->

</process>
```

Zunächst wird in Zeile 2 eine Message an das EOMS-Core gesendet, um diesem den Beginn des Prozesses mitzuteilen. java.version und os.name sind dabei *Systemvariablen*.

```
<message text="${'starting worker: java:' + properties['java.version']
+ ' on ' + properties['os.name']}" />
```

Danach wird über das *<workdir>*-Element das Arbeitsverzeichnis des Workers festgelegt. Dies sollte immer zu Beginn einer Prozessdefinition geschehen.

```
<workdir id="workdir" home="./WORK" clear-on-shutdown="false" />
```

Durch das *clear-on-shutdown*-Attribut wird festgelegt, dass temporäre Ordner nach Beendigung nicht *(false)* gelöscht werden sollen.

Wichtig ist das dann folgende Skript:

```
<script language="JavaScript">
<![CDATA[

var binaryBundleHome = 'binary-bundles';
var productionBundleHome = 'production-bundles';

var workerSystemPath = 'env-home';

function buildPath4ClientProcedure(aClient, aProcedure) {
var result = workerSystemPath + aClient + '/' + aProcedure;
return result;
}

]]>
</script>
```

Auf die Sprachelemente und den Aufbau von JavaScript wird hier nicht näher eingegangen. Falls Sie noch nicht mit JavaScript vertraut sind finden Sie hier ein Einstiegs-Tutorial.

Zunächst werden hier 3 Variablen angelegt: workerSystemPath soll später als temporäres Stammverzeichnis für die Verarbeitung der R-S Daten dienen *(env-home/)*, während binaryBundleHome und productionBundleHome die Namen der R-S Verzeichnisse enthalten, in denen vom R-S Binär- und Produktionspakete gespeichert werden.

Die Funktion *buildPath4ClientProcedure* dient dazu, einen Pfad für den temporären Ordner, in dem später die Verarbeitung der Daten stattfindet, zu konstruieren *(env-home/<client>/<procedure>)*.

Nun beginnt die eigentliche Arbeit des Workers. Zunächst gilt es, die Jobdaten vom Spooler zu empfangen:

```
<fetchresource id="inputFile" clear-on-shutdown="false"
resource="${process['eoms.process.input']}" />


<message text="${'Start ReportWriter RSI ' +
inputFile.getFile().getName() + ' ...'} " />
<message text="${'### Working with CLIENT/PROCEDURE: ' +
process['eoms.client'] + '/' + process['eoms.procedure']} " />
```

Mit *<fetchresource id="inputFile" clear-on-shutdown="false" resource="${process['eoms.process.input']}" />* wird die vom Spooler empfangene Datei an die id "InputFile" gebunden. Danach werden durch 2 *<message>*'s Nachrichten, die Informationen über die empfangene Datei und den dazugehörigen R-S-Host und -Auftrag enthalten, an das EOMS-Core geschickt.

Neben den Daten aus dem Spooler benötigen wir natürlich auch noch die aus dem R-S (Binärpaket + Produktionspaket):

```
<docxworld-fetch-production-environment id="rsProductionEnvironment"
runtime-environment="shared">
<docxworld-contract>${process['eoms.procedure']}</docxworld-contract>
<binary-bundles-home>${binaryBundleHome}</binary-bundles-home>
<production-bundles-home>${productionBundleHome}</production-bundles-ho
me>
<runtime-home>${workdir.getAbsolutePath()}</runtime-home>
<merge-home>${buildPath4ClientProcedure(process['eoms.client'],
process['eoms.procedure'])}</merge-home>
</docxworld-fetch-production-environment>
```

Alle R-S bezogenen Aktivitäten werden in ein *<docxworld-fetch-production-environment>*-Element eingebettet, das als Container dient. Durch runtime-environment="shared" geben wir an, dass die R-S Binärpakete in einem lokalen Cache gespeichert werden sollen (siehe Elementbeschreibung). Mit *<docxworld-contract>* muss die docxworld-Vertragsnummer zur Zuordnung zwischen Produktions-Paket und docxworld-Vertrag gesetzt werden. Über *<binary-bundle-home>* und *<production-bundle-home>* werden die zuvor im *<script>*-Bereich deklarierten Variablen binaryBundleHome und productionBundleHome als die Verzeichnisse gesetzt, in denen das R-S Binär-Pakete und Produktions-Pakete lokal speichert (damit der Worker weiss, unter welchen Verzeichnissen er auf die Pakete zugreifen kann). Zuletzt setzen wir die Arbeitsverzeichnisse, in denen die Verarbeitung stattfinden soll. Dabei muss unterschieden werden zwischen dem Verzeichnis für runtime-environment="merged" und runtime-environment="shared". Prinzipiell muss nur für das ausgewählte Verfahren das Verzeichnis gesetzt werden. Wir geben hier beides an, damit später lediglich durch Ändern des Attributs das Verfahren von shared zu merged und umgekehrt gewechselt werden kann. *<runtime-home>* gibt hierbei das Verzeichnis für shared an, *<merge-home>* das Verzeichnis für merged. Bei *<merge-home>* müssen wir dafür sorgen, dass für jeden unabhängigen Job ein individuelles Verzeichnis angelegt wird, weshalb wir den Pfad hier aus dem Clientname und dem docxworld-Vertrag durch die oben definierte Funktion buildPath4ClientProcedure konstruieren lassen.

Bevor wir nun mit der Verarbeitung beginnen, müssen wir sicherstellen, dass bei der Festlegung der Parameter innerhalb von *<docxworld-fetch-production-environment>* keine Fehler aufgetreten sind (z.B. Kein Zugriff auf Verzeichnis). Trat bei nur einem der innerhalb von *<docxworld-fetch-production-environment>* notierten Elemente ein Fehler auf wird der Rückgabewert von *<docxworld-fetch-production-environment>* entsprechend gesetzt (siehe auch ReturnCode - HowTo und *<result>*) Mit einem *<switch>* stellen wir nun sicher, dass unser nun folgender Code nur ausgeführt wird, wenn alles in Ordnung ist. Dazu notieren wir sämtlichen Code in einem *<case>* mit der Bedingung, dass der Rückgabewert des *<docxworld-fetch-production-environment>* 0 ist (keine Fehler aufgetreten). Dazu greifen wir per id auf unserer obrigen *<docxworld-fetch-production-environment>* -Container zu und überprüfen den Rückgabewert mit getResultCode():

```
<switch>
<case condition="${rsProductionEnvironment.getResultCode()=='0'}"> <!--
WIRD NUR AUSGEFÜHRT, FALLS KEINE FEHLER AUFGETRETEN SIND. -->
<message text="#docxworld-fetch-result=FETCHED" />
<message text="${'#
ProductionBundleResultCode='+rsProductionEnvironment.getProductionBundl
eResultCode()}" />
<message text="${'#
BinaryBundleResultCode='+rsProductionEnvironment.getBinaryBundleResultC
ode()}" />
<message text="${'#
ProductionBundleID='+rsProductionEnvironment.getProductionBundleID()}"
/>
<message text="${'#
SchemaName='+rsProductionEnvironment.getSchemaName()}" />
<message text="${'#
SchemaVersion='+rsProductionEnvironment.getSchemaVersion()}" />
<message text="${'#
BinaryBundleID='+rsProductionEnvironment.getBinaryBundleID()}" />
<message text="${'#
BinaryBundleQuery='+rsProductionEnvironment.getBinaryBundleQuery()}" />
<message text="${'#
CommandLine='+rsProductionEnvironment.getCommandLine(inputFile.getFile(
))}" />

<update-variable name="docxworld_fetched_bundle"
value="${rsProductionEnvironment.getProductionBundleID()}" />
<update-variable name="docxworld_environment_dir"
value="${rsProductionEnvironment.getProductionEnvironmentHome()}" />

<exec id="rwProcess" process="rwProcess" workdir="workdir">
<param name="cmd"
value="${rsProductionEnvironment.getCommandLine(inputFile.getFile())}"
/>
<commandline processor="velocity">$cmd</commandline>
</exec>
<switch>
<case condition="${rwProcess.testReturnCode('0')}">
<message text="${'####ReturnCode OK:'+rwProcess.getReturnCode()}" />
<upload file="workdir" destination="${process['eoms.process.output']}"
/>
</case>

<case condition="true">
<message text="${'####Fehler ReturnCode:'+rwProcess.getReturnCode()}"
/>
<upload file="workdir" destination="${process['eoms.process.output']}"
/>
</case>
</switch>
</case>
...
...
...
```

Anschließend werden zunächst sämtliche Informationen über die R-S Umgebung und verwendete Pakete an das EOMS-Core gesendet. Diese Informationen werden über das *<docxworld-fetch-production-environment>* -Element abgerufen (Zugriff über id), weshalb es auch so wichtig ist, zu überprüfen, ob dort Fehler aufgetreten sind.

In Zeile 13 und 14 werden die beiden Variablen docxworld_fetcher_bundle und docxworld_environment_dir gesetzt. Diese werden im Code nicht angesprochen, sondern werden für das Auftrags-System gesetzt.

In Zeile 16 beginnt nun endlich die eigentliche Verarbeitung der Daten durch *<exec>*. Diese wird durch *<commandline>* in Zeile 18 gestartet: Ausgeführt werden die in der vom Spooler empfangenen Datei enthaltenen Kommandozeilenbefehle.

Danach wird noch einmal durch eine *<switch>* Abfrage geprüft, ob bei der Verarbeitung Fehler aufgetreten sind und die entsprechende *<message>* an das EOMS-Core gesendet. An das Auftrags-System werden die Ergebnisdaten durch *<upload>* in jedem Fall gesendet.

Damit ist die Arbeit des Workers beinahe getan. Wir müssen allerdings noch festlegen, was passieren soll, falls in *<docxworld-fetch-production-environment>* eben doch ein Fehler aufgetreten ist:

```
<case condition="true"> <!-- WIRD AUSGEFÜHRT, FALLS EIN FEHLER
AUFGETRETEN IST. -->
<message text="#docxworld-fetch-result=ERROR" />
<message text="${'#
ProductionBundleResultCode='+rsProductionEnvironment.getProductionBundl
eResultCode()}" />
<message text="${'#
BinaryBundleResultCode='+rsProductionEnvironment.getBinaryBundleResultC
ode()}" />
<message text="${'#
ProductionBundleID='+rsProductionEnvironment.getProductionBundleID()}"
/>
<message text="${'#
SchemaName='+rsProductionEnvironment.getSchemaName()}" />
<message text="${'#
SchemaVersion='+rsProductionEnvironment.getSchemaVersion()}" />
<message text="${'#
BinaryBundleID='+rsProductionEnvironment.getBinaryBundleID()}" />
<message text="${'#
BinaryBundleQuery='+rsProductionEnvironment.getBinaryBundleQuery()}" />
<message text="${'#
CommandLine='+rsProductionEnvironment.getCommandLine(inputFile.getFile(
))}" />

<error message="${ ' ReturnCode: ' +
rsProductionEnvironment.getResultCode() + ' ' +
rsProductionEnvironment.getMessage()}" />
</case>
</switch>
```

Im Grunde unterscheiden sich die im Fehlerfall gesendeten *<message>*'s kaum von denen, die wir im Erfolgsfall übertragen. Wir melden hier lediglich "docxworld-fetch-result: ERROR", um die Ausnahme zu signalisieren. Natürlich wird im Fehlerfall keine Datenverarbeitung vorgenommen. Zuletzt wird noch die aufgetretene Fehlermeldung übertragen.

Als letzten Schritt geben wir noch die Ressource aus dem Spooler frei:

```
<message text="ReportWriter R-S beendet ..." />
<sleep time="1" />
<releaseresource resource="inputFile" />
```

copy

Der Prozess copy kopiert die Eingabedatei aus dem Auftrags-System an einen zuvor spezifizierten Ort.

```
<process id="copy" name="Data copy process">
<message text="${'starting worker: java:' +
properties['java.version'] + ' on ' + properties['os.name']}"/>
<message text="${'path:' + process['copy.path']}"/>
<workdir id="workdir" home="./WORK" clear-on-shutdown="false" />

<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />

<exec id="copyProcess" workdir="workdir">
<param name="infile"
value="${inputFile.getFile().getAbsolutePath()}"/>
<param name="outfile" value="${process['copy.output']}"/>

<commandline processor="velocity">cmd /C copy $infile
$outfile</commandline>
</exec>


<if condition="${!copyProcess.testReturnCode('1|0')}">
<error message="${'invalid return-code:
'+copyProcess.getReturnCode()}" />
</if>


<message text="uploading files" time="10000"/>
<upload file="workdir"
destination="${process['eoms.process.output']}" includes="*.txt,
std*"/>
<message text="upload complete" />

</process>
```

Zunächst werden in Zeile 2 + 3 *<message>*'s an das EOMS-Core gesendet, um diesem den Beginn des Prozesses mitzuteilen. java.version und os.name sind dabei *Systemvariablen*. Außerdem wird der Pfad, an den die Dateien kopiert werden sollen, angegeben. Auf weitere *<message>*"s im Dokument wird nicht mehr näher eingegangen.

```
<message text="${'starting worker: java:' + properties['java.version']
+ ' on ' + properties['os.name']}"/>
<message text="${'path:' + process['copy.path']}"/>
```

Danach wird über das *<workdir>*-Element das Arbeitsverzeichnis des Workers festgelegt. Dies sollte immer zu Beginn einer Prozessdefinition geschehen.

```
<workdir id="workdir" home="./WORK" clear-on-shutdown="false" />
```

Durch das *clear-on-shutdown*-Attribut wird festgelegt, dass temporäre Ordner nach Beendigung nicht *(false)* gelöscht werden sollen.

Nun wird direkt die Ressource vom Spooler empfangen und an die id "InputFile" gebunden. Den Quellort der Ressource bekommen wir aus "eoms.process.input":

```
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />
```

Dies ist die Ressource (Datei), die kopiert werden soll. Die Zieladresse ist in "copy.output" festgelegt. Im nun folgenden *<exec>* werden zunächst 2 Parameter angelegt, die den Pfad zur Quelldatei "InputFile" sowie den Zielpfad für den Kopiervorgang enthalten. In *<commandline>* wird dann der Kopierprozess durchgeführt.

```
<exec id="copyProcess" workdir="workdir">
<param name="infile" value="${inputFile.getFile().getAbsolutePath()}"/>
<!-- Pfad zur Quelldatei in Parameter speichern. -->
<param name="outfile" value="${process['copy.output']}"/> <!-- Pfad,
wohin kopiert werden soll, in Parameter speichern. -->

<commandline processor="velocity">cmd /C copy $infile
$outfile</commandline> <!-- Hier wird die Datei kopiert. -->
</exec>
```

Nun muss noch überprüft werden, ob der Kopiervorgang erfolgreich war. Dazu verwenden wir eine <*if*>-Abfrage:

```
 <if condition="${!copyProcess.testReturnCode('1|0')}">
<error message="${'invalid return-code: '+copyProcess.getReturnCode()}"
/>
</if>
```

Falls der ReturnCode des <*exec*> nicht 0 oder 1 ist (Der Operator **!** negiert die Aussage, der Ausdruck ist also genau dann wahr, wenn die Aussage falsch ist), wird eine Fehlermeldung an das EOMS-Core gesendet.

Als letzten Schritt müssen die Ergebnisse nur noch an das Auftrags-System zurückgesendet werden:

```
<message text="uploading files" time="10000"/>
<upload file="workdir" destination="${process['eoms.process.output']}"
includes="*.txt, std*"/>
<message text="upload complete" />
```

Hier übergeben wir einfach das komplette Arbeitsverzeichnis an <*upload*>, senden allerdings nur die darin enthaltenen txt-Dateien sowie die Dateien, die mit "std" beginnen (includes).

ReportWriter

Der Prozess ReportWriter ruft, wie der Name schon sagt, den ReportWriter für die Daten des Auftrags-Systems auf und liefert die Ergebnisse zurück.

```
<process id="ReportWriter" name="ReportWriter process">
<message text="${'starting worker: java:' +
properties['java.version'] + ' on ' + properties['os.name']}"/>
<workdir id="workdir" home="./WORK" clear-on-shutdown="false"/>

<switch>
<case condition="${properties['os.name'].substring(0,7) ==
'Windows'}">
<fetchresource id="inputFile" clear-on-shutdown="false"
resource="${process['eoms.process.input']}"/>

<message text="${'Start ReportWriter ' +
inputFile.getFile().getName() + ' ...'} "/>
<exec id="rwProcess" process="rwProcess" workdir="workdir">
<param name="infile"
```

```
value="${inputFile.getFile().getAbsolutePath()}"/>
<param name="workdir" value="${workdir.getAbsolutePath()}"/>
<param name="mandant"
value="${process['oms.spooler.path_appendix']==null ? '' :
process['oms.spooler.path_appendix']}"/>
<commandline processor="velocity">cmd /C
..\..\WORKER\OMS-ReportWriter\reportw $infile
..\..\WORKER\ReportWriter\reportw.tci -alp
..\..\WORKER\ReportWriter\logos -afp ..\..\WORKER\ReportWriter\forms
-aip ..\..\WORKER\ReportWriter\ -ifo
..\..\WORKER\ReportWriter\fonts.ini -awp $workdir -aop $workdir -atp
$workdir -rsp XML -vol -all process.log</commandline>
</exec>
<switch>
<case condition="${rwProcess.testReturnCode('0')}">
<message text="${'####ReturnCode OK:'+rwProcess.getReturnCode()}"/>
<upload file="workdir"
destination="${process['eoms.process.output']}"/>
</case>
<case condition="true">
<error message="${'####Fehler
ReturnCode:'+rwProcess.getReturnCode()}"/>
<upload file="workdir"
destination="${process['eoms.process.output']}"/>
</case>
</switch>
<!-- <result return-code="0"/> -->
<message text="ReportWriter beendet ..."/>
<sleep time="1"/>
<releaseresource resource="inputFile"/>

</case>

<case condition="true">
<error message="${'EOMS-WORKER RCML Definition --&gt; Unsupported
operating system: ' + properties['os.name']}"/>
</case>

</switch>
```

```
<delete file="workdir"/>

</process>
```

Wie gewohnt wird zu Beginn des Prozesses das Arbeitsverzeichnis per *<workdir>* eingerichtet. Entscheidend ist das danach folgende *<switch>*. Da der Aufruf des ReportWriters nur für Windows-Betriebssysteme unterstützt wird, müssen wir hier ausschließen, dass der Worker auf einem nicht unterstützten Betriebssystem arbeitet:

```
<process id="ReportWriter" name="ReportWriter process">
<message text="${'starting worker: java:' + properties['java.version']
+ ' on ' + properties['os.name']}"/>
<workdir id="workdir" home="./WORK" clear-on-shutdown="false"/>

<switch>

<case condition="${properties['os.name'].substring(0,7) == 'Windows'}">
<!-- Betriebssystem ist Windows, also können wir hier weitermachen. -->
...
</case>

<case condition="true">
<error message="${'EOMS-WORKER RCML Definition --&gt; Unsupported
operating system: ' + properties['os.name']}"/>
</case>

</switch>

<delete file="workdir"/>
</process>
```

Wir können nur fortfahren, wenn die Bedingung des 1. *<case>* wahr ist. Wir platzieren also sämtlichen Code für die ReportWriter-Ausführung in diesem *<case>*. Ist das Betriebssystem nicht Windows haben wir dafür das 2. *<case>*. In diesem Fall teilen wir dem EOMS-Core mit, dass wir nicht fortfahren können. Unterhalb des *<switch>* passiert nurnoch eine kleine Sache: Wir löschen das *<workdir>* manuell wieder. Der komplette wichtige Code steht also im 1. *<case>*. Diesen wollen wir uns jetzt näher anschauen.

Zunächst brauchen wir natürlich die Ressource vom Auftrags-System (Spooler):

```
<fetchresource id="inputFile" clear-on-shutdown="false"
resource="${process['eoms.process.input']}"/>
```

Im nächsten Schritt soll diese nun durch den ReportWriter verarbeitet werden. Dazu brauchen wir einen *<exec>*-Container, in dem wir auch gleich folgende Parameter setzen:

```
<exec id="rwProcess" process="rwProcess" workdir="workdir">
<param name="infile" value="${inputFile.getFile().getAbsolutePath()}"/>
<param name="workdir" value="${workdir.getAbsolutePath()}"/>
<param name="mandant"
value="${process['oms.spooler.path_appendix']==null ? '' :
process['oms.spooler.path_appendix']}"/>
```

"infile" ist wie gewohnt der Systempfad zum InputFile aus dem Auftrags-System, "workdir" der Systempfad zur zuvor gesetzten *<workdir>*, und "mandant" ist der Spooler, von dem die Ressource empfangen wurde.

Der wichtigste Teil folgt jetzt: Die Ausführung des ReportWriters auf das inputFile.

```
<commandline processor="velocity">cmd /C
..\..\WORKER\ReportWriter\reportw $infile
..\..\WORKER\ReportWriter\reportw.tci -alp
..\..\WORKER\ReportWriter\logos -afp ..\..\WORKER\ReportWriter\forms
-aip ..\..\WORKER\ReportWriter\ -ifo
..\..\WORKER\ReportWriter\fonts.ini -awp $workdir -aop $workdir -atp
$workdir -rsp XML -vol -all process.log</commandline>
```

Wir werden auf diesen doch sehr umfangreichen Befehlsstring nicht im Detail eingehen. Das wichtigste ist: Dem ReportWriter wird beim Aufruf das $infile übergeben und die Arbeits- und Ausgabeverzeichnisse auf das zu Beginn gesetzt $workdir gesetzt. Als Logfile setzen wir process.log. Der ReportWriter verarbeitet also $infile und liefer das Ergebnis direkt in $workdir. Von dort können wir, falls alles fehlerfrei ablief, diese Ergebnisse dann auch wieder hochladen. Genau das überprüfen wir im nachfolgenden *<switch>*:

```
 <switch>
<case condition="${rwProcess.testReturnCode('0')}">
<message text="${'####ReturnCode OK:'+rwProcess.getReturnCode()}"/>
<upload file="workdir"
destination="${process['eoms.process.output']}"/>
</case>
<case condition="true">
<error message="${'####Fehler
ReturnCode:'+rwProcess.getReturnCode()}"/>
<upload file="workdir"
destination="${process['eoms.process.output']}"/>
</case>
</switch>
```

Wir überprüfen, ob ein Fehler bei der Verarbeitung durch den ReportWriter aufgetreten ist und senden die entsprechende *<message>*. Hochgeladen wird die Ressource in beiden Fällen. Für den Produktiveinsatz ist es unter Umständen Sinnvoll, im Fehlerfall kein *<upload>* vorzunehmen.

Downloads

- worker.rcml

EOMS-Input RCML

In diesem Abschnitt wird die RCML für den EOMS-Input-Worker (eoms-input.rcml) in ihrem Auslieferungszustand behandelt. Die eoms-input.rcml können Sie hier herunterladen.

Die Standard RCML-Prozessdefinition für EOMS-Input-Worker

Die *eoms-input.rcml* enthält standardmäßig 2 Prozesse (***<process>***). Dies fällt auf, wenn man die Datei in einem XML-Editor bis auf die ***<process>***-Knoten reduziert.

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

+ <process id="eoms-input" name="eoms-prep-input process">...</process>
+ <process id="eoms-input-submitonly" name="eoms-prep-input-submitonly
process">...</process>

</rcml>
```

Der EOMS-Input-Worker unterstützt also im Auslieferungszustand 2 verschiedene Prozesse:

- **eoms-input**
- **eoms-input-submitonly**

In diesem Artikel werden die einzelnen Prozesse nun nacheinander behandelt.

eoms-input

Der Prozess eoms-input empfängt eine Ressource vom Auftrags-System (Spooler) und überträgt sie an einen EOMS-Input-Server (docxworld).

```
<process id="eoms-input" name="eoms-prep-input process">

<message text="${'starting worker: java:' +
properties['java.version'] + ' on ' + properties['os.name'] + ' for
customer ' + process['eoms.client']}"/>

<script language="JavaScript">
<![CDATA[
var submitVariables = new java.util.HashMap();
submitVariables.put('EOMS_CLIENT', process['eoms.client']);
submitVariables.put('EOMS_PROCEDURE', process['eoms.procedure']);
submitVariables.put('EOMS_REFERENCE', process['eoms.reference']);
]]>
</script>

<workdir id="workdir" home="./WORK"/>

<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />

<eoms-input-submit id="submitJob" file="inputFile"
variables="${submitVariables}"/>
<eoms-input-query-status id="queryJobStatus" job="submitJob"
interval="5" />
<eoms-input-query-data id="queryJobData" job="submitJob"
workdir="workdir" />

<upload file="workdir"
destination="${process['eoms.process.output']}" mask="*.*" />

</process>
```

In Zeile 3 informieren wir zuerst einmal das EOMS-Core über den Start des Prozesses.

```
 <message text="${'starting worker: java:' + properties['java.version']
+ ' on ' + properties['os.name'] + ' for customer ' +
process['eoms.client']}"/>
```

Bei der späteren Übertragung wollen wir ein Array mit Informationen über das docxworld-Verfahren mit übergeben. Dazu legen wir in JavaScript eine HashMap mit Namen *"submitVariables"* an:

```
<script language="JavaScript">
<![CDATA[
var submitVariables = new java.util.HashMap();
submitVariables.put('EOMS_CLIENT', process['eoms.client']);
submitVariables.put('EOMS_PROCEDURE', process['eoms.procedure']);
submitVariables.put('EOMS_REFERENCE', process['eoms.reference']);
]]>
</script>
```

Da wir noch kein Arbeitsverzeichnis angelegt haben, tun wir das jetzt:

```
<workdir id="workdir" home="./WORK"/>
```

Bevor wir etwas zum EOMS-Input übertragen können, müssen wir natürlich erst einmal die Ressource, die weitergeschickt werden soll, empfangen:

```
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />
```

Nun können wir mit der eigentlichen Arbeit, der Übertragung an den EOMS-Input-Server, beginnen. Dazu verwenden wir *<eoms-input-submit>*, das unser "inputFile" an den in der ***eoms.invoker.client.properties*** konfigurierten EOMS-Input-Server sendet. Außerdem übergeben wir hier auch unsere zuvor erstellte HashMap, die mitgesendet wird.

```
<eoms-input-submit id="submitJob" file="inputFile"
variables="${submitVariables}"/>
```

Mehr müssen wir theoretisch jetzt nicht mehr tun (genau hier hört auch ***eoms-input-submitonly*** auf). Da wir aber nicht sicher sein können, ob die Daten tatsächlich angekommen sind, fragen wir in regelmäßigen Abständen so lange den Status der Sendung ab, bis wir eine erfolgreiche Rückmeldung bekommen. Als Referenz geben wir die ***id*** des *<eoms-input-submit>* (also die id der Übertragung) als Attribut ***job*** an. Dadurch weiß das Element, dass es für diese Übertragung eine Statusanfrage tätigen soll:

```
<eoms-input-query-status id="queryJobStatus" job="submitJob"
interval="5" />
```

Die Ausführung wird erst fortgesetzt, wenn eine positive Reaktion vom EOMS-Input-Server eintrifft. In diesem Fall holen wir uns dann auch vom EOMS-Input-Server die Ergebnisse zurück. Als Referenz dient hierbei wieder die ***id*** des *<eoms-input-submit>*. Wir speichern die Ressource in unserer zu Beginn angelegten *<workdir>*.

```
<eoms-input-query-data id="queryJobData" job="submitJob"
workdir="workdir" />
```

Zu guter Letzt senden wir die Ergebnisdaten zurück an das Auftrags-System (Spooler). Wir senden dazu die komplette *<workdir> (in der ja sowieso nur unsere EOMS-Input-Server-Ergebnisressourcen liegen)*.

```
<upload file="workdir" destination="${process['eoms.process.output']}"
mask="*.*" />
```

eoms-input-submitonly

Der Prozess eoms-input-submitonly ist eine reduzierte Variante des eoms-input-Prozesses. Für Zeitkritische Operationen können Sie diesen Prozess statt eoms-input verwenden. Beachten Sie aber, dass im Gegensatz zu eoms-input hier keine Prüfung vorgenommen wird, ob die Daten erfolgreich vom EOMS-Input-Server angenommen / verarbeitet wurden.

```
<process id="eoms-input-submitonly" name="eoms-prep-input-submitonly
process">

<message text="${'starting worker: java:' + properties['java.version']
+ ' on ' + properties['os.name']}"/>

<script language="JavaScript">
<![CDATA[
var submitVariables = new java.util.HashMap();
]]>
</script>

<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />
<eoms-input-submit id="submitJob" file="inputFile"
variables="${submitVariables}"/>

</process>
```

Der Aufbau gleicht bis auf 2 Unterschiede vollständig dem von **eoms-input**:

1. Es wird nur eine leere HashMap bei der Übertragung übermittelt (der EOMS-Input-Server erhält hier keinerlei Informationen),
2. Es wird auf die "saubere" Übertragung verzichtet, das bedeutet es wird nur *<eoms-input-submit>* aufgerufen und nicht mit *<eoms-input-query-status>* überprüft, ob Probleme aufgetreten sind. Es erfolgt außerdem keine Rückübermittlung der Daten per *<eoms-input-query-data>*. Dementsprechend werden auch keine Daten an das Auftrags-System zurückgegeben.

Downloads

- eoms-input.rcml

How-To's

Die How-To Sektion enthält Schritt-für-Schritt Anleitung, Codebeispiele, oft benötigte Code-Snippets und interessante Anwendungsszenarien. Wir bemühen uns, die How-To's ständig zu erweitern.

Momentan sind folgende How-To Artikel verfügbar:

Anwendung von Kontrollstrukturen

Zeigt die Anwendung der Kontrollstrukturen if - else, switch - case und while.
Einlieferung in docxworld

Beschreibt, wie Daten zum EOMS-Input-Server von docxworld übertragen werden können.
Interaktion mit dem Redaktions-System

Hier wird gezeigt, wie Sie das EOMS an ein Redaktions-System anbinden können, sodass Pakete aus dem R-S für die Jobverarbeitung verfügbar werden.
JavaScript in RCML

Verdeutlicht anhand simpler Beispiele die Einbettung von JavaScript in RCML.
Umgang mit Return-Codes

Beschreibt die Abfrage, Behandlung und das Setzen von Rückgabewerten (Return-Codes).
Variablenaustausch mit Auftrags-Systemen

Beschreibt, wie in RCML-Code Variablen(-Werte) zwischen dem EOMS und OMS-Auftrags-Systemen ausgetauscht werden können.

Anwendung von Kontrollstrukturen

In diesem How-To wollen wir uns etwas näher mit den Kontrollstrukturen in RCML befassen. Kontrollstrukturen ermöglichen es, dynamische Verzweigungen im Code einzufügen. Das bedeutet, durch Kontrollstrukturen können Sie auf Veränderungen in der Ausführung reagieren. Prinzipiell benötigen Kontrollstrukturen immer eine Bedingung, die auf ihren Wahrheitswert (boolsche Bedingung) geprüft wird. Ist die Bedingung wahr (z.B. 1 == 1) wird von der Kontrollstruktur anders reagiert, als wie wenn sie unwahr ist (1 == 2). Diese Bedingung wird bei allen Kontrollstrukturen über das Attribut ***condition*** übergeben.

In RCML gibt es vier grundlegende Kontrollstrukturen:

- **<if>** und **<else>**
    - Der Grund, weshalb wir ***<if>*** und ***<else>*** zusammenfassen ist, dass ***<else>*** nie ohne ***<if>*** stehen darf, sich immer auf ein und deshalb kein eigenständiges Element ist.
- **<switch>** und **<case>**
- **<while>**

In diesem How-To werden die Kontrollstrukturen nun nacheinander vorgestellt. Detailliertere Informationen über die Elemente finden Sie wie immer in der Elementreferenz.

1. if und else

<if> und ***<else>*** bilden die einfachste Form der bedingten Verzweigung. ***<if>*** wird dabei eine Bedingung über das Attribut ***condition*** über geben, die überprüft wird. Nur wenn die Bedingung wahr ist, wird der Inhalt von ***<if>*** ausgeführt:

```
<update-variable name="year" value="2014" />
<if condition="${year == 2014}" />
<message text="Wir schreiben das Jahr 2014." />
</if>
```

Das ***<else>*** ist optional und kann nach einem ***<if>*** notiert werden. Der Sinn: Ist die Bedingung von ***<if>*** unwahr, wird der Inhalt des danach folgenden ***<else>*** ausgeführt:

```
<update-variable name="year" value="2014" />
<if condition="${year == 2014}" />
<message text="Wir schreiben das Jahr 2014." />
</if>
<else>
<message text="${'Es ist doch nicht das Jahr 2014, sondern' + year}" />
</else>
```

Die 1. Message wird also nur ausgeführt, wenn year 2014 ist und die 2. Message in absolut allen anderen Fällen. Steht ein ***<if>*** alleine ohne ***<else>*** und die Bedingung des if ist unwahr, wird die Verarbeitung fortgesetzt, als würde überhaupt kein ***<if>*** stehen.

<if>-Abfragen eignen sich besonders gut, wenn nur eine Bedingung überprüft werden muss:

```
<exec id="ExampleExec" workdir="workdir">
...
</exec>

<if condition="${ExampleExec.getReturnCode() != 0}" />
<error message="Fehler aufgetreten!" />
</if>
```

Nur falls bei der Ausführung des ***<exec>*** ein Fehler auftritt, wird durch einen ***<error>*** die Verarbeitung abgebrochen. War die Ausführung im ***<exec>*** hingegen erfolgreich, passiert garnichts.

Eben dieser Vorteil eines ***<if>***, schnell und ohne wenig Aufwand die Ausführung von Code von einer Bedingung abhängig zu machen, bereitet bei größeren Abfragen mit mehreren Bedingungen Probleme. Angenommen, Sie wollen nicht nur abfragen, ob ein Fehler aufgetreten ist, sondern auch, welcher:

```
<exec id="ExampleExec" workdir="workdir">
...
</exec>

<if condition="${ExampleExec.getReturnCode() == 0}" />
<error message="Kein Fehler aufgetreten!" />
</if>

<if condition="${ExampleExec.getReturnCode() == 1}" />
<error message="Fehler 1 aufgetreten." />
</if>

<if condition="${ExampleExec.getReturnCode() == 500}" />
<error message="Fehler 500 aufgetreten!" />
</if>
```

Da ein ***<if>***-Statement immer nur eine Bedingung überprüfen kann, müssen Sie für jede Bedingung ein neues ***<if>*** konstruieren. Dies kann sehr schnell unübersichtlich werden. Außerdem ist es nicht möglich, ein ***<else>*** auf mehrere ***<if>*** zu beziehen: Angenommen, Sie möchten im oberen Beispiel noch ein ***<else>***, dass sich um alle anderen Errorcode kümmert, implementieren. Wenn Sie das ***<else>*** einfa ch unterhalb der ***<if>***-Elemente notieren würden, würde es sich nur auf das letzte ***<if>*** beziehen. Der Interpreter würde also folgende Prüfreihenfolge abarbeiten:

1. Ist der ReturnCode == 0 ? Falls ja, führe Inhalt von 1. ***<if>*** aus.
2. Ist der ReturnCode == 1 ? Falls ja, führe Inhalt von 2. ***<if>*** aus (auch, falls 1.***<if>*** schon ausgeführt wurde).
3. Ist der ReturnCode == 500 ? Falls ja, führe Inhalt von 3. ***<if>*** aus (auch, falls vorher schon ***<if>***'s ausgeführt wurden). Falls Nein, führe Inhalt von ***<else>*** aus.

Der Inhalt von ***<else>*** wird nur dann nicht ausgeführt, wenn ReturnCode == 500. Ist der ReturnCode also gleich 1, wird das 2. ***<if>*** und das ***<else>*** ausgeführt. Das ist nicht das gewollte Verhalten.

Eine Umgehung dieses Problems könnte so aussehen:

```
<exec id="ExampleExec" workdir="workdir">
...
</exec>

<if condition="${ExampleExec.getReturnCode() != 0}" />
<if condition="${ExampleExec.getReturnCode() == 1}" />
<error message="" />
</if>

<if condition="${ExampleExec.getReturnCode() == 500}" />
<error message="Fehler 500 aufgetreten!" />
</if>

<if condition="${ExampleExec.getReturnCode() != 1 &&
ExampleExec.getReturnCode() != 500}" />
<error message="Unbekannter Fehler aufgetreten!" />
</if>
</if>
<else>
<error message="Kein Fehler aufgetreten!" />
</else>
```

Sie sehen, dass diese Vorgehensweise für viele Abfragen kaum übersichtlich zu halten ist (dieses Beispiel ist nur eine mögliche Variante). Für solche Fälle eignet sich deshalb ein ***<switch>*** wesentlich besser.

Wichtig ist auch: ***<if>***-Statements sind völlig unabhängig voneinander. Das bedeutet, wenn Sie mehrere ***<if>*** hintereinander notieren, wird trotzdem jedes ***<if>*** einzeln geprüft und gegebenenfalls ausgeführt. Es gibt also durch ***<if>*** so einfach keine Möglichkeit, mehrere Bedingungen zu formulieren, von denen nur eine ausgeführt werden soll. Auch dafür gibt es das ***<switch>***-Element:

2. switch und case

<switch>* - *<case>- Konstrukte ähneln ***<if>* - *<else>***-Verzweigungen. Im Gegensatz zu diesen weden bei einem ***<switch>*** aber mehrere Bedingungen formuliert, von denen immer nur eine ausgeführt wird. Die Bedingungen bei einem ***<switch>*** sind also im Gegensatz zu ***<if>*** nicht voneinander unabhängig. Außerdem erlaubt es ***<switch>***, ein Defaultverhalten zu implementieren, dass ausgeführt wird, wenn keine Bedingung zutrifft (was mit ***<if>*** nur umständlich oder garnicht möglich ist, wie wir gesehen haben).

Ein ***<switch>*** enthält beliebig viele ***<case>***-Elemente (mindestens jedoch 1). Jedes ***<case>*** stellt eine Bedingung dar (wie ein ***<if>***). Die Bedingung wird, wie bei ***<if>***, über das Attribut ***condition*** gesetzt. Das Beispiel von oben mit ***<switch>*-*<case>*** statt ***<if>***:

```
<exec id="ExampleExec" workdir="workdir">
...
</exec>

<switch>
<case condition="${ExampleExec.getReturnCode() == 0}" />
<error message="Kein Fehler aufgetreten!" />
</case>

<case condition="${ExampleExec.getReturnCode() == 1}" />
<error message="Fehler 1 aufgetreten." />
</case>

<case condition="${ExampleExec.getReturnCode() == 500}" />
<error message="Fehler 500 aufgetreten!" />
</case>
</switch>
```

Syntaktisch hat sich nicht viel getan - "if" wurde einfach durch "case" ersetzt und alle ***<case>*** von einem ***<switch>*** umschlossen. Semantisch ist der Unterschied schon größer: Wie bereits erwähnt wird immer nur 1 ***<case>*** eines ***<switch>*** ausgeführt. Im oberen Beispiel schließen sich die Bedingungen sowieso aus - in folgendem Beispiel wollen wir dies daher im Vergleich zu ***<if>*** verdeutlichen:

```
<update-variable name="teilnehmerzahl" value="15" />

<switch>
<case condition="${teilnehmerzahl < 10}" />
<error message="Kleine Gruppe!" />
</case>

<case condition="${teilnehmerzahl < 25}" />
<error message="Mittelgroße Gruppe!" />
</case>

<case condition="${teilnehmerzahl < 50}" />
<error message="Große Gruppe!" />
</case>
</switch>
```

Obwohl sowohl die Bedingung des 2., als auch des 3. ***<case>*** wahr ist, wird hier nur das 2. ***<case>*** ausgeführt. Die Ausgabe lautet also: "Mittelgroße Gruppe!". Grund ist, dass in einem ***<switch>*** immer die erste gültige ***<case>*** ausgeführt wird. Alle danach stehenden ***<case>*** werden nicht mehr geprüft, also auch dann nicht ausgeführt, wenn ihre Bedingung wahr ist!

Eine weitere Eigenschaften von ***<switch>*** ist, dass **immer** ein ***<case>*** ausgeführt wird, auch dann, wenn es kein ***<case>*** gibt, dessen Bedingung wahr ist:

```
<update-variable name="teilnehmerzahl" value="100" />


<switch>
<case condition="${teilnehmerzahl < 10}" />
<error message="Kleine Gruppe!" />
</case>

<case condition="${teilnehmerzahl < 25}" />
<error message="Mittelgroße Gruppe!" />
</case>

<case condition="${teilnehmerzahl < 50}" />
<error message="Große Gruppe!" />
</case>
</switch>
```

In diesem Beispiel trifft keine der Bedingungen zu. In solch einem Fall wird automatisch das **letzte** ***<case>*** ausgeführt. Die Ausgabe hier ist also: "Große Gruppe!".

Diese Eigenschaften von ***<switch>*** (Nur das erste gültige ***<case>*** wird ausgeführt + es wird immer 1 ***<case>*** ausgeführt) ermöglicht es, ein Standardverhalten zu implementieren, wie wir es bereits bei ***<if>*** vermisst haben:

```
<exec id="ExampleExec" workdir="workdir">
... <!-- Wirft ReturnCode = 400 -->
</exec>

<switch>
<case condition="${ExampleExec.getReturnCode() == 0}" />
<error message="Kein Fehler aufgetreten!" />
</case>

<case condition="${ExampleExec.getReturnCode() == 1}" />
<error message="Fehler 1 aufgetreten." />
</case>

<case condition="${ExampleExec.getReturnCode() == 500}" />
<error message="Fehler 500 aufgetreten!" />
</case>

<case condition="true" />
<error message="Ein unbekannter Fehler ist aufgetreten!" />
</case>
</switch>
```

Hier wird nun, falls der ReturnCode weder 0 noch 1 noch 500 ist, das letzte ***<case>*** ausgeführt: "Ein unbekannter Fehler ist aufgetreten!" wäre hier die Ausgabe. Wir könnten hier auch die Bedingung beim letzten ***<case>*** auf *false* setzen, das Verhalten wäre dasselbe: Da kein vorheriges ***<case>*** eine wahre Bedingung enthält, wird das letzte ***<case>*** ausgeführt, unabhängig davon, ob dessen Bedingung wahr ist.

3. while

Bisher haben wir nur Kontrollstrukturen gesehen, die Inhalt abhängig von einer Bedingung ausführen (oder eben nicht ausführen). Eine andere Art von Kontrollstruktur stellt ***<while>*** dar: Mit ***<while>*** ist es möglich, Inhalt abhängig von einer Bedingung mehrfach ausführen zu lassen *("Schleife")*.

Auch ***<while>*** wird die Bedingung über das Attribut ***condition*** übergeben. Zu Beginn wird geprüft, ob die Bedingung wahr ist und falls ja, der Inhalt von ***<while>*** ausgeführt. Nach Ausführung des Inhalts wird die Bedingung erneut geprüft und, falls sie immer noch wahr ist, die Schleife erneut ausgeführt. Erst wenn die Bedingung unwahr ist, wird die Schleife verlassen.

```
<update-variable name="var" value="1" />

<message text="Beginn Schleife" />
<while condition="${var <= 3}" >
<message text="${ 'Wert von var:' + var }" />
<update-variable name="var" value="${var + 1}" />
</while>
<message text="Ende Schleife" />
```

Die Ausgabe dieser Schleife:

```
Beginn Schleife
Wert von var: 1
Wert von var: 2
Wert von var: 3
Ende Schleife
```

Sie müssen unbedingt darauf achten, innerhalb der Schleife ein Verhalten zu implementieren, das dafür sorgt, dass die Bedingung nach einer endlichen Zahl von Ausführungen unwahr wird, da Sie sonst eine Endlosschleife generieren, die zum Absturz des Workers führt.

Sie können Schleifen auch Schachteln. Folgendes Beispiel zur Verdeutlichung:

```
<update-variable name="jahr" value="2013" />
<update-variable name="quartal" value="1" />

<message text="Beginn Schleife" />
<while condition="${Jahr <= 2014}" >
<message text="${ 'Jahr:' + jahr }" />

<while condition="${quartal <= 4}" >
<message text="${ 'Quartal:' + quartal }" />
<update-variable name="var" value="${var + 1}" />
</while>
<update-variable name="quartal" value="1" />
<update-variable name="jahr" value="${jahr + 1}" />
</while>
<message text="Ende Schleife" />
```

Ausgabe:

```
Beginn Schleife
Jahr:2013
Quartal:1

Jahr:2013
Quartal:2

Jahr:2013
Quartal:3

Jahr:2013
Quartal:4

Jahr:2014
Quartal:1

Jahr:2014
Quartal:2

Jahr:2014
Quartal:3

Jahr:2014
Quartal:4
Ende Schleife
```

Schleifen lassen sich beliebig tief Verschachteln - Achten Sie nur darauf, dass die Ausführungszeit dabei exponentiell steigt.

Einlieferung in docxworld

Daten an die EOMS-Input-Schnittstelle von docxworld zu schicken ist mit Hilfe des *<eoms-input-submit>*-Elements denkbar einfach. Im Folgenden soll die Standardvorgehensweise dazu aufgezeigt werden.

Um korrekt mit dem EOMS-Input-Server zu interagieren, sind folgende Schritte nötig:

1. Empfang der Ressource, die eingeliefert werden soll, aus dem Auftrags-System (Spooler),
2. Der eigentliche Versendevorgang,
3. Nachprüfung, ob Probleme aufgetreten sind,
4. Eventuell der Empfang von Ergebnisdaten und die Rücksendung dieser an das (oder ein) Auftrags-System.

1. Der Empfang der Ressource aus dem Auftrags-System

Wie immer verwenden wir dazu das <fetchresource>-Element:

```
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />
```

Unsere Ressource ist jetzt über "inputFile" erreichbar.

2. Der Sendevorgang

Bevor wir unsere Ressource abschicken, kontrollieren wir, ob in der eoms.invoker.client.properties die richtigen Verbindungsdaten eingetragen sind, da sich alle RCML-Elemente, die docxworld erreichen müssen, diese Verbindungsdaten verwenden. Danach können wir in RCML fortfahren. Zunächst legen wir in JavaScript eine HashMap an, die wir mit allen nötigen Informationen über uns (über den Client) füllen:

```
<script language="JavaScript">
<![CDATA[
var submitVariables = new java.util.HashMap();
submitVariables.put('EOMS_CLIENT', process['eoms.client']);
submitVariables.put('EOMS_PROCEDURE', process['eoms.procedure']);
submitVariables.put('EOMS_REFERENCE', process['eoms.reference']);
]]>
</script>
```

Danach können wir schon unsere Ressource mitsamt der HashMap versenden:

```
<eoms-input-submit id="submitJob" file="inputFile"
variables="${submitVariables}"/>
```

Wichtig ist die ***id*** des *<eoms-input-submit>*-Element's, da wir uns noch darauf beziehen werden. Über die id ist dieser "Einlieferungs-Auftrag" referenzierbar.

3. Nachprüfung

<eoms-input-submit> sendet zwar die Daten an den EOMS-Input-Server, überprüft nicht, ob die Daten auch erfolgreich in Empfang genommen / bearbeitet wurden. Verwenden Sie dazu *<eoms-input-query-status>*, das die Weiterverarbeitung so lange blockiert, bis es eine positive Rückmeldung vom EOMS-Input-Server erhalten hat, dass die Daten angekommen und der Job beendet ist:

```
<eoms-input-query-status id="queryJobStatus" job="submitJob"
interval="5" />
```

Setzen Sie das Attribut ***job*** auf die ***id*** des *<eoms-input-submit>*, für das Sie die Überprüfung machen wollen.

4. Empfang und Weitergabe von Ergebnisdaten

Wenn Sie Daten wieder vom EOMS-Input-Server zurückempfangen möchten, können Sie dazu *<eoms-input-query-data>* verwenden. Dann sollten Sie aber unbedingt vorher per *<eoms-input-query-status>* sicherstellen, dass nur fortgefahren wird, falls mit dem Job alles ok ist.

```
<workdir id="workdir" home="./WORK" />
<eoms-input-query-data id="queryJobData" job="submitJob"
workdir="workdir" />
```

Wieder müssen Sie über das Attribut ***job*** die ***id*** des *<eoms-input-submit>* angeben, für das die Abfrage gelten soll. Die Daten werden in dem angegebenen Arbeitsverzeichnis abgelegt. Um Sie dann an das Auftrags-System zurückzusenden (oder an ein anderes weiterzusenden), verwenden Sie wie gewohnt *<upload>* und senden Sie einfach das komplette Arbeitsverzeichnis:

```
<upload file="workdir" destination="${process['eoms.process.output']}"
/>
```

Interaktion mit dem Redaktions-System

Die Anbindung an das Redaktions-System erfolgt über das *<docxworld-fetch-production-environment>*-Element. Im Prinzip müssen Sie kaum mehr tun als das *<docxworld-fetch-production-environment>* mit den notwendigen Subelementen zu notieren.

Voraussetzung ist natürlich immer, dass die vom Auftrags-System (Spooler) empfangene Ressource sich überhaupt für die Zusammenführung mit Paketen aus einem Redaktions-System eignet (Spooler-XML).

Ist dies der Fall, ist der erste Teil die Ressource zu empfangen:

```
 <fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />
```

Dies ist jetzt die Ressource, die wir als Rohdaten für die Pakete aus dem Redaktions-System verwenden möchten. Damit können wir jetzt auch schon beginnen:

```
<docxworld-fetch-production-environment id="rsProductionEnvironment"
runtime-environment="shared">

<docxworld-contract>${process['eoms.procedure']}</docxworld-contract>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->

<binary-bundles-home>${binaryBundleHome}</binary-bundles-home>
<production-bundles-home>${productionBundleHome}</production-bundles-ho
me>

<runtime-home>${workdir.getAbsolutePath()}</runtime-home>
<merge-home>${buildPath4ClientProcedure(process['eoms.client'],
process['eoms.procedure'])}</merge-home>

</docxworld-fetch-production-environment>
```

Gehen wir den Code Schritt für Schritt durch:

Für unser <docxworld-fetch-production-environment> wählen wir die Umgebungsart "shared". Was das bedeutet wird hier nicht thematisiert, lesen Sie dazu die Elementbeschreibung.

Wir notieren dann folgende Subelemente:

<docxworld-contract>: Da wir mit docxworld-Vertragsnummern arbeiten, geben wir hier die Vertragsnummer aus dem Job an. Dies ist notwendig, damit den Jobdaten eindeutig die passenden Pakete im Redaktions-System zugewiesen werden. Eine alternative Zuordnung ist über docxworld-Links möglich (*<link-name>*). Sie können aber immer nur eine Zuweisungsmethode verwenden. Sprechen Sie sich deshalb mit dem Administrator des Redaktions-Systems und des Auftrags-Systems ab, welche Methode verwendet wird, denn dies ist bereits im Redaktions-System und in den Jobdaten fest vorgegeben. Wenn Sie hier die falsche Methode angeben, kann keine Zuordnung stattfinden.

Als nächstes geben wir den Pfad an, an dem Binär- und Produtions-Pakete lokal auf dem Worker abgelegt werden sollen (Cache). Dazu dienen die Elemente <binary-bundles-home> und <production-bundles-home>. Von dort werden die Pakete dann ins temporäre Jobverzeichnis kopiert oder direkt aufgerufen (je nachdem, ob merged oder shared verwendet wird).

Der letzte Schritt ist, das Homeverzeichnis für die Ausführung festzulegen: Dazu müssen Sie entweder nur *<runtime-home>* oder *<merge-home>* verwenden, je nachdem, ob Sie shared oder merged nutzen.

Damit ist die Anbindung an das Redaktions-System bereits eingerichtet. Jetzt können wir das Programm aus dem Binär-Paket ausführen und die Daten zusammenführen:

```
<exec id="rwProcess" process="rwProcess" workdir="workdir">
<param name="cmd"
value="${rsProductionEnvironment.getCommandLine(inputFile.getFile())}"
/>
<commandline processor="velocity">$cmd</commandline>
</exec>
```

Schlussendlich verwenden wir noch den Befehl aus den Rohdaten (Spooler-XML, getCommandLine()) um das Programm im Binär-Paket auszuführen und die Rohdaten mit dem Produktions-Paket zusammenzuführen. Danach empfiehlt es sich, noch zu überprüfen, ob die Verarbeitung auch gelungen ist und dann die Daten z.B. zurück an das Auftrags-System zu schicken. Dies wird hier aber nicht mehr gezeigt, da dies völlig unabhängig vom Redaktions-System passiert. Die vollständige Prozedur finden Sie in der Standard-OMS-Worker-RCML.

JavaScript in RCML

RCML ermöglicht die Einbettung von Skriptsprachen direkt in ihre Prozessdefinitionen. Damit steht Ihnen der komplette Umfang und die Flexibilität der Skriptsprache uneingeschränkt zur Verfügung. Momentan wird nur JavaScript als Sprache unterstützt. Falls Sie mit JavaScript nicht vertraut sind, empfiehlt es sich, zuvor ein JavaScript Tutorial zu lesen.

Die Einbettung des Skripts erfolgt, wie bei HTML auch, über das ***<script>***-Element. Skript kann nicht aus externen .js-Dateien eingebunden werden. Funktionen und Objekte, die Sie in JavaScript definieren, stehen in RCML uneingeschränkt zur Verfügung:

```
<script language="JavaScript">
<![CDATA[
var workdir_home = "C:/temp/worker/work";
function getErrorMsg(err){
if(err == 400)
return "Fehlercode 400 aufgetreten: Bad Request.";
else
return "Unbekannter Fehler aufgetreten: " + err;
}
]]>
</script>

<workdir id="workdir" home="workdir_home" />
<exec id="ExampleExec" workdir="workdir" />
...
</exec>

<if condition="${ExampleExec.getReturnCode() != 0}">
<error message="${getErrorMsg(ExampleExec.getReturnCode())}" />
</if>
```

Wie Sie sehen ist es auch ohne Weiteres möglich, Werte aus RCML als Parameter für JavaScript-Funktionen zu übergeben und umgekehrt Rückgabewerte aus JavaScript-Funktionen als Werte im RCML.

Da der EOMS-Worker mit einem vollwertigen JavaScript-Interpreter ausgestattet ist, steht Ihnen innerhalb von ***<script>*** die komplette Umfang von JavaScript zur Verfügung. Sie können also auch ohne Probleme Klasse deklarieren und nutzen, externe Library's einbinden, usw.

Umgang mit Return-Codes

Der Return-Code zeigt an, ob während der Verarbeitung eines Codeteils Fehler aufgetreten sind und wenn ja, welcher. Im RCML-Code oder im Auftrags-System kann dann entsprechend reagiert werden. Nur wenn der Rückgabewert 0 ist sind keine Fehler aufgetreten.

Es gibt momentan 2 Möglichkeiten, auf Return-Code eines Codeteils (eines Elements) zu reagieren.

1. Sofortige Beendigung der Verarbeitung ohne Rückgabe von Dateien

Innerhalb eines ***<exec>***-Elements kann über den ***<result>***-Tag auf den Rückgabewert eines innerhalb des **<exec>**-Tags (zuletzt) ausgeführten ***<commandline>*** zugegriffen werden. Dabei können boolsche Vergleiche mit dem Rückgabewert angestellt werden. Ergibt der Vergleich eine wahre Aussage wird die Verarbeitung nach dem ***<result>***-Tag fortgesetzt. Ergibt der Vergleich eine falsche Aussage, wird die Verarbeitung an dieser Stelle abgebrochen und es werden keine Ergebnisdaten aus dem ***<commandline>***-Aufruf an das aufrufende System zurückgegeben. Das aufrufende System empfängt dann lediglich den Rückgabewert.

```
...
<exec id="ExampeExec">
<commandline processor="velocity">$cmd</commandline> <!-- generiert
return-code = 500 -->
<result return-code="403 | 404" /> <!-- Fortsetzung nur bei return-code
= 403 und return-code = 404 -->
</exec>
...
```

Im obigen Beispiel schlägt der ***<commandline>***-Aufruf mit *return-code = 500* fehl. Im ***<result>***-Tag wird abgefragt, ob der return-code entweder 403 oder 404 ist. In diesem Fall würde die Verarbeitung hier fortgesetzt und nachfolgender Code ausgeführt. Da der Rückgabewert aus ***<commandline>*** aber 500 ist, wird die Verarbeitung an dieser Stelle abgebrochen.

2. Beendigung der Verarbeitung mit unterschiedlichen Reaktionen auf unterschiedliche Rückgabewerte

Die 2. Möglichkeit zur Abfrage von ***<exec>***-Return-Codes sind ***<if>***-Abfragen in Verbindung mit der *testReturnCode()*-Funktion. Damit lässt sich für verschiedene Fehlercodes unterschiedliches Verhalten implementieren. Die Abfragelogik steht dabei nicht innerhalb des ***<exec>***-Elements, sondern danach.

```
...
<exec id="ExampleExec">
<commandline processor="velocity">$cmd</commandline> <!-- endet
erfolgreich mit 0. -->
</exec>

<if condition="${ExampleExec.testReturnCode('-400')}"> <!-- Wird nur
ausgeführt, wenn return-code = -400. Dies ist hier nicht der Fall. Code
wird also nicht ausgeführt. -->
<error message="Font-Problem, Dokument trotzdem generiert ..."/>
<upload file="workdir"
destination="${process['eoms.process.output']}"/>
</if>

<if condition="${ExampleExec.testReturnCode('0')}"> <!-- Wird nur
ausgeführt, wenn return-code = 0. Dies ist hier der Fall. -->
<message text="${'ReturnCode OK: ' + TestExec.getReturnCode()}"/> <!--
Message mit entsprechendem Return-Code an das EOMS-Core schicken -->
<upload file="workdir"
destination="${process['eoms.process.output']}"/> <!-- Datei hochladen.
-->
</if>
...
```

Die Funktion *getReturnCode()* wird auf das ***<exec>***-Element angewendet (.) und bekommt als Parameter den zu vergleichenden Wert. Entspricht der Return-Code nicht dem übergebenen Vergleichsmuster gibt die Funktion **false** zurück und die ***<if>***-Anweisung wird nicht ausgeführt. Im obrigen Fall werden nur die Zeilen 14 und 15 ausgeführt. In Zeile 14 wird außerdem die Funktion *getReturnCode()* benutzt , die den return-code zurückgibt. Im obrigen Fall würde in Zeile 14 also ausgeführt: "*<message text ="ReturnCode OK: 0" />*".

3. Manuelles Setzen des Rückgabewerts

Der Rückgabewert eines EOMS-Prozesses kann in der RCML nicht nur abgefragt, sondern auch gesetzt werden. Dies gilt allerdings nur für den Rückgabewert des gesamten RCML-Prozesses (***<process>***), nicht für die Rückgabewerte einzelner Elemente (***<exec>***). Das Setzen des Rückgabewerts geschieht mit Hilfe des ***<update-variable>***-Elements.

```
...
<switch>
<case condition="${myprocess.testReturnCode('0')}">
<upload file="workdir"
destination="${process['eoms.process.output']}"/>
<message text="myprocess normal beendet."/>
</case>
<!-- Nur folgender case wird ausgeführt. -->
<case condition="${myprocess.testReturnCode('3')}"> <!-- Ergibt wahre
Aussage und wird damit ausgeführt. -->
<upload file="workdir"
destination="${process['eoms.process.output']}"/> <!-- Da es sich nur
um eine Warnung und keinen Fehler handelt, wird die Datei trotzdem
hochgeladen. -->
<update-variable name="return-code" value="0" /> <!-- return-code wird
von 3 auf 0 gesetzt. -->
<message text="${'WARNUNG: '+myprocess.getReturnCode()+' Bad Bounding
Boxes?'}"/> <!-- Zum Schluss noch Warnung an das EOMS-Core schicken.
-->
</case>
<!-- Ende Ausführung. -->
<case condition="true">
<error message="${'myprocess fehlerhaft beendet!
Return-Code:'+myprocess.getReturnCode()}"/>
</case>
</switch>
...
```

Im obigen Beispiel wird bereits vor der ***<switch>***-Abfrage im Prozess der Return-Code 3 generiert (z.B. durch ***<exec>***). In der ***<switch>***-Anweisung werden dann 3 verschiedene Szenarien abgefragt *(return-code = 0; return-code = 3; return-code ungleich 0)*. Da in einer ***<switch>***-Anweisung die ***<case>***-Abfragen der Reihe nach geprüft werden wird hier nur das 2., nicht jedoch das 3. ***<case>*** wahr (im Unterschied zu manchen höheren Programmiersprachen wird in RCML nach einem wahren ***<case>*** nicht weiter geprüft). Da der return-code 3 keinen Fehler, sondern lediglich eine Warnung signalisiert, soll nun der return-code nach entsprechender Behandlung wieder auf 0 (fehlerfrei) zurückgesetzt werden. Dies geschieht durch den ***<update-variable>***-Tag, dem der Name der zu ändernden Variable ("return-code") und der neue Wert als Attribute übergeben wird.

Variablenaustausch mit Auftrags-Systemen

In RCML besteht die Möglichkeit, Variablenwerte zu setzen, die dann auch im aufrufenden System (z.B. Spooler) verfügbar sind. Ebenso können dem EOMS direkt beim Aufruf durch das Auftrags-System Variablen mitgegeben werden.

Variablen aus dem Auftrags-System (Spooler) zu EOMS

Variablen für das EOMS werden in der Aufrufzeile des Auftrags-Systems (siehe hier für Spooler) definiert. Die Variablen haben dabei die Form 'EOMS.XXX', also z.B. EOMS.MYVAR oder EOMS.CLIENT.CONNECTIONINFO. Es gelten die üblichen Kriterien für Variablennamen. Neben den vordefinierten Systemvariablen wie eoms.invoker.host oder eoms.invoker.jms.host, die für die Konfiguration wichtig sind, können Sie unbegrenzt viele selbstdefinierte Variablen übergeben. Beispiel einer Aufrufzeile aus dem Spooler:

```
eoms.invoker.host=localhost eoms.invoker.port=8080
eoms.invoker.jms.host=polling eoms.invoker.jms.port=polling
eoms.invoker.transfer=http eoms.process=copy
eoms.author=Simon_Schoenwaelder eoms.date=01_09_2014
eoms.company=profiforms copy.output=@SourceName.dat
```

Hier werden die selbstdefinierten Variablen eoms.author, eoms.date und eoms.company übergeben.

In RCML kann dann direkt auf die Variablen referenziert werden:

```
<message text="${'Copyright: ' + process['eoms.author'] + ', ' +
process['eoms.company'] + ' on the ' + process['eoms.date']}"/>
```

Der Zugriff geschieht dabei über das ***process***-Feld.

Beachten Sie, dass in RCML die Variablen immer in Kleinbuchstaben angesprochen werden, auch wenn sie in der Aufrufzeile großgeschrieben sind!

Variablen aus dem EOMS zum Auftrags-System

Genauso simpel ist es, Variablen aus dem RCML im Auftrags-System verfügbar zu machen. Dazu dient das ***<update-variable>***-Element , das auch schon im How-To Return-Codes genutzt wurden, um den return-code eines Prozesses zu setzen. Es besteht allerdings keine Garantie dafür, dass das Auftrags-System die Variablen nicht verwirft. Beispiel:

```
<update-variable name="company" value="profiforms" />
```

Der Zugriff im Auftrags-System erfolgt über den Präfix "EOMS:PROCESS:XXX". Die Variable EOMS:PROCESS:COMPANY hat im Auftrags-System (Spooler) jetzt den Wert "profiforms".

Variablennamen müssen im RCML in Kleinbuchstaben definiert werden!

RCML-Elemente

Hier soll Ihnen ein Überblick über die RCML-Elemente *("Tags")* für das EOMS gegeben werden. Jedes Element wird ausführlich auf einer eigenen Seite beschrieben. In den Elementbeschreibungen wird vorausgesetzt, dass Sie mit dem Konzept von RCML vertraut sind.

Die Elementbeschreibungen finden Sie als diesem Artikel untergeordnete Seiten mit dem exakten Namen des Elements. Dabei befindet sich jedes Element entweder direkt unterhalb von *<process>,* was bedeutet, dass dieses Element in allen Top-Level-Element (Ausnahmen werden extra gekennzeichnet) notiert werden darf und nur vorgegeben ist, dass *<process>* ein Elternelement sein muss, oder unterhalb des Elements, dass explizit ein Elternelement sein muss. Unterschieden wird zwischen 3 Arten von Elementen:

- Das Root-Element
    - Bei dem Root-Element handelt es sich um das <rcml>-Element, das jedoch außer der Auszeichnung von RCML-Code ke Bedeutung hat und deshalb auch hier nicht ausgeführt wird. Wie Sie aber bereits im Tutorial gelernt haben, muss jegliche innerhalb von <rcml></rcml> stehen. Ein RCML-Dokument besteht also im Grunde nur aus einem einzigen Element mit U deshalb handelt es sich hier um das Wurzelelement ("Root"). Wichtig ist, dass ein <rcml>-Element als direkte Unterelem*s>* und *<script>* haben darf! Alle anderen Elemente werden dann als Unterelemente eines *<process>-Elements* notiert.

- Top-Level-Elemente (momentan gibt es davon 7)
    - Top-Level-Elemente sind Elemente, die andere Elemente beinhalten können. <rcml> an sich ist also auch ein Top-Level- besonderen Status als Top-Level-Element hat *<process>,* dass Elternelement aller RCML-Elemente (außer *<script>* und <rcml>) sein muss. ***<switch>*** ist das einzige Top-Level-Element, das keine beliebigen RCML-Elemente beinhalten darf, s *se>*.

- Standalone-Elemente (momentan gibt es davon 24)
    - Standalone-Elemente sind Elemente, die keine anderen Elemente beinhalten können. Das bedeutet jedoch nicht, dass si (z. B. einen Befehl in Textform wie bei *<commandline>*) haben können. Standalone-Elemente dürfen, wenn nicht anders innerhalb jedes Top-Level-Elements notiert werden (Ausnahmen sind Subelemente).

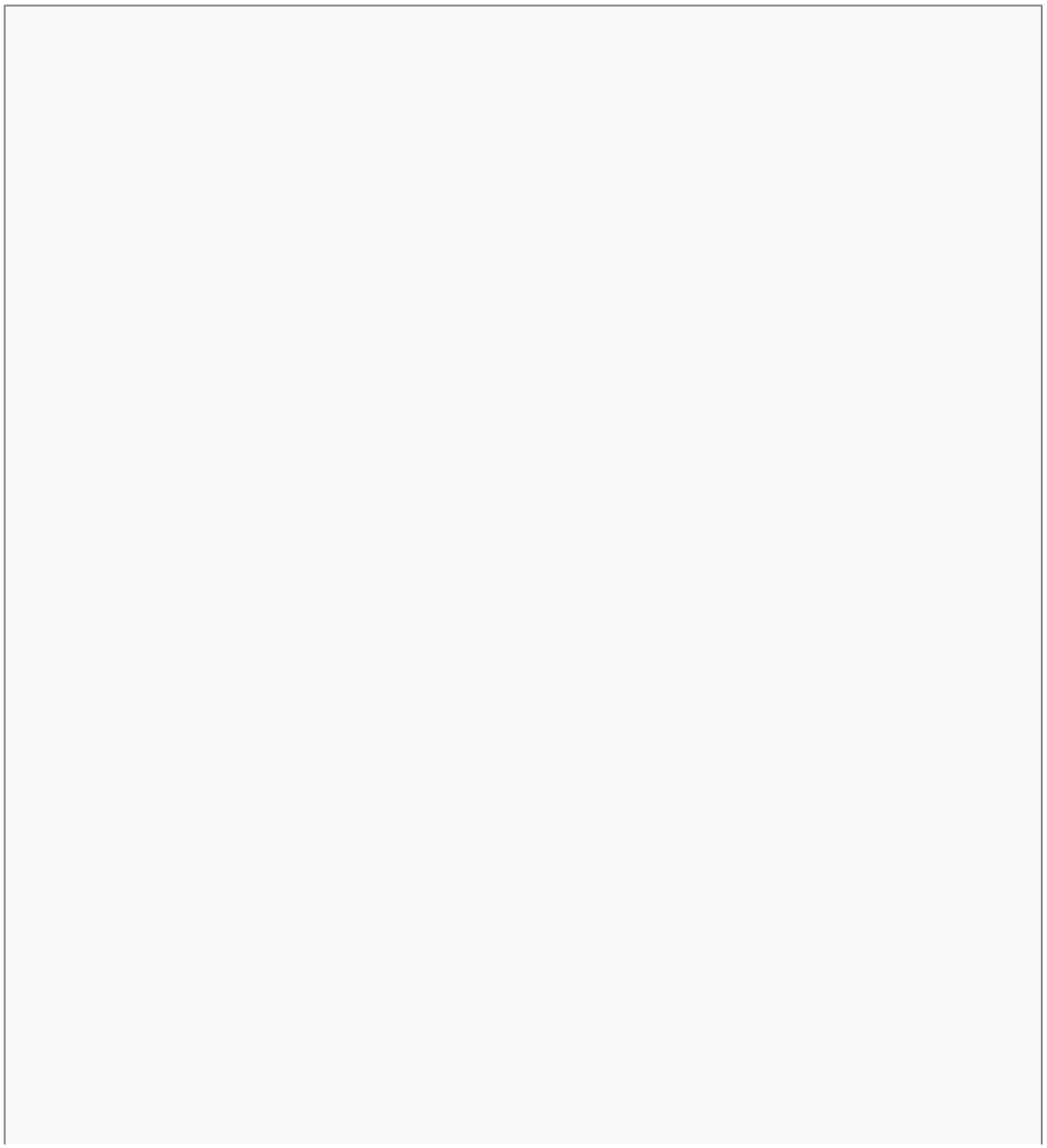
Alle Elemente außer *<script>* müssen zwingend in einem <process>-Element notiert werden. Es gibt aber auch Elemente, die nur innerh alb eines weiteren bestimmten Top-Level-Elements notiert werden dürfen, z. B. das eben erwähnte *<commandline>,* das neben *<process >* auch *<exec>* als Elternelement haben muss. Dabei muss solch ein Element, das ein zwingend vorgegebenes Element als Elternelement haben muss, nicht auch unbedingt direkt unterhalb des Elternelements notiert werden (siehe Tutorial). Momentan gibt es 4 Elemente, die nicht frei platziert werden können, sondern ein weiteres zwingendes Elternelement neben *<process>* haben müssen: *<par am>, <commandline>, <result>* (innerhalb von *<exec>*) sowie *<case>* (innerhalb von *<switch>).*
Hier ein Überblick über alle momentan existierenden Elemente (Klicken Sie auf ein Element, um zur Beschreibung zu gelangen):

- <process>
    - <delete>
    - <destination>
    - <docxworld-fetch-production-environment>
        - <docxworld-contract>
        - <link-name>
        - <binary-bundles-home>
        - <production-bundles-home>
        - <merge-home>
        - <runtime-home>
    - <eoms-input-query-data>
    - <eoms-input-query-status>
    - <eoms-input-submit>
    - <error>
    - <exec>
        - <param>
        - <commandline>
        - <result>
    - <fetch-production-bundle>
    - <fetch-production-environment>
    - <fetchresource>
    - <if> - <else>
    - <message>
    - <releaseresource>
    - <rw-response>
    - <sleep>
    - <switch>
        - <case>
    - <update-variable>
    - <upload>
    - <while>
    - <workdir>
- <script>

Einrückungen signalisieren, dass das Element ein Unterelement gegenüber dem Element ist, zu dem es eingerückt ist. So muss z. B. *<switch>* (wie alle Elemente) innerhalb eines *<process>* notiert werden (dabei ist egal ob direkt in *<process>* oder verschachtel, nur muss *<process>* ein Elternelement sein) und *<case>* innerhalb eines *<switch>.*

Aufbau der Elementbeschreibungsseiten

Jede Elementbeschreibung ist nach folgendem simplen und übersichtlichen Muster aufgebaut:



Semantik

[Hier steht die Elementbeschreibung]

Attribute

[Hier werden die Attribute des Elements aufgelistet]

Subelemente / Inhalt

[Hier werden die Subelemente und der Inhalt des Elements beschrieben]

Variablenbindungen

[Hier werden die Variablenbindungen des Elements beschrieben]

Beispiel

[Hier findet sich ein Beispiel, wie das Element in der Praxis eingesetzt werden kann]

<exec>

Zweck:Typ:Elternelement:

[Elementfunktion]

[Standalone oder Top-Level]

[Erlaubte Elternelement(e)]

Subelemente:Variablenbindungen:

[Kann das Element Elemente enthalten?]

[Hat das Element Variablenbindungen?]

Verzweigungsstruktur

[Die Verzweigungsstruktur für dieses Element (= Wo kann / darf das Element platziert werden?)]

Beachte

[Anmerkungen, die beachtet werden müssen.]

Im Folgenden sollen nun die einzelnen Teilbereiche detailliert erläutert werden:

Die Elementbeschreibungen sind in ihrem Aufbau zweigeteilt: Links der Inhalt der Elementbeschreibung, rechts allgemeine Informationen über das Element in Kurzform.

1. Die linke Spalte (Inhalt)

---

1. Semantik

Hier finden Sie eine allgemeine Beschreibung zu dem Objekt, generelle Hinweise zur Verwendung des Objekts und nützliche Hinweise

2. Attribute

Hier werden sämtliche Attribute tabellarisch aufgelistet.

- Attributname.
- Datentyp: Der Datentyp dieses Attributs. Die übergebenen Werte müssen von diesem Typ sein. Es gibt folgende D STRING (Zeichenkette), INTEGER (Ganzzahl), LONG (Ganzzahl), FILEOBJECT (Datei/Verzeichnis), FETCHEDR (Ressource aus einem *<fetchresource>*), BOOLEAN (Wahrheitswert true/false).
- Beschreibung.
- Mögliche Werte: Eine Beschreibung des Wertebereichs für dieses Attribut.
- Standardwert: Der Standardwert, der automatisch verwendet wird, falls das Attribut nicht angegeben wird.
- Obligatorisch: Gibt an, ob das Attribut obligatorisch (Pflichtattribut, muss angegeben werden) ist oder nicht

3. Subelemente / Inhalt

Hier werden die Subelemente des Elements aufgelistet. Nur Top-Level-Elemente können andere Elemente beinhalten. Beachten Sie, dass hier nur Elemente aufgelistet werden, die ausschließlich innerhalb dieses Elements notiert werden dürfen, die also zu diesem Element "gehören". Solche Elemente dürfen nicht alleine stehen, sondern müssen immer als Inhalt dieses Elements stehen. Abgesehen davon dürfen in Top-Level-Elementen in der Regel fast alle RCML-Elemente (außer *<rcml>* und *<process>)* notiert werden. Subelemente eines Top-Level-Elements können also neben den hier aufgelisteten Elementen beinahe alle anderen RCML-Elemente sein. Wo ein Element notiert werden darf steht auch in der Informationsspalte unter Elternelement.

Nur weil in einem Element keine anderen RCML-Elemente notiert werden dürfen, heisst das noch nicht, dass das Element keinen Inhalt haben darf. Solcher Inhalt kann z. B. reiner Text sein (wie im Falle von *<commandline>*).

4. Variablenbindungen

Hier werden Ihnen die Variablenbindungen des Elements (falls vorhanden) tabellarisch aufgelistet:

- **Name**
- **Beschreibung**
- **Rückgabetyp:** Der Datentyp, von dem der Wert ist, der von der Bindung zurückgegeben wird.

Variablenbindungen sind Funktionen eines Elements, auf die Sie zugreifen können und die einen bestimmten Wert, eine Eigenschaft des Elements, zurückgeben. Eine genau Beschreibung finden Sie im Tutorial.

5. Beispiel

Hier finden Sie ein praxisnahes Beispiel, dass den Einsatz des Elements und wichtige Kernfunktionen exemplarisch aufzeigt.

2. Die rechte Spalte (Übersicht)

---

1. Allgemeine Informationen

Diese Box enthält allgemeine Informationen über das Element:

- Zweck: Wozu dient das Element?
- Typ: Der oben bereits erwähnte Typ eines Elements gibt an, ob das Element andere Elemente beinhalten darf. "St
  bedeutet, das Element darf keine Elemente beinhalten, "Top-Level" bedeutet, das Element darf andere Elemente b
  ein Top-Level-Element nur bestimme andere Elemente beinhalten ist dies hier durch ⚠ gekennzeichnet und in "**S
  Inhalt**" bzw. "**Beachte**" weiter ausgeführt.
- Elternelement: Gibt an, innerhalb welches Elementes das Element notiert werden muss. Hier kann entweder ein be
  RCML-Element stehen oder "Top-Level-Elemente". Ein bestimmtes RCML-Element bedeutet, dass es sich hier um
  dieses Elternelements handelt. D.h. das Element darf nur innerhalb dieses Elements notiert werden (wo im Dokum
  Elternelement notiert ist ist dabei egal). Die Angabe "Top-Level-Elemente" bedeutet, dass das Element in jedem El
  werden darf, dass als **Typ** "Top-Level" ist.
- Subelemente / Inhalt: Gibt an, ob das Element Inhalt haben darf. Nur bei Top-Level-Elementen darf dieser Inhalt au
  Elemente bestehen, aber auch Standalone-Elemente können Inhalt (z. B. Text) enthalten. Erläuterungen dazu find
  der linken Spalte unter 3.
- Variablenbindungen: Gibt an, ob das Element Variablenbindungen besitzt. Diese werden dann in der linken Spalte
  erläutert.

2. Verzweigungsstruktur

Die Verzweigungsstruktur stellt übersichtlich dar, wo in einem RCML-Dokument das Element stehen darf. Die Elementpositionen sind dabei relativ zu verstehen. Das bedeutet, es werden nur die minimalen Eltern-Kind-Beziehungen gezeigt. Dazu wird sozusagen eine Beispielstruktur eines gültigen RCML-Dokuments gezeigt, das alle Elemente enthält, die das Element als direkte Elternelemente haben kann. Die Verzweigungsstruktur zeigt also ein gültiges RCML-Dokument, welches das Element einmal an allen möglichen Positionen enthält, wobei nur die direkten Elternelemente für das Element vorgegeben sind. Die Elternelement selbst müssen <*process*> aber nur als indirektes Elternelement haben! Das Element selbst wird violett gekennzeichnet, Elternelemente fett, eventuelle Kindelemente normal.

Da dies vielleicht nicht auf Anhieb zu verstehen ist, wollen wir die Vorgehensweise durch ein paar Beispiele anschaulich machen:

Verzweigungsstruktur

- <rcml>
  - <process>
    - <error>
    - <exec>
      - <error>
    - <if>/<else>
      - <error>
    - <switch>
      - <case>
        - <error>
    - <while>
      - <error>

Hier sehen Sie die Verzweigungsstruktur für das Element ***<error>***. Was können wir daraus nun lesen?

Als erstes sehen wir, dass ***<error>*** nicht direkt in *<rcml>* stehen darf, sondern nur in *<process>*, welches wiederum in *<rcml>* stehen muss. Die 1. mögliche Position von ***<error>*** ist also: An einer beliebigen Stelle in einem *<process>* (das in einem *<rcml>* steht).

Gehen wir weiter: Die nächste Position ist innerhalb eines *<exec>*. Die 2. mögliche Position ist also innerhalb eines *<exec>*. Nun kommt zum Tragen, dass in einer Verzweigungsstruktur nur die minimalen zwingenden Elternelemente notiert sind: *<exec>* ist hier innerhalb von *<process>* notiert. Das bedeutet, *<exec>* muss zwingend *<process>* als Elternelement haben, es muss aber nicht das direkte Elternelement sein! Erlaubt wäre also genauso *<rcml>* - *<process>* - ... beliebige Elemente, in denen exec stehen darf ... - *<exec>* - *<error>.* Wichtig ist nur, dass ***<error>*** als direktes Elternelement entweder *<process>, <exec>, <if>, <else>, <case>* oder *<while>* haben muss.

Eine Besonderheit sehen wir bei *<switch>*: Unterhalb von *<switch>* ist noch *<case>* notiert. Das bedeutet, *<error>* darf in einem *<case>* stehen, dass wiederum zwingend in einem *<switch>* stehen muss. Da in der Verzweigungsstruktur kein ***<error>*** direkt unterhalb von *<switch>* notiert ist, darf ***<error>*** nicht direkt in einem *<switch>* stehen!

Um noch einmal zusammenzufassen: Das Element selbst ist in der Verzweigungsstruktur nur in seinen direkten Elternelementen notiert (*<error>* darf nur die direkten Elternelemente haben, in denen es hier steht). Diese Elternelemente selbst müssen *<process>* aber nur als indirektes Elternelement haben. Schachtelungen sind also erlaubt. Dazu müssen Sie dann beachten, in welchem Element wiederum das Elternelement stehen darf. Beispiel: Sie haben eine RCML mit folgender Struktur:

```
<rcml>
<process>
<while>
<exec>
<commandline />
</exec>
<message />
</while>
</process>
</rcml>
```

Die Verzweigungsstruktur dafür sieht so aus:

Verzweigungsstruktur

- <rcml>
  - <process>
    - <while>
      - <exec>
        - <commandline>
      - <message>

Wäre dies gültig? Ja, denn:

In der Beschreibung zu *<commandline>* sehen wir:

Verzweigungsstruktur

- <rcml>
  - <process>
    - <exec>
      - <commandline>

<commandline> darf also innerhalb eines *<exec>* stehen (und nur da!). Außerdem sehen wir, dass *<exec>* nur *<process>* als indirektes Elternelement haben muss. Auch das wird erfüllt. Als nächstes müssen wir schauen, ob *<exec>* in einem *<while>* stehen darf. Wir schauen uns also die Verzweigungsstruktur zu *<while>* an. Dort sehen wir:

Verzweigungsstruktur

- <rcml>
  - <process>
    - <while>
      - <exec>

Dasselbe überprüfen wir dann noch für *<message>*.

Ein Spezialfall ist, wenn ein Element noch Kindelemente hat, die nur dort notiert werden dürfen. Dies sehen wir am Beispiel *<exec>*:

Verzweigungsstruktur

- <rcml>
  - <process>
    - <exec>
      - <param>
      - <commandline>
      - <result>
    - <exec>
      - <exec>
        - <param>
        - <commandline>
        - <result>
    - <if>/<else>
      - <exec>
        - <param>
        - <commandline>
        - <result>
    - <switch>
      - <case>
        - <exec>
          - <param>
          - <commandline>
          - <result>
    - <while>
      - <exec>
        - <param>
        - <commandline>
        - <result>

Wie Sie sehen, werden solche Kindelemente in einfachem Text notiert.

3. Beachte

Enthält wichtige Informationen, die unbedingt beachtet werden sollten.

<process>

Semantik

Ein **<process>**-Element definiert einen RCML-Prozess. Ein RCML-Dokument darf auf höchster Ebene (<rcml>) nur **<process>**-Elemente und <script>-Elemente enthalten. Alle anderen Elemente müssen in einem **<process>**-Element notiert werden. Ein RCML-Dokument besteht also im Wesentlichen aus **<process>**-Elementen, die durch ihre Subelemente beschrieben werden. **<process>** ist sog. Root-Element, also oberstes Element in der RCML-Hierarche (abgesehen von <rcml>, das semantisch keine Bedeutung hat und nur RCML-Inhalt signalisiert.). **<process>**-Elemente stellen die Prozessdefinitionen eines Workers dar. Für jeden Job wird exakt ein Prozess und damit ein **<process>**-Element aufgerufen und abgearbeitet. Alle anderen **<process>** der RCML werden für diesen Job ignoriert und nicht ausgeführt.

Die id des <process> muss unbedingt mit der id übereinstimmen, die in der List of Tasks (der *.properites) des Workers angegeben wurde, da der Worker den Prozess sonst nicht finden kann! Achten Sie dabei darauf, die Datei des richtigen Workertyps zu bearbeiten: Wenn die Prozesse in der worker.rcml eingetragen sind (also für den OMS-Worker bestimmt sind) müssen Sie natürlich auch die worker.properties bearbeiten.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Prozesses. Über diese ID wird der Prozess dann in der List of Tasks des Workers und in den Jobinformationen angesprochen.	beliebige, regelkonforme Zeichenkette.	—	
name	***STRING***	Optionale Angabe eines Prozessnamens zur besseren Übersicht.	beliebige, regelkonforme Zeichenkette.	—	

Subelemente / Inhalt

Das **<process>**-Element kann alle RCML-Elemente enthalten (außer <rcml>). Nur <rcml>, **<process>** und <script> dürfen außerhalb von **<process>** stehen. <rcml> ist das RCML-Wurzelelement und darf keine Elternelemente haben, also auch nicht in einem **<process>**-Element stehen (<rcml> ist immer Elternelement aller **<process>**-Elemente und damit aller RCML-Elemente). **<process>** darf kein Subelement eines anderen **<process>**-Elements sein (**<process>**-Elemente dürfen nicht geschachtelt werden). <script> kann sowohl außerhalb als auch innerhalb von **<process>** stehen.

Variablenbindungen

Das ***<process>***-Element besitzt keine Variablenbindungen.

Beispiel

In folgendem Beispiel werden 3 Prozesse definiert: "copy", "rs" und "rw". Durch Verknüpfung in der List of Tasks des Workers kann der Worker dann Jobs für diese 3 Prozesse empfangen und verarbeiten.

```
<rcml>
<process id="copy" name="Copy"><!-- Hier stehen RCML-Elemente, die
den Prozess beschreiben --></process>
<process id="rw" name="ReportWriter"><!-- Hier stehen
RCML-Elemente, die den Prozess beschreiben --></process>
<process id="rs" name="Redaktions-System"><!-- Hier stehen
RCML-Elemente, die den Prozess beschreiben --></process>
</rcml>
```

Damit diese Prozesse auch aufrufbar sind, müssen Sie sie in der List of Tasks des Workers verfügbar machen. Angenommen, wir haben die worker.rcml bearbeiten. Dann müssen die Prozesse auch in der worker.properties eingetragen werden (hätten wir die eoms-input.rcml bearbeiten, müssten wir eoms-input.properties bearbeiten):

```
# list of processes accepted by consumer
eoms.process = copy
eoms.process.group = *
eoms.process1 = rw
eoms.process1.group = *
eoms.process2 = rs
eoms.process2.group = *
```

<process>

Zweck:Typ:Elternelement:

Prozessdefinition

Top-Level

<rcml>

Subelemente:Variablenbindungen:

Ja

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**

Beachte

<process> darf alle RCML-Elemente außer *<rcml>* (Das Wurzelelement jedes RCML-Dokuments) enthalten. *<script>* ist das einzige andere Element neben ***<process>***, das ebenfalls direkt im *<rcml>*-Element notiert werden darf. Alle anderen RCML-Elemente müssen innerhalb eines ***<process>***-Elements notiert werden.

<delete>

Semantik

Das ***<delete>***-Element löscht Dateien und Verzeichnisse. Bei dem zu löschenden Objekt muss es sich um ein ***FILEOBJECT*** handeln (Datei oder Verzeichnis). Die die direkte Angabe von Systempfaden ist nicht erlaubt.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
file	***FILEOBJECT***	Die ID der Ressource, die gelöscht werden soll.	Ein existierendes *FILEOBJECT*.	—	✓

Subelemente / Inhalt

Das ***<delete>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<delete>***-Element besitzt keine Variablenbindungen.

Beispiel

In folgendem Beispiel wird das Arbeitsverzeichnis, in dem temporäre Dateien gespeichert werden, am Ende des Prozesses manuell durch ***<delete>*** gelöscht. Dies entspricht dem Verhalten wie bei *<workdir clear-on-shutdown=true>*.

```
<rcml>
<process id="ExampleProcess" name="example process">
<!-- Hier wird das Arbeitsverzeichnis festgelegt. -->
<workdir id="myworkdir" home="./WORK"/>
<!-- Hier wird etwas gemacht. -->
...
<!-- Am Ende wird das Arbeitsverzeichnis manuell gelöscht. -->
<delete file="myworkdir"/>

</process>
</rcml>
```

<delete>

Zweck:

Ressourcenlöschung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<delete>**
    - **<exec>**
      - **<delete>**
    - **<if>/<else>**
      - **<delete>**
    - **<switch>**
      - **<case>**
        - **<delete>**
    - **<while>**
      - **<delete>**

Beachte

Existiert das freizugebende Objekt nicht, wird der Zugriff verweigert oder tritt ein sonstiger Fehler auf, wirft ***<delete>*** eine Exception und bricht ab.

<destination>

Dieses Element ist veraltet und wird in der aktuellen Version nicht mehr unterstützt.

<docxworld-fetch-production-environment>

> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit ***<docxworld-fetch-production-environment>*** ist es möglich, Ressourcen (Produktions- und Binär-Pakete) aus einem Redaktions-System anzufordern. Damit stellt ***<docxworld-fetch-production-environment>*** die Anbindung des EOMS an das Redaktions-System bereit. In dem Element müssen Sie eine ganze Reihe an Subelementen notieren, damit eine Zuverlässige Verbindung und Übertragung der Pakete gewährleistet ist. Wenn alle nötigen Subelemente vorhanden und korrekt sind, stellt ***<docxworld-fetch-production-environment>*** eine Verbindung zu dem Redaktions-System her und läd automatisch die für den Job benötigten Pakete auf den Workerrechner, wo sie in einem lokalen Cache gespeichert werden. Die Zusammenführung der Pakete mit der XML-Spooler muss aber noch separat in einem *<exec>* vorgenommen werden. Es gibt 2 unterschiedliche Arten, wie <docxworld-fetch-production-environment> arbeiten kann. Sie geben dies über das Attribut runtime-environment an:

1. **merged (Standard):**
   Bei *merged* wird für jeden Job erneut ein temporäres Verzeichnis angelegt (angegeben durch *<merge-home>*). Zuerst wird geprüft, ob das benötigte Produktions-/Binär-Paket bereits im Cache des Workers vorhanden ist (durch einen vorherigen Job bereits dort abgelegt). Ist dies der Fall, werden die Pakete in das Jobverzeichnis kopiert. Fehlen beide oder 1 Paket, wird eine Verbindung zum Redaktions-System hergestellt und die benötigte Pakete von dort in den lokal Cache des Workers übertragen, von wo sie dann ins Jobverzeichnis kopiert werden. Zusätzlich wird noch die Spooler-XML in das Verzeichnis kopiert. Danach können die Pakete mit der XML zusammengeführt werden. Durch den hohen Kopieraufwand bei jedem Job ist *merged* langsamer als *shared*. Allerdings beherrschen einige externe Auftrags-Systeme *shared* eventuell nicht. Sollten Probleme auftreten, verwenden Sie *merged*.
2. **shared:**
   shared agiert wie *merged*, mit dem Unterschied, dass die Pakete und Spooler-XML nicht für jeden Job neu in ein temporäres Verzeichnis kopiert werden, sondern der Aufruf der Pakete und der XML bei der Zusammenführung erfolgt an dem Ort, wo sie liegen. Das bedeutet, anstatt alle benötigten Dateien in ein Verzeichnis zu kopieren und dort die Ausführung zu starten, wird dem Programm mitgeteilt, wo es die Pakete und die Spooler-XML finden kann. Das Programm greift dann auf die Pakete "aus der Entfernung" zu. *shared* ist wesentlich schneller als *merged* und sollte daher *merged* vorgezogen werden. Falls Ihr Auftrags-System *shared* nicht unterstützt, müssen Sie *merged* verwenden.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Elements, über die es im Code angesprochen werden kann.	beliebiger, regelkonformer Name.	—	✗
runtime-environment	***STRING***	Gibt an, wie die Daten gespeichert und verarbeitet werden sollen: shared (gecached) oder merged. Mehr dazu finden Sie in der Semantikbeschreibung.	"shared" : "merged"	merged	✗
execution-mode	***STRING***	Hiermit lässt sich bestimmen, ob der Prozess in einer produktiven- oder einer Testumgebung laufen soll. Wird hier "test" angegeben, läuft der Prozess in einer Testumgebung. Das bedeutet, es muss hier für den Worker kein Redaktions-System erreichbar sein, sondern der Datenaustausch wird nur simuliert. Wird "production" angegeben, interagiert der Worker ganz normal mit einem Redaktions-System. "error" ist in der momentanen Implementierung identisch zu "test". In einer produktiven Umgebung muss unbedingt "production" gesetzt werden (oder das Attribut weggelassen werden).	"error" : "test" : "production"	production	✗
clear-on-shutdown	***STRING (long)***	Gibt an, ob das Verzeichnis nach Beendigung des Prozesses automatisch gelöscht werden soll (true) oder nicht (false) oder ob es nur gelöscht werden soll, falls kein Fehler aufgetreten ist (keep-on-error)	"true" : "false" : "keep-on-error"	false	✗

Subelemente / Inhalt

Das ***<eoms-input-query-status>***-Element besitzt 6 Subelemente, die nur innerhalb eines ***<docxworld-fetch-production-environment>*** notiert werden dürfen. Davon sind nur *<binary-bundles-home>* und *<production-bundles-home>* obligatorisch.

Um eine zuverlässige Verarbeitung zu garantieren, muss **1** der folgenden beiden Subelemente notiert werden:

- Das **<docxworld-contract>**-Element zur Angabe einer docxworld-Vertragsnummer. Dieses Element darf nur notiert werden, fa kein *<link-name>* benutzt wird.
- Das **<link-name>**-Element zur Angabe eines docxworld-Links. Dieses Element darf nur notiert werden, falls kein *<docxworld-c act>* benutzt wird.

Folgende 2 Elemente sind obligatorisch:

- Das **<binary-bundles-home>**-Element zur Angabe des Home-Verzeichnisses für Binär-Pakete. Dieses Element ist obligatorisch.Jedes ***<docxworld-fetch-production-environment>*** muss dieses Element enthalten.
- Das **<production-bundles-home>**-Element zur Angabe des Home-Verzeichnisses für Produktions-Pakete. Jedes ***<docxworld tch-production-environment>*** muss dieses Element enthalten.

Um eine zuverlässige Verarbeitung zu garantieren, muss 1 der folgenden 2 Subelemente notiert werden, je nachdem, ob Sie als ***runtime-environment*** merged (dann *<merge-home>*) oder shared (dann *<runtime-home>*) verwenden. Es dürfen auch beide Elemente notiert werden.

- Das **<runtime-home>**-Element zur Angabe des Verzeichnisses, in dem bei ***runtime-environment***="shared" Daten gespeicher werden.
- Das **<merge-home>**-Element zur Angabe des Verzeichnisses, in dem bei ***runtime-environment***="merged" Daten gespeichert werden.

Variablenbindungen

Das ***<docxworld-fetch-production-environment>***-Element ist vom Typ ***FETCHED-PRODUCTION-ENVIRONMENT***. Dieser Typ besitzt folgende Variablenbindungen:

Bindung	Beschreibung	Rückgabetyp
getTxApiVersion()	Gibt die API-Version des Produktions-Pakets zurück.	*STRING*
getCommandLine(FILEOBJECT file)	Gibt den Kommandobefehl zurück, der in der übergebenen Datei enthalten ist.	*STRING*
getBinaryBundleQuery()	—	*STRING*
getMessage()	Gibt die interne Message zurück. Eignet sich für Debuggingzwecke.	*STRING*
getDocxworldContract()	Gibt den docxworld-Vertrag zurück, der mit diesem Job assoziiert ist (falls verwendet).	*STRING*
getProductionBundleLinkName()	Gibt den Linknamen des Produktions-Paktes zurück (falls gesetzt).	*STRING*
getResultCode()	Gibt den Rückgabewert des gesamten ***<docxworld-fetch-production-environment>*** zurück (zeigt an, ob die Verarbeitung innerhalb von ***<docxworld-fetch-production-environment>*** erfolgreich war. 0 falls erfolgreich, 4 falls CACHE_HIT, 8 bei einem Fehler.	*INTEGER*
getRuntimeEnvironmentResultCode()	Gibt den Rückgabewert des <runtime-environment> zurück: 0 falls "merged" verwendet wird, 4 falls "shared".	*INTEGER*
getProductionBundleResultCode()	Gibt den Rückgabewert des Produktions-Pakets zurück: 0 falls erfolgreich, 4 falls CACHE_HIT und 8 bei einem Fehler.	*INTEGER*
getBinaryBundleResultCode()	Gibt den Rückgabewert des Binär-Pakets zurück: 0 falls erfolgreich, 4 falls CACHE_HIT, 8 bei einem Fehler und 12 falls kein Binär-Paket angefordert wurde ("shared").	*INTEGER*
getProductionBundleId()	Gibt die ID des Produktions-Paket zurück.	*STRING*
getBinaryBundleId()	Gibt die ID des Binär-Paket zurück.	*STRING*
getSchemaName()	Gibt den Schemanamen zurück, der vom Produktions-Paket benutzt wird.	*STRING*
getSchemaVersion()	Gibt die Schemaversion zurück, die vom Produktions-Paket benutzt wird.	*STRING*

Beispiel

Zu diesem Element gibt es ein How-To mit umfangreichem Beispiel.

<docxworld-fetch-production-environment>

Zweck:

Interaktion mit R-S

Typ:

Top-Level

Elternelement:

Top-Level-Elemente

Subelemente:

Ja

Variablenbindungen:

Ja

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<docxworld-fetch-production-environment>**
      - <docxword-contract>
      - <link-name>
      - <binary-bundles-home>
      - <production-bundles-home>
      - <runtime-home>
      - <merge-home>
    - **<exec>**
      - **<docxworld-fetch-production-environment>**
        - <docxword-contract>
        - <link-name>
        - <binary-bundles-home>
        - <production-bundles-home>
        - <runtime-home>
        - <merge-home>
    - **<if>/<else>**
      - **<docxworld-fetch-production-environment>**
        - <docxword-contract>
        - <link-name>
        - <binary-bundles-home>
        - <production-bundles-home>
        - <runtime-home>
        - <merge-home>
    - **<switch>**
      - **<case>**
        - **<docxworld-fetch-production-environment>**
          - <docxword-contract>
          - <link-name>
          - <binary-bundles-home>
          - <production-bundles-home>
          - <runtime-home>
          - <merge-home>
    - **<while>**
      - **<docxworld-fetch-production-environment>**
        - <docxword-contract>
        - <link-name>
        - <binary-bundles-home>
        - <production-bundles-home>
        - <runtime-home>
        - <merge-home>

Beachte

—

<docxworld-contract>

Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Über ***<docxworld-contract>*** wird die Zuordnung der Spoolerdatei zum richtigen Produktions-Paket / Binär-Paket vorgenommen. Aus dem R-S wird genau das Produktions-Paket / Binär-Paket geladen, dass mit der hier übergebenen docxworld-Vertragsnummer versehen ist (siehe auch docxworld-Verträge im R-S). Über dieses Element ist also die Auswahl des gewünschten Produktions-Pakets / Binär-Pakets aus dem R-S für einen Job möglich. Sie können zur Zuordnung auch *<link-name>* verwenden. Dort wird allerdings keine docxworld-Vertragsnummer zur Identifizierung benutzt, sondern ein docxworld-Link.

Es darf immer entweder nur ein ***<docxworld-contract>*** oder ein *<link-name>* innerhalb eines *<docxworld-fetch-production-environment>* notiert werden, da sich die beiden Verlinkungsarten ausschließen (ein Paket im R-S kann entweder eine Vertragsnummer oder einen Link haben, nicht beides)! Werden beide Elemente notiert, bricht der Prozess mit einer Fehlermeldung ab.

Attribute

Das ***<docxworld-contract>***-Element besitzt keine Attribute.

Subelemente / Inhalt

Das ***<docxworld-contract>***-Element besitzt keine Subelemente. Der Inhalt des ***<docxworld-contract>***-Elements ist die zugeordnete docxworld-Vertragsnummer.

Variablenbindungen

Das ***<docxworld-contract>***-Element besitzt keine Variablenbindungen.

Beispiel

```
<rcml>
<process id="ExampleProcess" name="example process">

...
<docxworld-fetch-production-environment id="ExampleRS"
runtime-environment="shared">
<docxworld-contract>${process['eoms.procedure']}</docxworld-contrac
t>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->

<binary-bundles-home>binary-bundles</binary-bundles-home>
<production-bundles-home>production-bundles</production-bundles-hom
e>

<runtime-home>${workdir.getAbsolutePath()}</runtime-home>

</docxworld-fetch-production-environment>
...

</process>
</rcml>
```

<docxworld-contract>

Zweck:

Paket-Zuordnung

Typ:

Standalone

Elternelement:

<docxworld-fetch-production-environment>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<docxworld-fetch-production-environment>**
      - **<docxworld-contract>**
    - **<exec>**
      - **<docxworld-fetch-production-environment>**
        - **< docxworld-contract>**
    - **<if>/<else>**
      - **<docxworld-fetch-production-environment>**
        - **<docxworld-contract>**
    - **<switch>**
      - **<case>**
        - **<docxworld-fetch-production-environment>**
          - **<docxworld-contract>**
    - **<while>**
      - **<docxworld-fetch-production-environment>**
        - **<docxworld-contract>**

Beachte

Es darf immer entweder nur ein ***<docxworld-contract>*** oder ein *<link-name>* innerhalb eines *<docxworld-fetch-production-environment>* notiert werden, da sich die beiden Verlinkungsarten ausschließen! Werden beide Elemente notiert, bricht der Prozess mit einer Fehlermeldung ab.

<link-name>

Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Über ***<link-name>*** wird die Zuordnung der Spoolerdatei zum richtigen Produktions-Paket / Binär-Paket vorgenommen. Aus dem R-S wird genau das Produktions-Paket / Binär-Paket geladen, dass mit dem hier angegebenen Link verknüpft ist (siehe auch Produktions-Paket-Verknüpfungen im R-S). Über dieses Element ist also die Auswahl des gewünschten Produktions-Pakets / Binär-Pakets aus dem R-S für einen Job möglich. Sie können zur Zuordnung auch *<docxworld-contract>* verwenden. Dort wird allerdings kein Linkname zur Identifizierung benutzt, sondern ein docxworld-Vertrag.

Es darf immer entweder nur ein *<docxworld-contract>* oder ein ***<link-name>*** innerhalb eines *<docxworld-fetch-production-environment>* notiert werden, da sich die beiden Verlinkungsarten ausschließen (ein Paket im R-S kann entweder eine Vertragsnummer oder einen Link haben, nicht beides)! Werden beide Elemente notiert, bricht der Prozess mit einer Fehlermeldung ab.

Attribute

Das ***<link-name>***-Element besitzt keine Attribute.

Subelemente / Inhalt

Das ***<link-name>***-Element besitzt keine Subelemente. Der Inhalt des ***<link-name>***-Elements ist der zugeordnete Linkname.

Variablenbindungen

Das ***<link-name>***-Element besitzt keine Variablenbindungen.

Beispiel

```
<rcml>
<process id="ExampleProcess" name="example process">

...
<docxworld-fetch-production-environment id="ExampleRS"
runtime-environment="shared">
<docxworld-contract>${process['eoms.procedure']}</docxworld-contrac
t>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->

<binary-bundles-home>binary-bundles</binary-bundles-home>
<production-bundles-home>production-bundles</production-bundles-hom
e>

<runtime-home>${workdir.getAbsolutePath()}</runtime-home>

</docxworld-fetch-production-environment>
...

</process>
</rcml>
```

<link-name>

Zweck:

Paket-Zuordnung

Typ:

Standalone

Elternelement:

<docxworld-fetch-production-environment>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<docxworld-fetch-production-environment>**
      - **<link-name>**
    - **<exec>**
      - **<docxworld-fetch-production-environment>**
        - **<link-name>**
    - **<if>/<else>**
      - **<docxworld-fetch-production-environment>**
        - **<link-name>**
    - **<switch>**
      - **<case>**
        - **<docxworld-fetch-production-environment>**
          - **<link-name>**
    - **<while>**
      - **<docxworld-fetch-production-environment>**
        - **<link-name>**

Beachte

Es darf immer entweder nur ein *<docxworld-contract>* oder ein ***<link-name>*** innerhalb eines *<docxworld-fetch-production-environment>* notiert werden, da sich die beiden Verlinkungsarten ausschließen! Werden beide Elemente notiert, bricht der Prozess mit einer Fehlermeldung ab.

<binary-bundles-home>

> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit dem ***<binary-bundles-home>***-Element geben Sie das Verzeichnis an, in dem Binär-Pakete gespeichert werden. Analog dazu wird das Verzeichnis für Produktions-Pakete durch *<production-bundles-home>* gesetzt.

Attribute

Das ***<binary-bundles-home>***-Element besitzt keine Attribute.

Subelemente / Inhalt

Das ***<binary-bundles-home>***-Element besitzt keine Subelemente.

Der Inhalt eines ***<binary-bundles-home>***-Elements ist die Pfadadresse, die als Homeverzeichnis für Binär-Pakete übergeben werden soll.

Variablenbindungen

Das ***<binary-bundles-home>***-Element besitzt keine Variablenbindungen.

Beispiel

```
<rcml>
<process id="ExampleProcess" name="example process">

...
<docxworld-fetch-production-environment id="ExampleRS"
runtime-environment="shared">
<docxworld-contract>${process['eoms.procedure']}</docxworld-contrac
t>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->

<binary-bundles-home>binary-bundles</binary-bundles-home>
<production-bundles-home>production-bundles</production-bundles-hom
e>

<runtime-home>${workdir.getAbsolutePath()}</runtime-home>

</docxworld-fetch-production-environment>
...

</process>
</rcml>
```

<binary-bundles-home>

Zweck:

Pfadangabe (R-S)

Typ:

Standalone

Elternelement:

<docxworld-fetch-production-environment>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<docxworld-fetch-production-environment>**
      - **<binary-bundles-home>**
    - **<exec>**
      - **<docxworld-fetch-production-environment>**
        - **<binary-bundles-home>**
    - **<if>/<else>**
      - **<docxworld-fetch-production-environment>**
        - **<binary-bundles-home>**
    - **<switch>**
      - **<case>**
        - **<docxworld-fetch-production-environment>**
          - **<binary-bundles-home>**
    - **<while>**
      - **<docxworld-fetch-production-environment>**
        - **<binary-bundles-home>**

Beachte

<binary-bundles-home> ist ein obligatorisches Element von *<docxworld-fetch-production-environment>*, es muss also in jedem *<docxworld-fetch-production-environment>* vorkommen.

<production-bundles-home>

> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit dem ***<production-bundles-home>***-Element geben Sie das Verzeichnis an, in dem Produktions-Pakete gespeichert werden. Analog dazu wird das Verzeichnis für Binär-Pakete durch *<binary-bundles-home>* gesetzt.

Attribute

Das ***<production-bundles-home>***-Element besitzt keine Attribute.

Subelemente / Inhalt

Das ***<production-bundles-home>***-Element besitzt keine Subelemente.

Der Inhalt eines ***<production-bundles-home>***-Elements ist die Pfadadresse, die als Homeverzeichnis für Produktions-Pakete übergeben werden soll.

Variablenbindungen

Das ***<production-bundles-home>***-Element besitzt keine Variablenbindungen.

Beispiel

```
<rcml>
<process id="ExampleProcess" name="example process">

...
<docxworld-fetch-production-environment id="ExampleRS"
runtime-environment="shared">
<docxworld-contract>${process['eoms.procedure']}</docxworld-contrac
t>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->

<binary-bundles-home>binary-bundles</binary-bundles-home>
<production-bundles-home>production-bundles</production-bundles-hom
e>

<runtime-home>${workdir.getAbsolutePath()}</runtime-home>

</docxworld-fetch-production-environment>
...

</process>
</rcml>
```

<production-bundles-home>

Zweck:

Pfadangabe (R-S)

Typ:

Standalone

Elternelement:

<docxworld-fetch-production-environment>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<docxworld-fetch-production-environment>**
      - **<production-bundles-home>**
    - **<exec>**
      - **<docxworld-fetch-production-environment>**
        - **<production-bundles-home>**
    - **<if>/<else>**
      - **<docxworld-fetch-production-environment>**
        - **<production-bundles-home>**
    - **<switch>**
      - **<case>**
        - **<docxworld-fetch-production-environment>**
          - **<production-bundles-home>**
    - **<while>**
      - **<docxworld-fetch-production-environment>**
        - **<production-bundles-home>**

Beachte

<production-bundles-home> ist ein obligatorisches Element von *<docxworld-fetch-production-environment>*, es muss also in jedem *<docxworld-fetch-production-environment>* vorkommen.

<merge-home>

> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit dem ***<merge-home>***-Element geben Sie Adresse des Verzeichnisses, in dem Binär-Paket, Produktions-Paket und Spooler-XML für diesen Job abgelegt werden, bevor dort aus dem Binär-Paket der ReportWriter aufgerufen wird, an. Die Dateien werden also vor der ReportWriter-Ausführung aus dem lokalen Repository (*<production-bundle-home>* und *<binary-bundle-home>*) oder, falls dort noch nicht vorhanden, vom entfernten R-S heraus, dorthin kopiert. Dies gilt nur, falls als ***runtime-environment*** des *<docxworld-fetch-production-environment>*-Elternelements "merged" gesetzt ist. Für jeden Job wird dann ein eigenes Verzeichnis zur Ausführung des ReportWriters angelegt, weshalb es sich anbietet, das Verzeichnis auf Basis von z.B. docxworld-client (eoms.client) und docxworld-Vertrag (eoms.procedure) dynamisch generieren zu lassen (siehe Beispiel).

Falls Sie als ***runtime-environment*** nicht "merged", sondern "shared" verwenden, müssen Sie *<shared-home>* setzen!

Attribute

Das ***<merge-home>***-Element besitzt keine Attribute.

Subelemente / Inhalt

Das ***<merge-home>***-Element besitzt keine Subelemente.

Der Inhalt eines ***<merge-home>***-Elements ist die Adresse des Verzeichnisses, in dem Binär-Paket, Produktions-Paket und Spooler-XML für diesen Job abgelegt werden, bevor dort aus dem Binär-Paket der ReportWriter aufgerufen wird. Die Dateien werden also vor der ReportWriter-Ausführung dorthin aus dem lokalen Repository oder, falls dort noch nicht vorhanden, vom entfernten R-S heraus, kopiert. Dies gilt nur, falls als ***runtime-environment*** des *<docxworld-fetch-production-environment>*-Elternelements "merged" gesetzt ist.

Variablenbindungen

Das ***<merge-home>***-Element besitzt keine Variablenbindungen.

Beispiel

```
<rcml>
<process id="ExampleProcess" name="example process">

...
<docxworld-fetch-production-environment id="ExampleRS"
runtime-environment="shared">
<docxworld-contract>${process['eoms.procedure']}</docxworld-contrac
t>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->

<binary-bundles-home>binary-bundles</binary-bundles-home>
<production-bundles-home>production-bundles</production-bundles-hom
e>

<runtime-home>${workdir.getAbsolutePath()}</runtime-home>

</docxworld-fetch-production-environment>
...

</process>
</rcml>
```

<merge-home>

Zweck:

Pfadangabe (R-S)

Typ:

Standalone

Elternelement:

<docxworld-fetch-production-environment>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<docxworld-fetch-production-environment>**
      - **<production-bundles-home>**
    - **<exec>**
      - **<docxworld-fetch-production-environment>**
        - **<merge-home>**
    - **<if>/<else>**
      - **<docxworld-fetch-production-environment>**
        - **<merge-home>**
    - **<switch>**
      - **<case>**
        - **<docxworld-fetch-production-environment>**
          - **<merge-home>**
    - **<while>**
      - **<docxworld-fetch-production-environment>**
        - **<merge-home>**

Beachte

<merge-home> muss nur gesetzt werden, wenn Sie als ***runtime-environment*** des Elternelements *<docxworld-fetch-production-environment>* "merged" verwenden.

<runtime-home>

Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit dem ***<runtime-home>***-Element geben Sie Adresse des Verzeichnisses, in dem der ReportWriter die Verarbeitung von Binär-Paket, Produktions-Paket und Spooler-XML vornehmen soll, vor. Im Gegensatz zu *<merge-home>* wird hier keine Verzeichnis für den Job, in den die Pakete und die XML kopiert werden, gesetzt, sondern der Spooler greift für den Job auf die Pakete, die in *<production-bundle-home>* und *<binary-bundle-home>* abgelegt sind, zu, ohne sie in das Verzeichnis kopieren zu müssen, in dem der ReportWriter ausgeführt wird. Der ReportWriter arbeitet also nicht mit lokalen Kopien bei jedem Job, sondern greift "aus der Entfernung" auf die Originaldateien zu. Dies spart bei großen Datenmengen enorm viel Rechenzeit, da die Dateien nicht jedes Mal kopiert werden müssen (sog. "Cache").

Falls Sie als ***runtime-environment*** nicht "shared", sondern "merge" verwenden, müssen Sie *<merge-home>* setzen!

"shared" kann bei manchen Auftrags-Systemen fremder Hersteller Probleme verursachen, da diese eventuell alle zu verarbeitenden Dateien in Arbeitsverzeichnis erwarten. Sollten solche Probleme auftreten, wechseln Sie zurück zu "merged".

Attribute

Das ***<runtime-home>***-Element besitzt keine Attribute.

Subelemente / Inhalt

Das ***<runtime-home>***-Element besitzt keine Subelemente.

Der Inhalt eines ***<runtime-home>***-Elements ist die Adresse des Verzeichnisses, aus dem heraus der ReportWriter aus dem Binär-Paket aufgerufen wird.

Variablenbindungen

Das ***<runtime-home>***-Element besitzt keine Variablenbindungen.

Beispiel

```
<rcml>
<process id="ExampleProcess" name="example process">

...
<docxworld-fetch-production-environment id="ExampleRS"
runtime-environment="shared">
<docxworld-contract>${process['eoms.procedure']}</docxworld-contrac
t>
<!-- <link-name>${process['eoms.procedure']}</link-name> -->

<binary-bundles-home>binary-bundles</binary-bundles-home>
<production-bundles-home>production-bundles</production-bundles-hom
e>

<runtime-home>${workdir.getAbsolutePath()}</runtime-home>

</docxworld-fetch-production-environment>
...

</process>
</rcml>
```

<runtime-home>

Zweck:

Pfadangabe (R-S)

Typ:

Standalone

Elternelement:

<docxworld-fetch-production-environment>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<docxworld-fetch-production-environment>**
      - **<runtime-home>**
    - **<exec>**
      - **<docxworld-fetch-production-environment>**
        - **<runtime-home>**
    - **<if>/<else>**
      - **<docxworld-fetch-production-environment>**
        - **<runtime-home>**
    - **<switch>**
      - **<case>**
        - **<docxworld-fetch-production-environment>**
          - **<runtime-home>**
    - **<while>**
      - **<docxworld-fetch-production-environment>**
        - **<runtime-home>**

Beachte

<merge-home> muss nur gesetzt werden, wenn Sie als ***runtime-environment*** des Elternelements *<docxworld-fetch-production-environment>* "merged" verwenden.

<eoms-input-query-data>

> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit ***<eoms-input-query-data>*** können Sie Ressourcen (z.B. Fehlerberichte) vom EOMS-Input-Server empfangen, wenn Sie zuvor mit *<eoms-input-submit>* Daten übertragen haben. Stellen Sie davor aber mit *<eoms-input-query-status>* sicher, dass zuvor keine Probleme bei der Übertragung und Abarbeitung des Jobs an den EOMS-Input-Server aufgetreten sind. Die empfangene Ressource können Sie dann z.B. zurück an das Auftrags-System schicken.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Elements, über die es im Code angesprochen werden kann.	beliebiger, regelkonformer Name.	—	✗
job	***STRING***	Die ID des *<eoms-input-submit>*, für das die Abfrage getätigt werden soll.	ID eines existierenden *<eoms-input-submit>*	—	✓
workdir	***FILEOBJECT***	Gibt das Arbeitsverzeichnis an, in dem die übertragenen Daten gespeichert werden.	Die ID eines *<workdir>*-Elements	—	✓
timeout	***STRING (long)***	Die Zeit in ms, die auf eine erfolgreiche Übertragung gewartet werden soll, bis der Vorgang mit einer Ausnahme abgebrochen wird.	Numerische Zeitangabe in ms.	1000 (ms)	✗

Subelemente / Inhalt

Das ***<eoms-input-query-data>***-Element besitzt keine Subelemente.

Variablenbindungen

Das **<eoms-input-query-data>**-Element ist vom Typ **REQUEST-OBJECT** und besitzt daher die gleichen Variablenbindungen wie das <eoms-input-submit>-Element. Allerdings beziehen sich die Werte hier auf das **<eoms-input-query-data>**-Element und die dadurch initiierte Übertragung.

Bindung	Beschreibung	Rückgabetyp
getId()	Gibt die **Id** des **<eoms-input-query-data>**-Elements zurück.	STRING
getCreated()	Gibt den Zeitpunkt, zu dem die Übertragung durch <eoms-input-submit> initialisiert wurde, zurück.	DATE
getExpires()	Gibt den Zeitpunkt, bis zu dem der <eoms-input-submit>-Vorgang gültig ist, zurück. Ab diesem Zeitpunkt wird der Vorgang verworfen, falls er noch nicht geendet hat.	DATE
getMessages()	Gibt die internen Messages zurück.	LIST OF STRINGS
getProperties()	Gibt die HashMap, die für den Übertragungsvorgang <eoms-input-submit> als **variables** übergeben wurde, zurück.	MAP
getStatus()	Gibt den Status des <eoms-input-submit> zurück. Kann entweder ACTIVE (aktiv) oder TERMINATED (fertig) sein.	REQUESTSTATUS
getBody()	Sollte nicht verwendet werden.	SUBMITJOBREQUEST

Beispiel

In folgendem Beispiel wird die vom Aufrags-System (Spooler) durch <fetchresource> empfangene Ressource direkt an den EOMS-Input-Server gesendet. Mit übergeben wird eine HashMap, die per JavaScript erstellt wurde und per **variables** übergeben wird. Anschließend wird mit **<eoms-input-query-status>** die weitere Verarbeitung so lange unterbrochen, bis der EOMS-Input-Server meldet, dass die Übertragung des <eoms-input-submit> mit der id="submitJob" erfolgreich war. Erst dann wird die Verarbeitung fortgesetzt. So wird sichergestellt, dass weiterer Code nur ausgeführt wird, wenn der Vorgang auch erfolgreich war und es wird verhindert, dass mit inkonsistenten oder nicht vorhandenen Daten gearbeitet wird. Nun können auch durch **<eoms-input-query-data>** Ergebnisdaten vom EOMS-Input-Server angefordert werden, die wir dann auch mit <upload> weiterschicken (z.B. zurück zum Auftrags-System).

```
<rcml>
<process id="ExampleProcess" name="example process">

<script language="JavaScript"><![CDATA[
var submitVariables = new java.util.HashMap();
submitVariables.put('EOMS_CLIENT', process['eoms.client']);
submitVariables.put('EOMS_PROCEDURE', process['eoms.procedure']);
submitVariables.put('EOMS_REFERENCE', process['eoms.reference']);
]]>
</script>

<workdir id="workdir" home="./WORK"/>
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}"/>

<eoms-input-submit id="submitJob" file="inputFile"
variables="${submitVariables}"/>

<eoms-input-query-status id="queryJobStatus" job="submitJob"/>

<eoms-input-query-data id="queryJobData" job="submitJob"
workdir="workdir" />

<upload file="workdir"
destination="${process['eoms.process.output']}" mask="*.*" />

</process>
</rcml>
```

<eoms-input-query-data>

Zweck:

Datenübertragung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Ja

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<eoms-input-query-data>**
    - **<exec>**
      - **<eoms-input-query-data>**
    - **<if>/<else>**
      - **<eoms-input-query-data>**
    - **<switch>**
      - **<case>**
        - **<eoms-input-query-data>**
    - **<while>**
      - **<eoms-input-query-data>**

Beachte

Falls die Übertragung nach Ablauf von ***timeout*** nicht erfolgreich war bricht die Verarbeitung mit einer Ausnahme (Exception) ab und nachfolgender Code wird nicht ausgeführt.

<eoms-input-query-status>

> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit dem ***<eoms-input-query-status>***-Element ist es möglich, die Weiterverarbeitung so lange zu unterbrechen, bis das Element eine positive Rückmeldung des EOMS-Input-Servers, dass die Daten, die durch ein vorheriges *<eoms-input-submit>* versendet wurden, im Server eingegangen sind (d.h. aber nicht, dass sie auch erfolgreich verarbeitet wurden). Das Element wartet maximal so lange wie in ***timeout*** angegeben. Ist ***timeout*** abgelaufen und ***<eoms-input-query-status>*** hat keine positive Rückmeldung vom Server erhalten, wirft das Element eine Ausnahme *(Exception)* und die Verarbeitung bricht an diesem Punkt mit einer Fehlermeldung ab (nachfolgender Code wird nicht ausgeführt). Da sich ***<eoms-input-query-status>*** auf ein *<eoms-input-submit>* -Element bezieht (der Bezug wird über ***job*** hergestellt), muss dieses auch vor ***<eoms-input-query-status>*** notiert werden.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Elements, über die es im Code angesprochen werden kann.	beliebiger, regelkonformer Name.	—	✗
job	***STRING***	Die ID des *<eoms-input-submit>*, für das die Abfrage getätigt werden soll.	ID eines existierenden *<eoms-input-submit>*	—	✓
timeout	***STRING (long)***	Die Zeit in ms, die auf eine positive Rückmeldung des EOMS-Input-Servers gewartet werden soll.	Numerische Zeitangabe in ms.	1000 (ms)	✗

Subelemente / Inhalt

Das ***<eoms-input-query-status>***-Element besitzt keine Subelemente.

Variablenbindungen

Das ***<eoms-input-query-status>***-Element ist vom Typ ***REQUEST-OBJECT*** und besitzt daher die gleichen Variablenbindungen wie das *<eoms-input-submit>*-Element. Dementsprechend beziehen sich auch die meisten Variablenbindungen auf das *<eoms-input-submit>*-Element, dass durch ***<eoms-input-query-data>*** abgefragt werden soll:

Bindung	Beschreibung	Rückgabetyp
getId()	Gibt die ***Id*** des ***<eoms-input-query-status>***-Elements zurück.	*STRING*
getCreated()	Gibt den Zeitpunkt, zu dem die Übertragung durch *<eoms-input-submit>* initialisiert wurde, zurück.	*DATE*
getExpires()	Gibt den Zeitpunkt, bis zu dem der *<eoms-input-submit>*-Vorgang gültig ist, zurück. Ab diesem Zeitpunkt wird der Vorgang verworfen, falls er noch nicht geendet hat.	*DATE*
getMessages()	Gibt die internen Messages zurück.	*LIST OF STRINGS*
getProperties()	Gibt die HashMap, die für den Übertragungsvorgang *<eoms-input-submit>* als ***variables*** übergeben wurde, zurück.	*MAP*
getStatus()	Gibt den Status des *<eoms-input-submit>* zurück. Kann entweder ACTIVE (aktiv) oder TERMINATED (fertig) sein.	*REQUESTSTATUS*
getBody()	Sollte nicht verwendet werden.	*SUBMITJOBREQUEST*

Beispiel

In folgendem Beispiel wird die vom Aufrags-System (Spooler) durch *<fetchresource>* empfangene Ressource direkt an den EOMS-Input-Server gesendet. Mit übergeben wird eine HashMap, die per JavaScript erstellt wurde und per ***variables*** übergeben wird. Anschließend wird mit ***<eoms-input-query-status>*** die weitere Verarbeitung so lange unterbrochen, bis der EOMS-Input-Server meldet, dass die Übertragung des *<eoms-input-submit>* mit der id="submitJob" erfolgreich war. Erst dann wird die Verarbeitung fortgesetzt. So wird sichergestellt, dass weiterer Code nur ausgeführt wird, wenn der Vorgang auch erfolgreich war und es wird verhindert, dass mit inkonsistenten oder nicht vorhandenen Daten gearbeitet wird.

```
<rcml>
<process id="ExampleProcess" name="example process">

<script language="JavaScript"><![CDATA[
var submitVariables = new java.util.HashMap();
submitVariables.put('EOMS_CLIENT', process['eoms.client']);
submitVariables.put('EOMS_PROCEDURE', process['eoms.procedure']);
submitVariables.put('EOMS_REFERENCE', process['eoms.reference']);
]]>
</script>

<workdir id="workdir" home="./WORK"/>
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}"/>

<eoms-input-submit id="submitJob" file="inputFile"
variables="${submitVariables}"/>

<eoms-input-query-status id="queryJobStatus" job="submitJob"/>

<!-- Hier kann man jetzt sicher sein, dass die Daten erfolgreich
übertragen wurden. -->
</process>
</rcml>
```

<eoms-input-query-status>

Zweck:

Statusabfrage

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Ja

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<eoms-input-query-status>**
    - **<exec>**
      - **<eoms-input-query-status>**
  - **<if>/<else>**
    - **<eoms-input-query-status>**
  - **<switch>**
    - **<case>**
      - **<eoms-input-query-status>**
  - **<while>**
    - **<eoms-input-query-status>**

Beachte

Falls <eoms-input-query-status> nach Ablauf von timeout keine positive Antwort vom EOMS-Input-Server erhält bricht die Verarbeitung mit einer Ausnahme (Exception) ab und nachfolgender Code wird nicht ausgeführt.

<eoms-input-submit>

Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit dem ***<eoms-input-submit>***-Element können Daten (Dateien / Verzeichnisse) an die EOMS-Input-Schnittstelle von docxworld geschickt (eingeliefert) werden. Optional können Sie ein Array (in JavaScript erstellte HashMap) mit Informationen übergeben, dass mit versendet werden soll. Besonders wichtig ist das Attribut **id**, denn über diese id können Sie sich später z.B. in einem *<eoms-input-query-data>* oder *<eoms-input-query-status>* auf diesen Sendevorgang beziehen.

Die Verbindungsinformationen, wohin sich ***<eoms-input-submit>*** verbinden soll, sind in der *eoms.invoker.client.properties* festgelegt.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Elements, über die es im Code angesprochen werden kann.	beliebiger, regelkonformer Name.	—	✗
file	***FILEOBJECT***	Das *FILEOBJECT* (Datei / Verzeichnis), das übertragen werden soll.	Ein existierendes ***FILEOBJECT***.	—	✓
variables	***MAP***	Eine Map (Eigenschaft - Wert), die bei der Übertragung berücksichtigt und mitgesendet wird. Diese Map kann z.B. Informationen wie docxworld-Client oder docxworld-Vertrag enthalten und muss per Skript erstellt werden *(siehe Beispiel)*.	HashMap (JavaScript).	—	✗
timeout	***STRING (long)***	Die Zeit in ms, die auf eine erfolgreiche Übertragung gewartet werden soll, bis der Vorgang mit einer Ausnahme abgebrochen wird.	Numerische Zeitangabe in ms.	1000 (ms)	✗

Subelemente / Inhalt

Das ***<eoms-input-submit>***-Element besitzt keine Subelemente.

Variablenbindungen

Das ***<eoms-input-submit>***-Element ist vom Typ ***REQUEST-OBJECT***. Dieser Typ besitzt folgende Variablenbindungen:

Bindung	Beschreibung	Rückgabetyp
getId()	Gibt die ***Id*** des ***<eoms-input-submit>***-Elements zurück.	*STRING*
getCreated()	Gibt den Zeitpunkt, zu dem die Übertragung initialisiert wurde, zurück.	*DATE*
getExpires()	Gibt den Zeitpunkt, bis zu dem der Vorgang gültig ist, zurück. Ab diesem Zeitpunkt wird der Vorgang verworfen, falls er noch nicht geendet hat.	*DATE*
getMessages()	Gibt die internen Messages zurück.	*LIST OF STRINGS*
getProperties()	Gibt die HashMap, die für den Übertragungsvorgang als ***variables*** übergeben wurde, zurück.	*MAP*
getStatus()	Gibt den Status des ***<eoms-input-submit>*** zurück. Kann entweder ACTIVE (aktiv) oder TERMINATED (fertig) sein.	*REQUESTSTATUS*
getBody()	Sollte nicht verwendet werden.	*SUBMITJOBREQUEST*

Beispiel

In folgendem Beispiel wird die vom Aufrags-System (Spooler) durch *<fetchresource>* empfangene Ressource direkt an den EOMS-Input-Server gesendet. Mit übergeben wird eine HashMap, die per JavaScript erstellt wurde und per ***variables*** übergeben wird. Das Beispiel verzichtet auf eine Nachbehandlung des ***<eoms-input-submit>***-Vorgangs. Beispiele dazu finden Sie bei *<eoms-input-query-status>* und *<eoms-input-query-data>*.

```
<rcml>
<process id="ExampleProcess" name="example process">

<script language="JavaScript"><![CDATA[
var submitVariables = new java.util.HashMap();
submitVariables.put('EOMS_CLIENT', process['eoms.client']);
submitVariables.put('EOMS_PROCEDURE', process['eoms.procedure']);
submitVariables.put('EOMS_REFERENCE', process['eoms.reference']);
]]>
</script>

<workdir id="workdir" home="./WORK"/>
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />

<eoms-input-submit id="submitJob" file="inputFile"
variables="${submitVariables}"/>

<!-- Hier kann mit <eoms-input-query-status> und
<eoms-input-query-data> Status und Daten abgefragt werden... -->
</process>
</rcml>
```

<eoms-input-submit>

Zweck:

Datenübertragung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Ja

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<eoms-input-submit>**
    - **<exec>**
      - **<eoms-input-submit>**
    - **<if>/<else>**
      - **<eoms-input-submit>**
    - **<switch>**
      - **<case>**
        - **<eoms-input-submit>**
    - **<while>**
      - **<eoms-input-submit>**

Beachte

Falls die Übertragung durch ***<eoms-input-submit>*** fehlschlägt, wird das Element eine Ausnahme (Exception) und die Verarbeitung wird nicht fortgesetzt. Folgender Code wird nicht ausgeführt.

<error>

Semantik

Mit dem ***<error>***-Element lässt sich eine rudimentäre Fehlerbehandlung implementieren. Das ***<error>***-Element bewirkt den sofortigen Abbruch der Prozessausführung. Die Verarbeitung stoppt also bei dem ***<error>***-Element und nachfolgender Code wird nicht ausgeführt. Gleichzeitig können Sie als Attribut eine Message angeben, die an das EOMS-Core gesendet wird (Fehlermeldung). Verwenden Sie das ***<error>***-Element z.B. in Verbindung mit dem *<if>*-Element, um auf verschiedene Ausnahmen unterschiedlich zu reagieren (siehe auch Beispiel).

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
message	***STRING***	Ermöglicht es, eine Fehlermeldung in Form einer *<message>* an das EOMS-Core zu senden.	beliebige Zeichenkette	—	✗

Subelemente / Inhalt

Das ***<error>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<error>***-Element besitzt keine Variablenbindungen.

Beispiel

Im folgenden Beispiel wird der Return-Code eines einfachen copy-Prozesses abgeglichen (nicht durch *<result>*, sondern durch testReturnCode()). Passend dazu wird mit ***<error>*** jeweils eine angepasste Fehlermeldung gesendet. Die Verarbeitung wird nur fortgesetzt, wenn der Fall return-code = false (=0) eintritt, wenn also keine Fehler aufgetreten sind. Dieses Beispiel verwendet das *<exec>*-, *<commandline>*-, *<workdir>*- und *<if>*-Element.

```
<rcml>
<process id="ExampleProcess" name="example process">

<workdir id="workdir" home="./WORK" />

<!-- Ausführung -->
<exec id="ExampleExec" workdir="workdir">
<commandline processor="velocity">cmd move c:\*.txt
c:\TXTs</commandline>
<!-- <commandline> setzt return-code von <exec> auf 0, falls
erfolgreich, sonst != 0. -->
</exec>

<!-- Falls ein Fehler aufgetreten ist: Fehlermeldung an das
EOMS-Core und Abbruch der Verarbeitung. -->
<if condition="${ExampleExec.testReturnCode(true)}">
<error message="${'Beim Aufruf des Befehls [cmd] Fehler: '
+ ExampleExec.getReturnCode() + '... der Prozess wird
abgebrochen.'}"/>
</if>

<!-- Falls kein Fehler aufgetreten ist geht es normal weiter. -->
...
</process>
</rcml>
```

<error>

Zweck:

Fehlerbehandlung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<error>**
    - **<exec>**
      - **<error>**
    - **<if>/<else>**
      - **<error>**
    - **<switch>**
      - **<case>**
        - **<error>**
    - **<while>**
      - **<error>**

Beachte

Das ***<error>***-Element bricht die Verarbeitung der RCML sofort ab. Nachfolgender Code wird nicht ausgeführt!

<exec>

Semantik

<exec> dient zur Ausführung von Kommandos (Prozessen) auf Betriebssystemebene. Dabei wird der Befehl nicht direkt in ***<exec>*** deklariert, sondern ***<exec>*** enthält als Container unter anderem den Aufrufbefehl in Form eines *<commandline>*-Elements. Das ***<exec>***-Element dient lediglich als Container für seine Subelemente. Deshalb muss ein ***<exec>***-Element immer ein *<commandline>*-Element enthalten. Über *<param>*-Elemente werden dem *<commandline>*-Element Parameter übergeben. Mit *<result>* lässt sich überprüfen, ob die Ausführung erfolgreich war.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Elements, über die es im Code angesprochen werden kann.	beliebiger, regelkonformer Name.	—	✗
process	***STRING***	Hier kann zur besseren Übersichtlichkeit optional ein Name für den ***<exec>***-Process angegeben werden.	beliebiger, regelkonformer Name.	—	✗
workdir	***STRING***	Gibt das Arbeitsverzeichnis an, in dem der Prozess gestartet werden soll.	Die ID eines *<workdir>*-Elements.	—	✓
override	***BOOLEAN***	Falls es in dem RCML-Prozess vor diesem ***<exec>*** bereits ein ***<exec>*** mit der selben ID gab, werden, falls override eingeschalten wird (true), das alte ***<exec>*** im Speicher überschrieben. Nachfolgend kann dann über diese ID nur noch auf das überschreibende ***<exec>*** zugegriffen werden.	true : false	false	−

Subelemente / Inhalt

Das **<exec>**-Element besitzt 3 Subelemente, die nur innerhalb eines **<exec>** notiert werden dürfen. Davon ist nur das *<commandline>*-Element ist obligatorisch.

- Das **<param>**-Element zur Übergabe von Parametern an ein nachfolgendes *<commandline>*-Element. Ein **<exec>**-Element ka 0 bis beliebig viele *<param>*-Elemente enthalten.
- Das **<commandline>**-Element dient zur Ausführung eines Befehls. Ein **<exec>**-Element **muss** min. ein *<commandline>*-Eleme enthalten.
- Das **<result>**-Element ermöglicht es, den Return-Code (Rückgabewert) des Kommandoaufrufs abzufragen und die Verarbeitur bei einem Fehler zu stoppen. Ein **<exec>**-Element kann ein *<result>*-Element beinhalten.

In einem **<exec>**-Element dürfen des Weiteren alle RCML-Elemente notiert werden, außer: *<rcml>, <process>.*

Es ist auch erlaubt, **<exec>**-Elemente zu schachteln (**<exec>** innerhalb von **<exec>** zu notieren).

Variablenbindungen

Das **<exec>**-Element ist vom Typ ***RUNTIME-OBJECT***. Dieser Typ besitzt folgende Variablenbindungen:

Name	Beschreibung	Rückgabetyp
getWorkdir()	Gibt das Arbeitsverzeichnis zurück.	*FILEOBJECT*
getReturnCode()	Gibt den Rückgabewert des **<exec>** zur ück.	*INTEGER*
testReturnCode(STRING param)	Vergleicht den Rückgabewert des **<exe c>** mit **param**. Sind die Werte identisch, gibt die Funktion true zurück, andernfalls false.	*BOOLEAN*
getExecutionTime()	Gibt die Dauer der Ausführung in ms zurück.	*LONG*
getExitValue()	Identisch zu ***getReturnCode()***.	*INTEGER*

Auf **FILEOBJECT** (*id.getWorkdir()*) sind außerdem folgende Variablenbindungen anwendbar:

Name	Beschreibung	Rückgabetyp
getName()	Gibt den Namen der Datei zurück.	*STRING*
getAbsolutePath()	Gibt den absoluten Pfad zur Datei zurück.	*STRING*

Beispiel

Im folgenden Beispiel wird ein einfacher copy-Prozesses ausgeführt. Die Verarbeitung soll durch *<result>* abgebrochen werden, falls die Bearbeitung nicht erfolgreich beendet wurde (irgendein Fehler oder irgendeine Warnung auftrat).

```
<rcml>
<process id="ExampleProcess" name="example process">

<exec id="copyProcess" workdir="workdir">

<workdir id="workdir" home="./WORK" />

<!-- Setze 2 Parameter: Pfad zur zu kopierenden Datei und Zielpfad.
-->
<param name="infile"
value="${inputFile.getFile().getAbsolutePath()}"/>
<param name="outfile" value="${process['copy.output']}"/>

<!-- Datei kopieren. -->
<commandline processor="velocity">cmd /C copy $infile
$outfile</commandline>

  <!-- Nur wenn return-code = 0 ist (erfolgreich), Verarbeitung
fortsetzen. -->
<result return-code="0" />

</exec>

</process>
</rcml>
```

<exec>

Zweck:

Befehlsausführung

Typ:

Top-Level

Elternelement:

Top-Level-Elemente

Subelemente:

Ja

Variablenbindungen:

Ja

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<exec>**
      - <param>
      - <commandline>
      - <result>
    - **<exec>**
      - **<exec>**
        - <param>
        - <commandline>
        - <result>
    - **<if>/<else>**
      - **<exec>**
        - <param>
        - <commandline>
        - <result>
    - **<switch>**
      - **<case>**
        - **<exec>**
          - <param>
          - <commandline>
          - <result>
    - **<while>**
      - **<exec>**
        - <param>
        - <commandline>
        - <result>

Beachte

—

<param>

Semantik

Das ***<param>***-Element macht innerhalb eines *<exec>*-Elements Parameter für die Kommandoausführung durch *<commandline>*-verfügbar. ***<param>***-Elemente müssen innerhalb eines *<exec>*-Elements stehen und vor dem *<commandline>*-Element, damit die Parameter im *<commandline>*-Element genutzt werden können. Als Attribute werden der Name des Parameters und sein neuer Wert übergeben. Der Name des Parameters ist dabei frei wählbar (innerhalb der Regeln zur Benennung von Variablen). Als Wert können neben statischen Werten auch berechnete Werte zugewiesen werden. In einem nachfolgenden *<commandline>*-Element ist der Parameter mit seinem Wert dann über seinen Namen ansprechbar (siehe *<commandline>*). Alle Objekte, die innerhalb von *<commandline>* angesprochen werden sollen, müssen zuvor als *<commandline>* deklariert sein! Soll z.B. innerhalb von *<commandline>* auf die Ressource eoms.process.input zugegriffen werden, so muss diese einem ***<param>*** zugewiesen und dann per $***name*** darauf zugegriffen werden (siehe Beispiel).

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
name	***STRING***	Gibt den Namen des Parameters an. Kann beliebig gewählt werden.	beliebige, regelkonforme Zeichenkette.	—	✓
value	***OBJECT***	Gibt den Wert an, den der Parameter enthalten soll.	beliebiges Objekt (alle Datentypen gültig, auch z.B. Files).	—	✓

Subelemente / Inhalt

Das <***param***>-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das <***param***>-Element besitzt keine Variablenbindungen. Der Zugriff auf das Element erfolgt direkt über ***name***, innerhalb von *<commandline>* z.B. durch ***$name*** (siehe *<commandline>*).

Beispiel

In *<commandline>*-Elementen können ausschließlich ***<param>***'s referenziert werden. Deshalb muss jegliches Objekt, dass im Befehl verwendet werden soll, zuvor einem ***<param>*** zugewiesen werden. Dies gilt auch für Variablen aus *<update-variable>* und anderen dynamisch generierten Objekten im Speicher. Beispiel *(verwendet das <workdir>-Element)*:

```
<rcml>
<process id="ExampleProcess" name="example process">

<workdir id="workdir" home="./WORK" />
<update-variable name="command" value="cmd mkdir C:/newfolder" />

<exec id="copyProcess" workdir="workdir">
<!-- Ich will in meinem commandline-Befehl die Variable command
benutzen. Folgendes ist nicht erlaubt:
<commandline processor="velocity">command</commandline> -->

<!-- stattdessen muss die Resource zuvor einem <param> zugewiesen
werden: -->
<param name="mycommand" value="${command}"/>

<!-- Befehlsausführung -->
<commandline processor="velocity">$mycommand</commandline>

</exec>

</process>
</rcml>
```

Noch einmal ein vollständiges Beispiel, das 2 Parameter setzt und sie danach im <*commandline*>-Befehl verwendet. Das Beispiel verwendet das <*workdir*>-Element.

```
<rcml>
<process id="ExampleProcess" name="example process">

<workdir id="workdir" home="./WORK" />

<exec id="copyProcess" workdir="workdir">

<!-- Setzt 2 Parameter und übergibt als Werte berechnete Ausdrücke:
Parameter 1:
Der absolute Pfad zur InputRessource. Parameter 2: Kopierziel. -->

<param name="infile"
value="${inputFile.getFile().getAbsolutePath()}"/>
<param name="outfile" value="${process['copy.output']}"/>

<!-- Kopiert Datei $infile nach $outfile. -->
<commandline processor="velocity">cmd /C copy $infile
$outfile</commandline>

</exec>

</process>
</rcml>
```

<param>

Zweck:

Parameterübergabe

Typ:

Subelement

Elternelement:

<exec>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<exec>**
      - **<param>**

Beachte

<param>-Elemente müssen in *<exec>*-Elementen vor dem *<commandline>*-Element stehen!

<commandline>

Semantik

Das ***<commandline>***-Element dient zur Ausführung von Befehlen auf Betriebssystemebene. Einem ***<commandline>***-Element können manuell gesetzte Parameter durch vorangehende *<param>*-Elemente verfügbar gemacht werden. Der auszuführende Befehl wird dem ***<commandline>***-Element als dessen Inhalt übergeben. Das ***<commandline>***-Element muss immer innerhalb eines *<exec>*-Elements notiert werden, ein *<exec>*-Element muss ein ***<commandline>***-Element enthalten. Stellen Sie dem Befehl "cmd " voran, um den Befehl an das Kommandozeilentool von Windows zu übergeben und es von diesem ausführen zu lassen. In Befehlen selbst kann nicht direkt auf RCML-Elemente per ID zugegriffen werden (so ist z.B. *<commandline ...> cmd $resource.getFile() </commandline>* ist nicht möglich!). Deshalb müssen Sie solche Zugriffe vorher in einem *<param>* speichern und dann darauf zugreifen (siehe Beispiel von *<exec>*).

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
processor	***STRING***	Gibt den Interpreter an, der für die Interpretation des Befehls geladen werden soll.	momentan nur *velocity*.	velocity	✓

Subelemente / Inhalt

Das ***<commandline>***-Element besitzt keine Subelemente.

Der Inhalt eines ***<commandline>***-Elements ist der Befehl, der ausgeführt werden soll. Die Syntax des Befehls hängt dabei vom Betriebssystem und dem gewählten ***processor*** ab.

Variablenbindungen

Das ***<commandline>***-Element besitzt keine Variablenbindungen. Informationen über die Befehlsausführung werden über die Variablenbindungen des *<exec>*-Elements abgerufen. Ob ein ***<commandline>***-Aufruf erfolgreich war, kann auch durch das *<result>*-Element in Erfahrung gebracht werden.

Beispiel

Im folgenden Beispiel wird ein einfach Befehlsaufruf ohne Parameter übergeben. Dieser verschiebt alle .txt-Dateien unter C:/ in ein neues Verzeichnis C:/TXTs. Das Beispiel verwendet das *<workdir>*-Element.

```
<rcml>
<process id="ExampleProcess" name="example process">

<workdir id="workdir" home="./WORK" />
<exec id="ExampleExec" workdir="workdir">
<commandline processor="velocity">cmd move c:\*.txt
c:\TXTs</commandline>
</exec>

</process>
</rcml>
```

<commandline>

Zweck:

Befehlsausführung

Typ:

Subelement

Elternelement:

<exec>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<exec>**
      - **<commandline>**

Beachte

Eventuelle *<param>*-Elemente müssen in *<exec>*-Elementen vor dem ***<commandline>***-Element stehen. Das *<result>*-Element darf erst nach dem ***<commandline>***-Element stehen!

<result>

How-To

Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Durch das ***<result>***-Element kann der Return-Code des umgebenen <*exec*>-Elements abgeglichen werden. Damit ist es möglich, die Verarbeitung abzubrechen, falls ein bestimmter Fehlercode bei der Ausführung von <*commandline*> aufgetreten ist.

Das <*commandline*>-Element signalisiert nach der Ausführung durch einen sog. Return-Code, ob bei der Verarbeitung des Befehls Fehler oder Warnungen aufgetreten sind. Ist der Return-Code ungleich 0 ist die Bearbeitung nicht ohne Fehler / Warnungen abgelaufen. Der Return-Code ist dabei eine Eigenschaft des umgebenden <*exec*>-Elements. Das <*commandline*>-Element setzt also den Return-Code des <*exec*>-Elements. Dieser Return-Code kann durch ***<result>*** mit einem angegebenen Wert abgeglichen werden, hierbei sind auch logische Ausdrücke erlaubt. Stimmt der Return-Code des <*exec*>-Elements mit dem übergebenen Wert nicht überein, ist also bei der Abarbeitung von <*commandline*> ein unerwarteter Fehler / eine unerwartete Warnung aufgetreten, bricht ***<result>*** an dieser Stelle sofort die Verarbeitung der RCML ab. Nachfolgender Code wird nicht ausgeführt! Ergibt der Abgleich eine Übereinstimmung wird die Verarbeitung ohne Effekt fortgesetzt.

Das ***<result>***-Element kann nur innerhalb des <*exec*>-Elements notiert werden. Nutzen Sie für eine allgemeinere Fehlerbehandlung <*error*>.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
return-code	***INTEGER***	Ermöglicht es, einen Fehlercode oder einen logischen Ausdruck anzugeben. ***<result>*** vergleicht den angegebenen Wert mit dem Return-Code des <*exec*>-Elements, der durch das vorangegangene <*commandline*>-Element gesetzt wurde. Stimmt der übergebene Ausdruck mit dem Return-Code nicht überein, wird die Verarbeitung sofort abgebrochen.	Fehlercodes und logische Ausdrücke. Mehr unter Beispiel	—	✓

Subelemente / Inhalt

Das ***<result>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<result>***-Element besitzt keine Variablenbindungen.

Beispiel

Einige Beispielwerte für return-code:

- 400
- 400 | 401 (400 oder 401).
- true (alles außer 0, also alles außer erfolgreich beendet).

Im folgenden Beispiel wird der Return-Code eines einfach copy-Prozesses abgeglichen. Die Verarbeitung soll abgebrochen werden, falls die Bearbeitung nicht erfolgreich beendet wurde (irgendein Fehler oder irgendeine Warnung auftrat).

```
<rcml>
<process id="ExampleProcess" name="example process">

<exec id="copyProcess" workdir="workdir">

<workdir id="workdir" home="./WORK" />

<!-- Setze 2 Parameter: Pfad zur zu kopierenden Datei und Zielpfad.
-->
<param name="infile"
value="${inputFile.getFile().getAbsolutePath()}"/>
<param name="outfile" value="${process['copy.output']}"/>

<!-- Datei kopieren. -->
<commandline processor="velocity">cmd /C copy $infile
$outfile</commandline>

  <!-- Nur wenn return-code = 0 ist (erfolgreich), Verarbeitung
fortsetzen. -->
<result return-code="0" />

</exec>

</process>
</rcml>
```

<result>

Zweck:

Fehlerbehandlung

Typ:

Subelement

Elternelement:

<exec>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<exec>**
      - **<result>**

Beachte

Das ***<result>***-Element muss nach dem <*commandline*>-Element stehen!

Bei fehlender Übereinstimmung wird die Verarbeitung sofort abgebrochen und nachfolgender Code nicht ausgeführt!

<fetch-production-bundle>

Dieses Element ist veraltet und wird in der aktuellen Version nicht mehr unterstützt. Verwenden Sie stattdessen **<docxworld-fetch-production-environment>**.

<fetch-production-environment>

Entspricht exakt ***<docxworld-fetch-production-environment>***.

<fetchresource>

Semantik

<fetchresource> bindet eine Ressource an ein Objekt im Speicher, so dass dieses über die angegebene ***id*** im Code ansprechbar und verwendbar ist. Üblicherweise wird die Ressource vom Auftrags-System empfangen, Sie können aber Ressource von jeder beliebigen, erreichbaren Adresse anfordern (lokal oder remote). ***<fetchresource>*** stellt damit eine Schnittstelle zwischen Dateien und Objekten in RCML her. Selbstverständlich müssen Sie nicht zwingend mit einem ***<fetchresource>*** in einem *<process>* arbeit en, Worker können auch Prozesse völlig ohne Anbindung an externe Systeme ausführen.

Beachten Sie, dass Sie über Variablenbindungen wie .getFile() auf die Datei selbst, die durch ein ***<fetchresource>***-Objekt repräsentiert wird, zugreifen müssen.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Elements, über die es im Code angesprochen werden kann.	beliebiger, regelkonformer Name.	—	✗
resource	***STRING***	Die zu bindende Ressource.	—	—	✓
clear-on-shutdown	***BOOLEAN***	Gibt an, ob das Verzeichnis nach Beendigung des Prozesses automatisch gelöscht werden soll (true) oder nicht (false) oder ob es nur gelöscht werden soll, wenn kein Fehler aufgetreten ist (keep-on-error). ⚠ Im Produktiv einsatz sollte hier true gesetzt werden.	"true" : "false" : "keep-on-error"	false	✗

Subelemente / Inhalt

Das ***<fetchresource>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<fetchresource>***-Element ist vom Typ ***FETCHED-RESOURCE***. Dieser Typ besitzt folgende Variablenbindungen:

Bindung	Beschreibung	Rückgabetyp
getFile()	Gibt die durch ***<fetchresource>*** gebundene Datei als Dateiobjekt zurück (siehe unten *FILEOBJECT*).	*FILEOBJECT*
getUrl()	Gibt den URL zur Ressource zurück.	*STRING*
getLocator()	Gibt die genaue Adresse des Objekts zurück.	*STRING*
isShared()	Gibt an, ob die Ressource shared oder merged abgelegt wird *(siehe auch hier)*.	*BOOLEAN*

Auf **FILEOBJECT** (*id.getFile()*) sind außerdem folgende Variablenbindungen anwendbar:

Name	Beschreibung	Rückgabetyp
getName()	Gibt den Namen der Datei zurück.	*STRING*
getAbsolutePath()	Gibt den absoluten Pfad zur Datei zurück.	*STRING*

Beispiel

In folgendem Beispiel wird mit ***<fetchresource>*** die vom Spooler empfangene Datei an die id = "inputFile" gebunden. Über die Variablenbindungen .getFile().getAbsolutePath() wird der Pfad zur Datei in einem *<param>* gespeichert und dann verwendet. Anschließend wird die Datei umbenannt und direkt wieder an den Spooler zurückgeschickt. Zuletzt wird die Ressource mit *<releaseresource>* manuell freigegeben. Das Beispiel verwendet das *<workdir>*-, *<exec>*-, *<param>*-, *<commandline>*-, *<result>*- und *<releaseresource>*-Element.

```
<rcml>
<process id="ExampleProcess" name="example process">

<!-- Arbeitsverzeichnis für den Prozess setzen. -->
<workdir id="workdir" home="./WORK" />

<!-- Vom Spooler Ressource empfangen und an die id "inputFile"
binden. -->
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />

<exec id="RenameProcess" workdir="workdir">

<!-- Parameter anlegen: Pfad zur Ressource "inputFile". Dies muss
über <param> geschehen! -->
<param name="infile"
value="${inputFile.getFile().getAbsolutePath()}" />

<!-- Parameter anlegen: Der neue Pfad mit neuem Namen. Dies muss
über <param> geschehen! -->
<param name="outfile" value="${workdir.getAbsolutePath() +
'/cool_new_file'}" />

<!-- Ressource umbenennen in "cool_new_file"... -->
<commandline processor="velocity">cmd move $infile $outfile
</commandline>

<!-- Bei Fehlern abbrechen. -->
<result return-code="0" />

</exec>

<!-- Umbenannte Datei samt komplettem Arbeitsverzeichnis an Spooler
schicken. -->
<upload file="workdir"
destination="${process['eoms.process.output']}" />

<!-- Ressource manuell wieder freigeben. -->
<releaseresource resource="inputFile" />

</process>
</rcml>
```

<fetchresource>

Zweck:

Ressourcenbindung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Ja

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<fetchresource>**
    - **<exec>**
      - **<fetchresource>**
    - **<if>/<else>**
      - **<fetchresource>**
    - **<switch>**
      - **<case>**
        - **<fetchresource>**
    - **<while>**
      - **<fetchresource>**

Beachte

Mit *<releaseresource>* kann die Ressource wieder freigegeben (zur Löschung markiert) werden.

<if> - <else>

> **How-To**
>
> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit ***<if>*** und ***<else>*** ist es wie mit *<switch>* möglich, bedingte Anweisungen zu formulieren, die nur ausgeführt werden, falls eine bestimmte Bedingung eintritt. Im Gegensatz zu *<switch>* wird hier aber immer nur eine Bedingung geprüft (mit ***<if>***). Ist die Bedingung wahr, wird der Inhalt von ***<if>*** ausgeführt. Ist die Bedingung unwahr, wird der Inhalt des nachfolgenden ***<else>*** ausgeführt. ***<else>*** ist hierbei optional, ***<if>*** kann also auch alleine verwendet werden. ***<else>*** kann aber immer nur nach einem ***<if>*** stehen (auf das es sich dann bezieht).

Attribute

Das ***<if>***-Element besitzt folgende Attribute:

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
condition	***STRING***	Die boolsche Bedingung, die geprüft werden soll. Nur wenn die Bedingung wahr ist, wird der Inhalt des ***<if>*** ausgeführt. Hier können auch berechnete Werte stehen.	Beliebiger boolscher Ausdruck	—	✓

Das ***<else>***-Element besitzt keine Attribute.

Subelemente / Inhalt

In einem ***<if>***-Element dürfen alle RCML-Elemente notiert werden, außer: *<rcml>, <process>.*

Als Inhalt wird der Code notiert, der ausgeführt werden soll, falls die ***condition*** wahr ist.

Das ***<else>***-Element dürfen alle RCML-Elemente notiert werden, außer: *<rcml>, <process>.*

Als Inhalt wird der Code notiert, der ausgeführt werden soll, falls die ***condition*** aus dem vorangehenden ***<if>*** unwahr ist.

Es ist auch erlaubt, ***<if>/<else>***-Elemente zu schachteln (***<if>/<else>*** innerhalb von ***<if>/<else>*** zu notieren).

Variablenbindungen

Das ***<if>***-Element besitzt keine Variablenbindungen.

Das **<else>**-Element besitzt keine Variablenbindungen.

Beispiel

Das folgende Beispiel orientiert sich am Beispiel von *<switch>* und implementiert eine Fehlerbehandlung mit **<if> - <else>**. Treten beim Ausführen der *<commandline>* keine Fehler auf, ist die Bedingung von ***<if>*** wahr und dessen Inhalt wird ausgeführt. In allen anderen Fällen wird der Inhalt von ***<else>*** ausgeführt. Weitere Beispiele finden Sie im HowTo - Kontrollstrukturen.

```
<rcml>
<process id="ExampleProcess" name="example process">

<exec id="ExampleExec" workdir="workdir">
<commandline processor="velocity">cmd echo Wow, what a simple
exec!</commandline>
</exec>

<if condition="${ExampleExec.testReturnCode('0')}"> <!-- Falls
ReturnCode = 0 wird der Inhalt von <if> ausgeführt. -->
<message text="Befehl erfolgreich ausgeführt." />
</if>
<else> <!-- Ansonsten (bei allem außer ReturnCode = 0) wird der
Inhalt von <else> ausgeführt. -->
<message text="Befehlsausführung fehlgeschlagen." />
</else>

</process>
</rcml>
```

<if> / <else>

Zweck:

Bedingte Anweisung

Typ:

Top-Level

Elternelement:

Top-Level-Elemente

Subelemente:

Ja

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<if>/<else>**
    - **<exec>**
      - **<if>/<else>**
    - **<if>/<else>**
      - **<if>/<else>**
    - **<switch>**
      - **<case>**
        - **<if>/<else>**
    - **<while>**
      - **<if>/<else>**

Beachte

Sie können **<if>** - **<else>**-Konstrukte auch schachteln. D.h. **<if>/<else>**-Elemente dürfen auch **<if>/<else>**-El
emente beinhalten. **<if>/<else>**-Elemente dürfen alle RCML-Elemente (außer <*rcml*>, <*process*>) enthalten.

<message>

Semantik

Das ***<message>***-Element ermöglicht es, Nachrichten an das EOMS-Core zu übermitteln. Diese Nachrichten sind dann sowohl für das Auftrags-System als auch für den EOMS-Core-Monitor auslesbar (für EOMS-Core-Monitor siehe hier).

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
text	***STRING***	Der Text, der an das EOMS-Core übermittelt werden soll.	Beliebiger Text.	—	✓

Subelemente / Inhalt

Das ***<message>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<message>***-Element besitzt keine Variablenbindungen.

Beispiel

In folgendem Beispiel wird eine simple Nachricht an das EOMS-Core übertragen.

```
<rcml>
<process id="ExampleProcess" name="example process">
...
<message text="Hallo EOMS!" />
...
</process>
</rcml>
```

<message>

Zweck:

Nachrichtenübertragung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<exec>**
      - **<message>**
    - **<if>/<else>**
      - **<message>**
    - **<switch>**
      - **<case>**
        - **<message>**
    - **<while>**
      - **<message>**

Beachte

Messages werden eventuell nicht sofort abgeschickt, sondern es wird aus Performancegründen gewartet, bis eine Message kosteneffektiv übertragen werden kann.

<releaseresource>

Semantik

<releaseresource> gibt eine Ressource, die zuvor mit *<fetchresource>* gebunden wurde, manuell wieder frei. Dadurch entfällt die Bindung an die Ressource im Speicher. Nach Aufruf von ***<releaseresource>*** ist die betreffende Ressource nicht weiter verwendbar.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
resource	***FETCHEDRESOURCE***	Die ID der *<fetchresource>*, deren Ressource freigegeben werden soll.	ID einer existierenden Ressource.	—	✓

Subelemente / Inhalt

Das ***<releaseresource>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<releaseresource>***-Element besitzt keine Variablenbindungen.

Beispiel

In folgendem Beispiel wird mit *<fetchresource>* die vom Spooler empfangene Datei an die id = "inputFile" gebunden. Über die Variablenbindungen .getFile().getAbsolutePath() wird der Pfad zur Datei in einem *<param>* gespeichert und dann verwendet. Anschließend wird die Datei umbenannt und direkt wieder an den Spooler zurückgeschickt. Zuletzt wird die Ressource mit ***<releaseresource>*** manuell freigegeben. Das Beispiel verwendet außerdem das *<workdir>*-, *<exec>*-, *<param>*-, *<commandline>*-, und *<result>*-Element.

```
<rcml>
<process id="ExampleProcess" name="example process">

<!-- Arbeitsverzeichnis für den Prozess setzen. -->
<workdir id="workdir" home="./WORK" />

<!-- Vom Spooler Ressource empfangen und an die id "inputFile"
binden. -->
<fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />

<exec id="RenameProcess" workdir="workdir">

<!-- Parameter anlegen: Pfad zur Ressource "inputFile". Dies muss
über <param> geschehen! -->
<param name="infile"
value="${inputFile.getFile().getAbsolutePath()}" />

<!-- Parameter anlegen: Der neue Pfad mit neuem Namen. Dies muss
über <param> geschehen! -->
<param name="outfile" value="${workdir.getAbsolutePath() +
'/cool_new_file'}" />

<!-- Ressource umbenennen in "cool_new_file"... -->
<commandline processor="velocity">cmd move $infile $outfile
</commandline>

<!-- Bei Fehlern abbrechen. -->
<result return-code="0" />

</exec>

<!-- Umbenannte Datei samt komplettem Arbeitsverzeichnis an Spooler
schicken. -->
<upload file="workdir"
destination="${process['eoms.process.output']}" />

<!-- Ressource manuell wieder freigeben. -->
<releaseresource resource="inputFile" />

</process>
</rcml>
```

<releaseresource>

Zweck:

Ressourcenfreigabe

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<releaseresource>**
    - **<exec>**
      - **<releaseresource>**
    - **<if>/<else>**
      - **<releaseresource>**
    - **<switch>**
      - **<case>**
        - **<releaseresource>**
    - **<while>**
      - **<releaseresource>**

Beachte

—

<rw-response>

Semantik

<rw-response> dient dazu, ein ReportWriter ResponseObject zu empfangen, zu speichern und den Zugriff darauf (und auf seine Eigenschaften) zu ermöglichen.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Elements, über die es im Code angesprochen werden kann.	beliebiger, regelkonformer Name.	—	✗
file	***FILEOBJECT***	Das ReportWriter ResponseObject, das in den Speicher geschrieben werden soll.	—	reportw_response	✗
workdir	***FILEOBJECT***	Gibt das Arbeitsverzeichnis an, in dem der Prozess gestartet werden soll.	Die ID eines *<workdir>*-Elements.	—	✓

Subelemente / Inhalt

Das ***<rw-response>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<rw-response>***-Element ist vom Typ ***RW-RESPONSE***. Dieser Typ besitzt folgende Variablenbindungen:

Name	Beschreibung	Rückgabetyp
getVariable(STRING name)	Gibt die Variable mit dem Namen ***name*** aus dem ResponseObject zurück.	Abhängig vom Typ der Variable.
var(STRING name)	Identisch zu ***getVariable(STRING name)***.	Abhängig vom Typ der Variable.
intVar(STRING name)	wie ***getVariable(STRING name)***, gibt aber in jedem Fall einen *INTEGER* zurück. Gibt 0 zurück, falls der Variablenname unbekannt ist.	*INTEGER*
booleanVar(STRING name)	wie ***getVariable(STRING name)***, gibt aber in jedem Fall einen *BOOLEAN* zurück. Gibt false zurück, falls der Variablenname unbekannt ist.	*BOOLEAN*
getReturnCode()	Gibt den ReturnCode des ReportWriters zurück. Vergleiche dazu die Dokumentation des ReportWriters.	*INTEGER*
getPriority()	Vergleiche dazu die Dokumentation des ReportWriters.	—
getLocale()	Vergleiche dazu die Dokumentation des ReportWriters.	*STRING*
getMessage()	Gibt die (Status-)Message des Objekts zurück.	*STRING*
getFiles()	Gibt eine Liste der ReportWriter ResponseFiles zurück (falls mehrere vorhanden).	*LIST OF REPORTW_RESPONSEOBJECTS*

Beispiel

Hierzu ist momentan leider kein Beispiel verfügbar.

<rw-response>

Zweck:

Datenübertragung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Ja

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<rw-response>**
    - **<exec>**
      - **<rw-response>**
    - **<if>/<else>**
      - **<rw-response>**
    - **<switch>**
      - **<case>**
        - **<rw-response>**
    - **<while>**
      - **<rw-response>**

Beachte

—

<sleep>

Semantik

Das ***<sleep>***-Element hält die Verarbeitung für die angegebene Anzahl ms an ("legt sich schlafen"). Die Ausführung wird nur fortgesetzt, wenn während der sleeptime keine Ausnahme aufgetreten ist.

<sleep> kann z.B. eingesetzt werden, um externen Prozessen die Möglichkeit zu geben, eine Verarbeitung zu beenden. So kann es zum Beispiel vorkommen, dass ein externer Prozess die Steuerung an den Worker-Prozess zurückgibt, aber das Schreiben der Ergebnisdateien über Festplatten-Cache noch einige Millisekunden länger Zeit benötigt. In diesem Fall ist es günstig ***<sleep>*** zu verwenden, welches den Worker-Prozess um die angegebene Zeit (in Millisekunden) pausiert.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
time	***STRING (long)***	Die Anzahl Millisekunden, für die die Verarbeitung unterbrochen werden soll.	Numerische Angabe in ms. Muss mindestens 1 Sekunde betragen.	—	✔

Subelemente / Inhalt

Das ***<sleep>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<sleep>***-Element besitzt keine Variablenbindungen.

Beispiel

In folgendem Beispiel wird die Verarbeitung für 1 Sekunde unterbrochen, falls nach der Ausführung von *<commandline>* die Warnung mit dem Warncode 3 auftritt. Dieses Beispiel verwendet außerdem das *<workdir>*-, *<exec>*-, *<commandline>*-, ***<if>***- und *<message>*-Element.

```
<rcml>
<process id="ExampleProcess" name="example process">

<workdir id="workdir" home="./WORK" />

<exec id="ExampleExec" workdir="workdir">
<!-- Irgendeinen Befehl ausführen... -->
<commandline processor="velocity">...</commandline>
</exec>
<!-- Bei return-code = 3 Warnung an EOMS-Core senden und 1 Sekunde
schlafen legen. -->
<if condition="${ExampleExec.testReturnCode('3')}">
<message text="${'Warnung aufgetreten: ' +
ExampleExec.getReturnCode()}" />
<sleep time="1000" />
</if>

</process>
</rcml>
```

<sleep>

Zweck:

Prozessunterbrechung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<sleep>**
    - **<exec>**
      - **<sleep>**
    - **<if>/<else>**
      - **<sleep>**
    - **<switch>**
      - **<case>**
        - **<sleep>**
    - **<while>**
      - **<sleep>**

Beachte

Tritt während der Ablaufunterbrechung eine Exception auf, wird die Verarbeitung durch ***<sleep>*** nicht fortgesetzt. Nachfolgender Code wird nicht ausgeführt!

<switch>

> **How-To**
>
> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit dem ***<switch>***-Element ist es möglich, eine Kontrollstruktur mit beliebig vielen bedingten Ausführungen (<*case*>) zu implementieren. Ein <*case*> enthält Code, der nur ausgeführt wird, wenn die zugehörige Bedingung erfüllt ist. Das erste <*case*>, dessen Bedingung erfüllt ist, wird ausgeführt (dessen Inhalt wird ausgeführt). Alle nachkommenden <*case*> werden ignoriert, auch wenn deren Bedingung ebenfalls zutrifft. Die Überprüfung erfolgt von oben nach unten, weshalb es sinnvoll ist, Fallback-Routinen (Was soll passieren, wenn kein anderes <*case*> zutrifft) am Ende zu notieren. Eine vergleichbare Kontrollstruktur bilden <*if*>-<*else*>-Konstrukte. Bei <*if*> kann allerdings immer nur eine Bedingung (und gegebenenfalls per <*else*> das Gegenereignis) überprüft werden.

Attribute

Das ***<switch>***-Element besitzt keine Attribute.

Subelemente / Inhalt

Das ***<switch>***-Element kann nur das <*case*>-Element enthalten. Eine ***<switch>***-Anweisung muss mindestens 1 <*case*>-Element enthalten. Innerhalb eines ***<switch>*** dürfen keine anderen Elemente notiert werden.

- Das **<case>**-Element definiert eine Bedingung und enthält Code als Inhalt. Dieser Inhalt wird nur ausgeführt, wenn die Bedingu erfüllt ist. Alle nachfolgenden <*case*> werden dann ignoriert.

Variablenbindungen

Das ***<switch>***-Element besitzt keine Variablenbindungen.

Beispiel

In folgendem Beispiel wird der Rückgabewert eines <*exec*> mit einer ***<switch>***-Anweisung behandelt. Basierend auf dem Rückgabewert wird eine <*message*> an das EOMS-Core geschickt, die den Status beschreibt. Ein Beispiel, wie dies auch mit <if> - <else> statt ***<switch>*** umgesetzt werden kann, finden Sie hier. Weitere Beispiele zu <switch> finden Sie im HowTo - Kontrollstrukturen.

```
<rcml>
<process id="ExampleProcess" name="example process">

<exec id="ExampleExec" workdir="workdir">
<commandline processor="velocity">cmd echo Wow, what a simple
exec!</commandline>
</exec>

<switch>
<case condition="${ExampleExec.testReturnCode('0')}"> <!-- Falls
ReturnCode = 0 wird dieses case ausgeführt. -->
<message text="Befehl erfolgreich ausgeführt." />
</case>
<case condition="true"> <!-- Falls ReturnCode ungleich 0 wird
dieses case ausgeführt. -->
<message text="Befehlsausführung fehlgeschlagen." />
</case>
</switch>

</process>
</rcml>
```

<switch>

Zweck:

Kontrollstruktur

Typ:

Top-Level

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<switch>**
      - <case>
    - **<exec>**
      - **<switch>**
        - <case>
    - **<if>/<else>**
      - **<switch>**
        - <case>
    - **<switch>**
      - **<case>**
        - **<switch>**
          - <case>
    - **<while>**
      - **<switch>**
        - <case>

Beachte

Trifft keine <*case*>-Bedingung zu, wird der letzte <*case*> ausgeführt (es wird immer zwingend ein <*case*> au sgeführt). Innerhalb eines ***<switch>***-Elements dürfen nur <*case*>-Elemente notiert werden.

⚠ ***<switch>*** ist zwar ein Top-Level-Element, darf aber nur <*case*> beinhalten!

<case>

> **How-To**
>
> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

<case>-Elemente stellen bedingte Anweisungen innerhalb eines <switch> dar. In einem <switch> dürfen beliebig viele **<case>** not iert werden (mindestens jedoch 1). **<case>** wird über das Attribut ***condition*** die boolsche Bedingung übergeben. Bei der Abarbeitung des <switch> wird dann von oben nach unten jedes **<case>** auf seine Bedingung geprüft. Das erste **<case>**, dessen ***condition*** eine wahre Aussage ergibt (zutriff), wird ausgeführt und damit der Code, den es beinhaltet. Alle nachfolgenden **<case>** werden dann ignoriert. Deshalb ist die Reihenfolge der **<case>**-Ausdrücke in einem <switch> wichtig. Trifft keines der **<case>**-Ele mente eines <switch> zu (keines hat eine wahre Bedingung) wird das letzt **<case>** ausgeführt, obwohl die Bedingung unwahr ist.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
condition	***STRING***	Die boolsche Bedingung, die geprüft werden soll. Nur wenn die Bedingung wahr ist, wird der Inhalt des **<case>** ausgeführt. Hier können auch berechnete Werte stehen.	Beliebiger boolscher Ausdruck	—	✓

Subelemente / Inhalt

In einem **<case>**-Element dürfen alle RCML-Elemente notiert werden, außer: *<rcml>, <process>.*

Als Inhalt wird der Code notiert, der ausgeführt werden soll, falls die ***condition*** wahr ist.

Ein **<case>**-Element kann auch <switch>-Elemente enthalten. Somit ist es auch möglich, <switch>-Anweisungen zu schachteln.

Variablenbindungen

Das **<case>**-Element besitzt keine Variablenbindungen.

Beispiel

In folgendem Beispiel wird der Rückgabewert eines *<exec>* mit einer *<switch>*-Anweisung behandelt. Basierend auf dem Rückgabewert werden dann das entsprechende ***<case>*** ausgeführt, welches eine *<message>* an das EOMS-Core geschickt, die den Status beschreibt (entspricht weitgehend dem Beispiel aus *<switch>*). Weitere Beispiele finden Sie im HowTo - Kontrollstrukturen.

```
<rcml>
<process id="ExampleProcess" name="example process">

<exec id="ExampleExec" workdir="workdir">
<commandline processor="velocity">cmd echo Wow, what a simple
exec!</commandline> <!-- Endet mit ReturnCode 0 -->
</exec>

<switch>
<case condition="true"> <!-- Falls ReturnCode ungleich 0 wird
dieses case ausgeführt.
Dies ist hier nicht der Fall -> Code wird nicht ausgeführt. -->
<message text="Befehlsausführung fehlgeschlagen." />
</case>
<case condition="${ExampleExec.testReturnCode('0')}"><!-- Falls
ReturnCode = 0 wird dieses <case> ausgeführt.
Dies ist hier der Fall -> Code wird ausgeführt. -->
<message text="Befehl erfolgreich ausgeführt." />
</case>
</switch>

</process>
</rcml>
```

<case>

Zweck:

Bedingte Anweisung

Typ:

Top-Level

Elternelement:

<switch>

Subelemente:

Ja

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<switch>**
      - **<case>**
    - **<exec>**
      - **<switch>**
        - **<case>**
    - **<if>/<else>**
      - **<switch>**
        - **<case>**
    - **<switch>**
      - **<case>**
        - **<switch>**
          - **<case>**
    - **<while>**
      - **<switch>**
        - **<case>**

Beachte

<case> darf nur innerhalb von *<switch>* notiert werden. ***<case>*** darf alle RCML-Elemente (außer *<rcml>*, *<pr ocess>*) enthalten. Es ist auch erlaubt, innerhalb von ***<case>*** *<switch>* zu notieren, somit können <switch>-A nweisungen auch geschachtelt werden.

<update-variable>

> **How-To verfügbar**
>
> Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Das ***<update-variable>***-Element ermöglicht es, Variablen anzulegen und zu ändern. Variablen sind Objekte im Speicher, die einen Namen und einen zugehörigen Wert haben. Existiert bereits eine Variable mit dem angegebenen Namen wird ihr Wert auf den neuen Wert geändert, existiert die Variable noch nicht, wird sie neu angelegt. Mit ***<update-variable>*** können auch bestimmte RCML-Variablen, die durch das System angelegt wurden, geändert werden (z.B. "return-code"). Mit ***<update-variable>*** deklarierte Variablen sind außerdem im Auftrags-System verfügbar, weshalb sich ***<update-variable>*** auch zum Informations- und Variablenaustausch mit dem Auftrags-System einsetzen lässt (siehe auch How-To - Variablenaustausch mit Auftrags-Systemen). Es besteht allerdings keine Garantie dafür, dass das Auftrags-System die Variablen nicht verwirft.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
name	***STRING***	Der Name der Variable, die geändert oder neu angelegt werden soll.	beliebige, regelkonforme Zeichenkette.	—	✓
value	***STRING***	Der (neue) Wert der Variable.	beliebige, regelkonforme Zeichenkette.	—	✓

Subelemente / Inhalt

Das ***<update-variable>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<update-variable>***-Element besitzt keine Variablenbindungen.

Beispiel

Im folgenden Beispiel wird per ***<update-variable>*** eine Variable "myvar" mit dem Inhalt "hello!" angelegt. Der Inhalt der Variable soll nun in *<commandline>* verwendet werden. Dazu muss sie zuvor mit *<param>* verfügbar gemacht werden (dies gilt nur für *<exec>*-Container). Deshalb wird in Zeile 11 dem Parameter "myparam" der Wert von *myvar* zugewiesen und in Zeile 13 per Befehl ausgegeben. Das Beispiel verwendet außerdem das *<workdir>*-Element.

```
<rcml>
<process id="ExampleProcess" name="example process">

<workdir id="workdir" home="./WORK" />

<!-- Variable mit Wert "hello!" anlegen -->
<update-variable name="myvar" value="hello!" />

<exec id="ExampleExec" workdir="workdir">
<!-- Parameter mit Wert von myvar anlegen -->
<param name="myparam" value="${myvar}" />
<!-- Ausgeben. -->
<commandline processor="velocity">cmd echo $myparam</commandline>
</exec>

</process>
</rcml>
```

<update-variable>

Zweck:

Variablendeklaration

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Variablenbindungen:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- <rcml>
  - <process>
    - <update-variable>
    - <exec>
      - <update-variable>
    - <if>/<else>
      - <update-variable>
    - <switch>
      - <case>
        - <update-variable>
    - <while>
      - <update-variable>

Beachte

—

<upload>

Semantik

Das ***<upload>***-Element dient dazu, Dateien oder Verzeichnisse (Objekte vom Typ ***FILEOBJECT***) an eine Zieladresse zu senden (z.B. das Auftrags-System). Verzeichnisse werden redundant durchlaufen, d.h. der komplette Inhalt eines Verzeichnisses inklusive Unterverzeichnissen wird versendet. Dabei kann durch die Attribute ***include*** und ***exclude*** Filterausdrücke angegeben werden, die bestimmen, welche Dateien und Verzeichnisse nicht gesendet bzw. explizit gesendet werden sollen.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
file	***FILEOBJECT***	Die Datei oder das Verzeichnis, das an die angegebene Adresse (***destination***) gesendet werden soll.	ein existierendes ***FILEOBJECT***	—	✓
destination	***STRING***	Die Adresse, an die ***file*** gesendet werden soll.		—	✓
includes	***STRING***	Regulärer Ausdruck, der angibt, welche Dateien und Verzeichnisse in die Übertragung eingeschlossen werden sollen. Es werden nur Objekte übertragen, deren Name auf den Ausdruck passt. Alle anderen Dateien werden nicht übertragen.	Regulärer Ausdruck.	*.*	✗
excludes	***STRING***	Regulärer Ausdruck, der angibt, welche Dateien und Verzeichnisse von der Übertragung ausgeschlossen werden sollen. Objekte, deren Name auf den Ausdruck passt, werden nicht übertragen. Alle anderen Dateien werden nicht übertragen.	Regulärer Ausdruck.	—	✗

Subelemente / Inhalt

Das ***<upload>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<upload>***-Element besitzt keine Variablenbindungen.

Beispiel

Im folgenden Beispiel soll die Funktionsweise von ***<upload>*** mit Filtern verdeutlicht werden. Dazu wird zunächst mit *<fetchresource>* eine Ressource vom Auftrags-System geholt und an die ***id*** "inputFile" gebunden. Die Datei soll mit der Endung .txt versehen und ins Arbeitsverzeichnis (*<workdir>*) kopiert werden. Von dort soll dann nur diese Datei an das Auftrags-System zurückgesendet werden. Da wir den Dateinamen mehrfach verwenden, legen wir zunächst mit <update-variable> die Variable "FileName" an, die genau das enthält. Anschließend soll die Umbenennung per *<commandline>* durchgeführt werden. Dazu legen wir 2 Parameter an: Parameter 1 Enthält den Pfad zur Datei, Parameter 2 enthält den neuen Pfad der Datei: Dieser besteht aus dem Pfad der *<workdir>*, dem Dateinamen sowie der Endung .txt. Danach folgt die Umbenennung / Verschiebung. Abschließend wird die "neue" Datei mit ***<upload>*** an das Auftrags-System geschickt. Dabei wird durch den Parameter ***includes*** angegeben, dass nur Dateien und Verzeichnisse, die exakt den Namen (samt Endung) unserer Datei haben, mitgeschickt werden.

```
<rcml>
<process id="ExampleProcess" name="example process">

<workdir id="workdir" home="./WORK" />
<!-- Ressource vom Spooler holen. -->
  <fetchresource id="inputFile"
resource="${process['eoms.process.input']}" />

<!-- Variable anlegen, die den Dateinamen enthält -->
<update-variable name="FileName"
value="inputFile.getFile().getName()" />

<!-- Hier wird die Ressource zu einer .txt konvertiert... -->
<exec id="ExampleExec" workdir="workdir">
<!-- Parameter 1: Der Pfad zur Datei. -->
<param name="file" value="${inputFile.getFile().getAbsolutePath()}"
/>
<!-- Parameter 2: Der neue Pfad, an den kopiert werden soll. Die
Datei bekommt eine neue Endung .txt. -->
<param name="TXTfile" value="${workdir + '/' + FileName + '.txt'}"
/>
<commandline>cmd /C move $file $TXTfile</commandline>
</exec>

<!-- Nur Dateien hochladen, die den Dateinamen samt .txt Endung
haben. -->
<upload file="workdir"
destination="${process['eoms.process.output']}" include="${FileName
+ '.txt'}" />

</process>
</rcml>
```

<upload>

Zweck:

Datenübertragung

Typ:

Standalone

Elternelement:

Top-Level-Elemente

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<upload>**
    - **<exec>**
      - **<upload>**
    - **<if>/<else>**
      - **<upload>**
    - **<switch>**
      - **<case>**
        - **<upload>**
    - **<while>**
      - **<upload>**

Beachte

—

<while>

How-To

Für dieses Element ist ein How-To Artikel verfügbar.

Semantik

Mit dem **<*while*>**-Element ist es möglich, Anweisungen mehrfach ausführen zu lassen *(=Schleife)*. Der Inhalt von **<*while*>** wird so oft wiederholt bis die ***condition*** unwahr wird. Dazu wird jedes Mal vor der Ausführung geprüft, ob die Bedingung wahr ist und falls sie wahr ist, der Inhalt des Elements ausgeführt. Achten Sie unbedingt darauf, dass Sie sicherstellen (durch das Verhalten innerhalb der Schleife), dass die Bedingung nach einer endlichen Anzahl von Wiederholungen unwahr wird, da Sie sonst eine Endlosschleife riskieren, die zum Absturz des Prozesses führen kann.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
condition	***STRING***	Die boolsche Bedingung, die geprüft werden soll. Nur wenn die Bedingung wahr ist, wird der Inhalt des **<*while*>** ausgeführt. Hier können auch berechnete Werte stehen. Die Bedingung wird bei jeder neuen Schleifenwiederholung erneut geprüft.	Beliebiger boolscher Ausdruck	—	✓

Subelemente / Inhalt

In einem **<*while*>**-Element dürfen alle RCML-Elemente notiert werden, außer: *<rcml>, <process>.*

Als Inhalt wird der Code notiert, der ausgeführt werden soll, falls die ***condition*** wahr ist.

Es ist auch erlaubt, **<*while*>**-Elemente zu schachteln (**<*while*>** innerhalb von **<*while*>** zu notieren).

Variablenbindungen

Das **<*while*>**-Element besitzt keine Variablenbindungen.

Beispiel

Zu diesem Element gibt es ein umfangreiches HowTo mit Beispielen.

<while>

Zweck:

Schleife

Typ:

Top-Level

Elternelement:

Top-Level-Elemente

Subelemente:

Ja

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<while>**
    - **<exec>**
      - **<while>**
    - **<if>/<else>**
      - **<while>**
    - **<switch>**
      - **<case>**
        - **<while>**
    - **<while>**
      - **<while>**

Beachte

Achten Sie darauf, innerhalb der Schleife dafür zu sorgen, dass die Bedingung nach einer endlichen Anzahl von Schritten unwahr wird, um eine Endlosschleife zu verhindern!

<workdir>

Semantik

Durch das ***<workdir>***-Element können Arbeitsverzeichnisse für den Prozess gesetzt werden. Sämtliche Prozessvorgänge finden dann standardmäßig in diesem Verzeichnis statt, der Worker nutzt ***<workdir>*** dann auch als File-Repository, weshalb die Ergebnisdaten des Prozesses auch dort zu finden sind (und gegebenenfalls per *<upload file="workdir_id">* auch wieder abgeschickt werden). Der Worker nimmt für diesen Prozess dann das gesetzte Verzeichnis als Arbeitsverzeichnis. ***<workdir>*** kan n außerdem von Elementen wie z.B. *<exec>* referenziert und als Rootverzeichnis genutzt werden. Diese Elemente greifen dann in ihrer Ausführung auf das gesetzte Verzeichnis zurück (*<exec>* z.B. startet die Ausführung dann in der angegebenen ***<workdir>***). Das ***<workdir>***-Element wird dabei von anderen Elementen über seine ***id*** angesprochen. Ohne explizite ***<workdir>***-Angabe wird grundsätzlich das in der *eoms.invoker.client.properties* festgelegte Arbeitsverzeichnis verwendet. Existiert das Verzeichnis noch nicht, wird es angelegt.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
id	***STRING***	Die ID des Elements, über die es im Code angesprochen werden kann.	beliebige, regelkonforme Zeichenkette.	—	✗
home	***INTEGER***	Pfad zu dem zu setzenden Arbeitsverzeichnis (wird angelegt falls es noch nicht existiert). Pfadangaben können relativ oder absolut sein. Es gelten die üblichen Syntaxregel für Pfadnamen (. = aktuelles Verzeichnis; .. = Übergeordnetes Verzeichnis; / = Trennzeichen). Wird für home ein leerer Pfad übergeben wird standardmäßig das in *eoms.invoker.client.properties* angegebene Arbeitsverzeichnis genommen.	Gültiger Systempfad	Standard-Arbeitsverzeichnis aus *eoms.invoker.client.properties*	✗

clear-on-shutdown	***BOOLEAN***	Gibt an, ob das Verzeichnis nach Beendigung des Prozesses automatisch gelöscht werden soll (true) oder nicht (false) oder ob es nur gelöscht werden soll, falls kein Fehler aufgetreten ist (keep-on-error). Im Produktiv einsatz sollte hier true gesetzt werden. Verzeichnisse können mit *<delete>* auch manuell gelöscht werden.	"true" : "false" : "keep-on-error"	false	

Subelemente / Inhalt

Das ***<workdir>***-Element besitzt keine Subelemente und kann keinen Inhalt enthalten.

Variablenbindungen

Das ***<workdir>***-Element ist vom Typ ***FILE-OBJECT***. Dieser Typ besitzt folgende Variablenbindungen:

Name	Beschreibung	Rückgabetyp
getName()	Gibt den Namen des Verzeichnisses zurück.	*STRING*
getAbsolutePath()	Gibt den absoluten Pfad zur Datei zurück.	*STRING*

Beispiel

Im folgenden Beispiel wird das Arbeitsverzeichnis auf *C:/EOMS_WORKER_TEMP_DIR* gesetzt. Nach Beendigung des Prozesses soll das Verzeichnis wieder gelöscht werden. Außerdem wird das Arbeitsverzeichnis von *<exec>* als Startverzeichnis für seine Befehlsausführung genutzt. Per *<commandline>* wird eine neue Datei angelegt, die Aufgrund der workdir-Angabe danach in *C:/EOMS_WORKER_TEMP_DIR* zu finden ist.

```
<rcml>
<process id="ExampleProcess" name="example process">

<!-- Arbeitsverzeichnis: C:/EOMS_WORKER_TEMP_DIR. Nach Beendigung
löschen. -->
<workdir id="workdir" home="./WORK" clear-on-shutdown="true" />

<!-- Verwende für die Befehlsausführung C:/EOMS_WORKER_TEMP_DIR als
Startverzeichnis -->
<exec id="ExampleExec" workdir="workdir">
<!-- <commandline> wird jetzt in C:/EOMS_WORKER_TEMP_DIR
ausgeführt.
Der Befehl legt eine neue Datei mit Namen newfile.txt an. Da der
Prozess im workdir "workdir" gestartet wird,
wird die neue Datei C:/EOMS_WORKER_TEMP_DIR/newfile.txt angelegt.
-->
<commandline processor="velocity">cmd echo Hallo Welt >
newfile.txt</commandline>

</exec>

</process>
</rcml>
```

<workdir>

Zweck:

Verzeichnisdeklaration

Typ:

Standalone

Elternelement:

<process>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<process>**
    - **<workdir>**
    - **<exec>**
      - **<workdir>**
    - **<if>/<else>**
      - **<workdir>**
    - **<switch>**
      - **<case>**
        - **<workdir>**
    - **<while>**
      - **<workdir>**

Beachte

Wenn Sie clear-on-shutdown="false" verwenden, können Sie das Verzeichnis auch manuell durch *<delete>* l öschen.

<script>

How-To

Für dieses Element ist ein HowTo - Artikel verfügbar.

Semantik

Das ***<script>***-Element ermöglicht es, Skriptcode *(JavaScript)* in RCML einzufügen und zu verwenden. Der Scriptcode wird als Elementinhalt von ***<script>*** notiert. Die notierten Methoden und Eigenschaften stehen dann in der gesamten RCML zur Verfügung, unabhängig davon, in welchem Element das ***<script>*** notiert ist (es gibt nur den globalen Speicherbereich). ***<script>*** i st zwar überall im Dokument erlaubt, Kontrollanweisungen haben aber keinen Effekt auf das Element: So können Sie zwar ***<script >*** innerhalb eines *<while>* notieren, das Skript wird aber dennoch nur einfach definiert. Generell ist davon abzuraten. Notieren Sie ***<script>***-Elemente besser zu Beginn des Dokuments oder zu Beginn des entsprechenden *<process>*. Hier wird nicht näher auf JavaScript eingegangen, da dies den Rahmen dieser Dokumentation sprengen würde. Ein gutes Einsteiger-Tutorial finden Sie auf W3Schools.

Skript kann nicht aus externen .js-Dateien eingebunden werden.

Attribute

Attributname	Datentyp	Beschreibung	Mögliche Werte	Standardwert	Obligatorisch?
language	***STRING***	Die verwendete Skriptsprache für diesen Skriptbereich.	Standardmäßig wird nur "JavaScript" unterstützt.	—	✓

Subelemente / Inhalt

Das ***<script>***-Element besitzt keine Subelemente. Als Inhalt wird der Skriptcode notiert.

Variablenbindungen

Das ***<script>***-Element besitzt keine Variablenbindungen.

Beispiel

In folgendem Beispiel wird eine einfache JavaScript-Funktion definiert, die die aktuelle Zeit zurückgibt. Dies wird dann in einem <*message*>-Element verwendet und an das EOMS-Core gesendet. Weitere Beispiele finden Sie im HowTo - JavaScript in RCML.

```
<rcml>
<process id="rs" name="Redaktions-System">
<script>
<![CDATA[
function getTime{
var date = new Date();
return date.toTimeString();
}
]]>
</script>
<!-- Aufruf der JavaScript-Funktion. -->
<message text="${'Starting rs process at' + getTime()}" />
</process>
</rcml>
```

<script>

Zweck:

Skriptbereich

Typ:

Standalone

Elternelement:

<rcml>,<process>

Subelemente:

Nein

Variablenbindungen:

Nein

Verzweigungsstruktur

- **<rcml>**
  - **<script>**
  - **<process>**
    - **<script>**
    - **<exec>**
      - **<script>**
    - **<if>/<else>**
      - **<script>**
    - **<switch>**
      - **<case>**
        - **<script>**
    - **<while>**
      - **<script>**

Beachte

Umschließen Sie den Skriptcode im ***<script>***-Elemente mit der Vorsilbe **<![CDATA[** und der Nachsilbe **]]>**, um auszuschließen, dass Skriptcode vom Interpreter versehentlich als XML erkannt wird.

Anbindung Auftrags-Systeme

Dieses Kapitel beschreibt, wie Auftrags-Systeme an das EOMS angebunden werden können. Momentan sind folgende Beschreibungen verfügbar:

Spooler Anbindung

Die Anbindung eines oder mehrerer Spooler an das EOMS ist einfach und in wenigen Schritten abgeschlossen. Einen Überblick darüber bietet die Kurzübersicht Konfiguration und auch der entsprechende Artikel in der Spooler-Dokumentation.

Schritt 1: EOMS-Core und EOMS-Worker ordnungsgemäß einrichten und konfigurieren

Besonders wichtig für den Spooler ist hierbei, dass in der **eoms.invoker.client.properties:**

- mindestens ein Ressource-Fetcher definiert ist (z.B. der Standardfetcher)
- der oder die Ressource-Fetcher, die mit dem Spooler kommunizieren sollen, richtig konfiguriert sind, nämlich:
  - ***eoms.resource.spooler.XXX.protocol*** das richtige Protokoll angibt, standardmäßig "http" (oder "https"),
  - ***eoms.resource.spooler.XXX.host*** die korrekte IP-Adresse des Rechners, auf dem der Spooler läuft, enthält und
  - ***eoms.resource.spooler.XXX.port*** den richtigen Port, an dem der Spooler lauscht, standardmäßig 62616.

    dabei ist XXX optional der Name des Fetchers.
- Beispielfetcher:

```
eoms.resource.spooler.protocol = http
eoms.resource.spooler.host = 192.168.128.85
eoms.resource.spooler.port = 62616
```

Schritt 2: Einrichtung des Spoolers

Dazu muss ein neues Programm in der Prozesssteuerung des Spoolers eingerichtet werden.

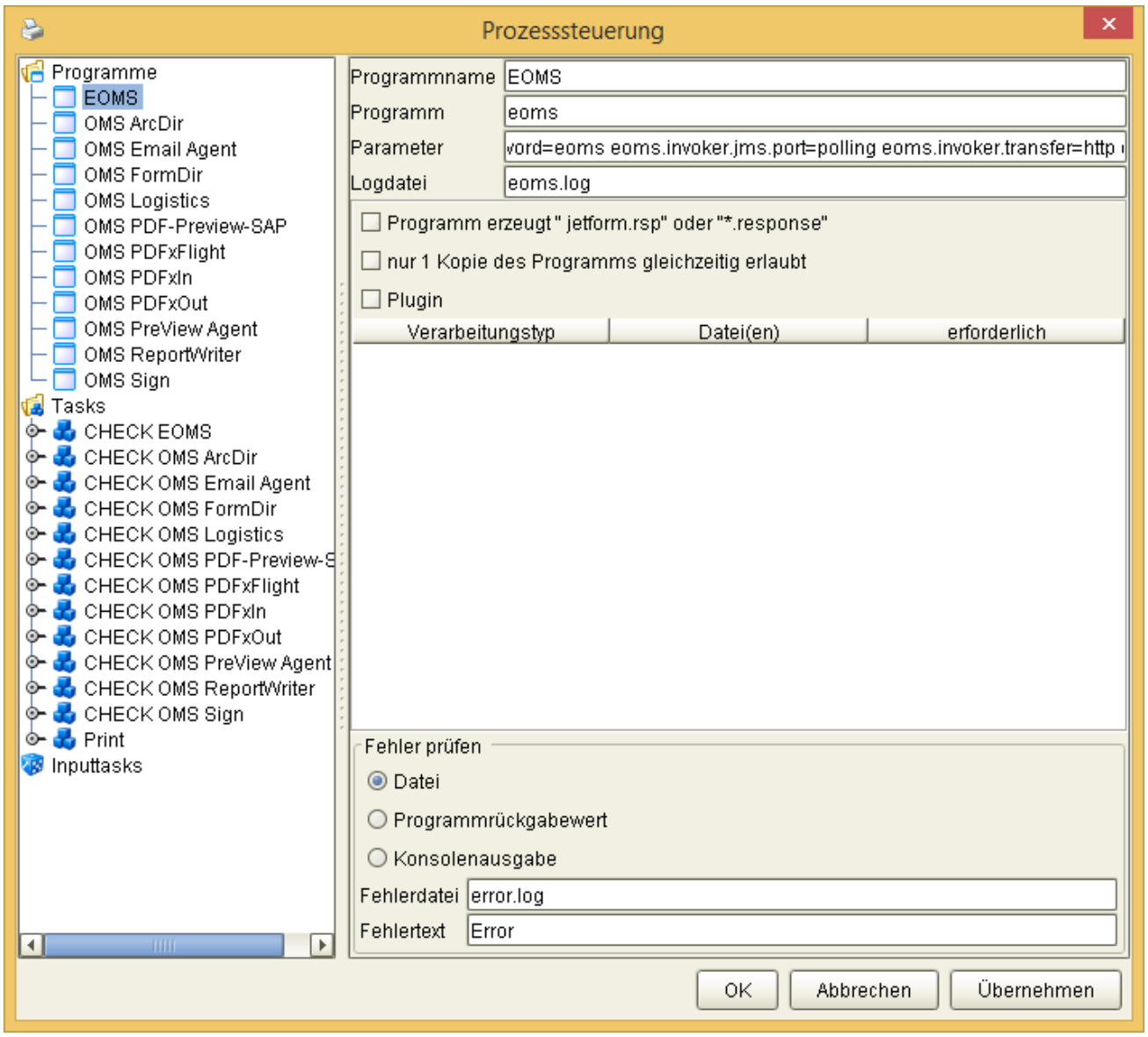


Abbildung A: Prozesssteuerung des Spoolers.

Eine minimale Parameterübergabe an das EOMS könnte so aussehen:

```
eoms.invoker.host=<host> eoms.invoker.port=<port>
eoms.invoker.jms.host=polling
eoms.invoker.user=<user> eoms.invoker.password=<pw>
eoms.invoker.spooler=<fetcher>
eoms.invoker.jms.port=polling eoms.invoker.transfer=<protocol>
eoms.process=<process>
```

Beschreibung:

- ***<host>***
  - Ist die IP-Adresse des EOMS-Core-Hostrechners.
- ***<port>***
  - Ist der Port, auf dem der EOMS-Core auf dem Hostrechner lauscht.
- ***<user>***
  - Ist der Nutzername für die Authentifizierung am EOMS-Core.
- ***<pw>***
  - Ist das Passwort für die Authentifizierung am EOMS-Core.
- ***<fetcher>***
  - Ist der Name des Ressourcen-Fetchers, (nur nötig, falls im Worker mehr als 1 Fetcher definiert ist).
- ***<protocol>***
  - Ist das Übertragungsprotokoll (Standard: http).
- ***<process>***
  - Ist der Prozessname, der ausgeführt werden soll (z.B. 'rw').

Beispielaufruf:

```
eoms.invoker.host=192.168.128.85 eoms.invoker.port=8080
eoms.invoker.jms.host=polling eoms.invoker.user=eoms
eoms.invoker.password=eoms eoms.invoker.jms.port=polling
eoms.invoker.transfer=http eoms.process=copy copy.output=@FileName.
```

Details zur Parameterübergabe an dass EOMS werden hier nicht behandelt. Siehe dazu Variablenaustausch mit Auftrags-Systemen.

Nachdem Sie das Programm angelegt haben, müssen Sie einen neuen Task anlegen (Rechtsklick auf "Tasks" -> "Task hinzufügen"). Fügen Sie dann zu diesem Task ein neues TaskItem hinzu (Rechtsklick auf den eben erstellten Task -> "TaskItem hinzufügen"):

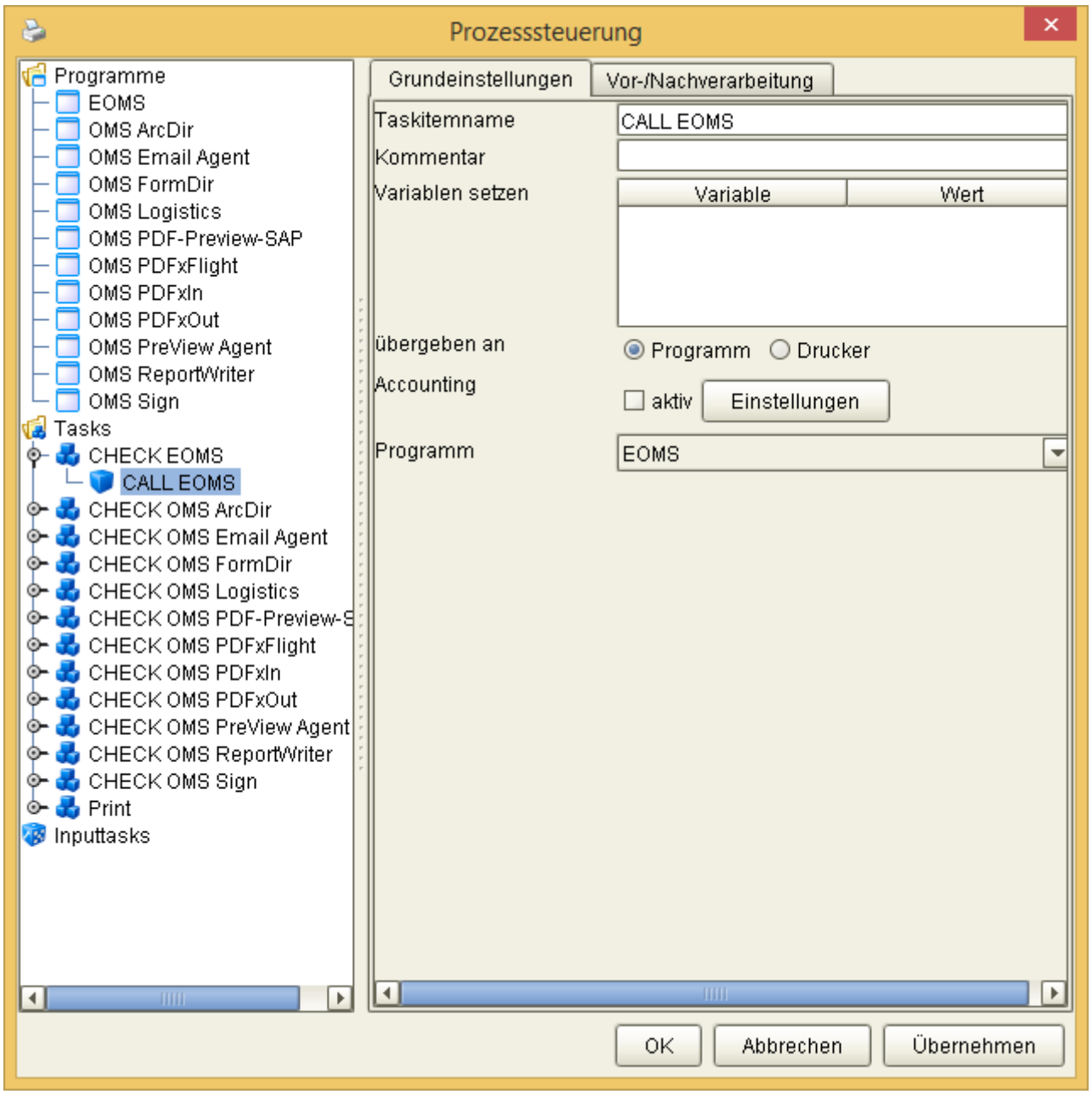


Abbildung B: Task mit TaskItem.

Setzten Sie im TaskItem als Programm das oben erstellte Programm EOMS.

Als letzten Schritt müssen Sie den neu erstellten Task als Inputtask setzen, damit dieser auch ausgeführt wird.

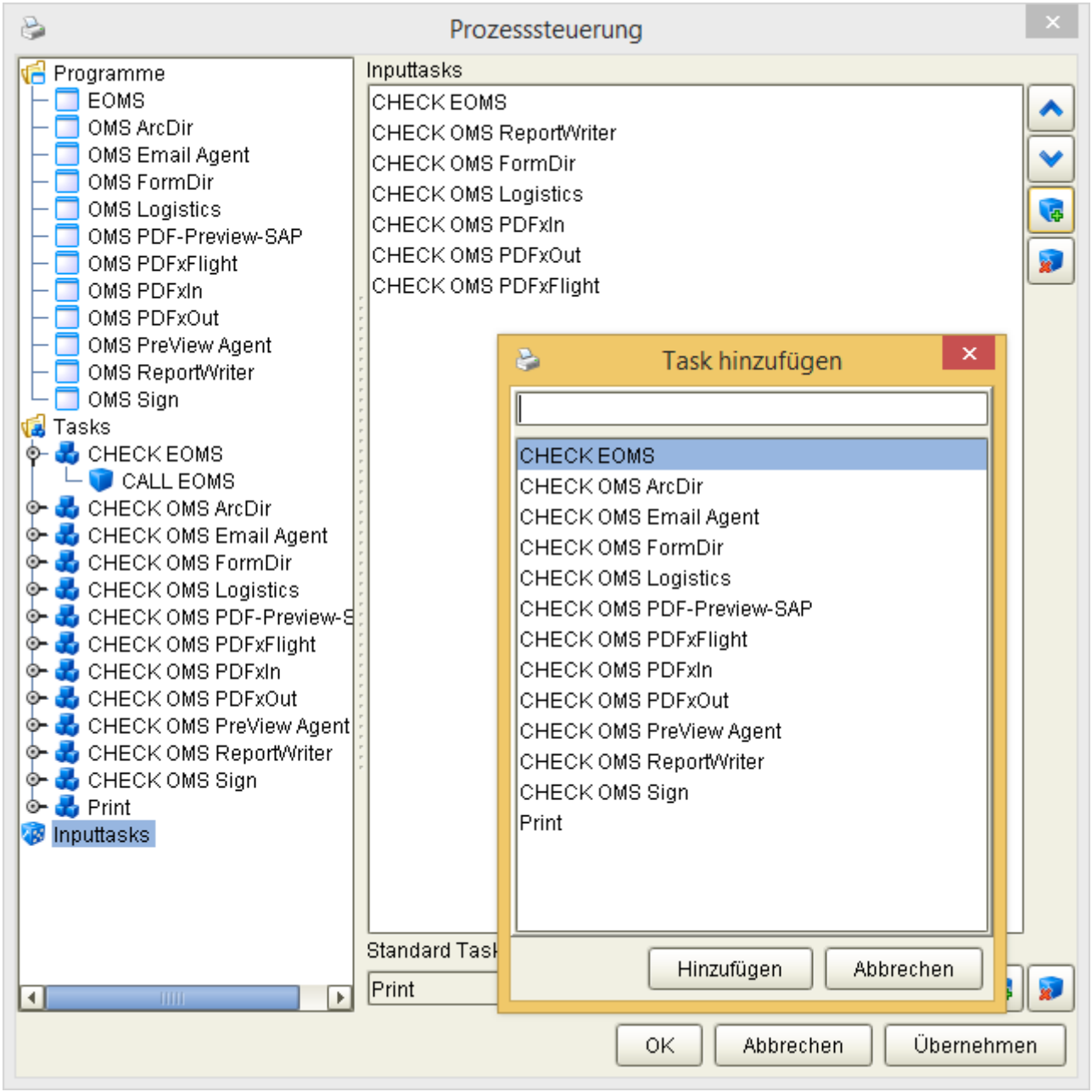


Abbildung C: Task hinzufügen.

Ordnen Sie den EOMS-Task dann in der Inputtaskliste dort ein, wo er aufgerufen werden soll (die Tasks werden chronologisch von oben nach unten aufgerufen).

Damit ist die Einrichtung des Spoolers für die Kommunikation mit dem EOMS abgeschlossen. Testen Sie nun die Verbindung, indem Sie einen Testjob vom Spooler verschicken.

REST

Das EOMS bietet Informationen und gewisse Steuerungselemente über eine REST-Schnittstelle an. Über diese Schnittstelle können Informationen angefordert werden, entweder direkt durch Aufruf der entsprechenden URI oder über spezielle Tools wie z.B. cURL. Der Zugriff auf diese Daten über Tools wie cURL erlaubt eine automatisierte, maschinelle Verarbeitung der Informationen.

Die Daten werden im XML-Format zurückgegeben.

Für die mit < > eingeschlossenen Platzhalter müssen Sie Ihre Werte eintragen, also z.B. für <host> die IP-Adresse des EOMS-Core-Rechners oder für <user> den Nutzername, mit dem Sie sich am EOMS authentifizieren.

Diese Auflistung der verfügbaren REST-URI's ist eventuell nicht vollständig.

Wenn mehr als eine URI für einen Aufruf aufrufbar ist, werden alle URI's für die entsprechende Funktionalität gelistet.

Die Basis-URI für REST-Aufrufe lautet:

```
http://<host>:<port>/omsinvoker/rest/
```

Folgende URI's (diese müssen an die Basis-URI angehängt werden) sind im EOMS über REST erreichbar (Davor in [ ] die entsprechende HTTP-Methode für den Aufruf angegeben):

URI	Beschreibung
[GET] /v1/monitor	Monitoring-Seite. Enthält Informationen über den aktuellen Zustand des EOMS.
[GET] /v1/monitor/history	Wie .../**monitor**, aber enthält nicht nur die aktuellen, sondern auch vergangene Monitoring-Daten. Wird als Menge von XML-Strings dargestellt.
[GET] /version **[GET] /v1/version**	Enthält Versionsangaben zum EOMS. Enthält ähnliche Daten wie die Über-Seite im EOMS-Core UI.
[GET] /info **[GET] /v1/info**	Enthält Informationen zum aktuellen Prozesszustand des EOMS (geplante, aktivierte, beendete Prozesse, ob die Prozessabarbeitung gestoppt ist, etc.).

[GET] /ip **[GET] /v1/ip**	Enthält die IP-Adresse des Servers, auf dem das EOMS-Core läuft.
[GET] /worker-list **[GET] /v1/worker-list**	Enthält Angaben zu den am EOMS-Core registrierten Workern.
[POST] /v1/worker?id={id}	Enthält Angaben zu einem bestimmten Worker mit der angegebenen id. Bsp.: ***/omsinvoker/rest/worker?id=41526***
[GET] /process-list?status={scheduled | activated | terminated} **[GET] /v1/process-list?status={scheduled | activated | terminated}**	Enthält eine List aller Prozesse (ohne den Parameter status, also: ***/omsinvoker/rest/process-list***) oder eine Liste aller Prozesse mit dem angegebenen Status, also z.B.: ***/omsinvoker/rest/process-list?status=terminated*** für eine Liste alle beendeten Prozesse.
[GET] /process-search?query={id} **[GET] /v1/process-search?query={id}**	Enthält Angaben zu allen Prozessen, deren id {id} enthält. Bsp.: ***/omsinvoker/rest/process?query=212*** liefert sowohl Angaben für den Prozess mit der id 212 als auch den mit der id 52126.
[GET] /process?query={id} **[GET] /v1/process/{id}**	Wie **.../process-search**, aber enthält nur Angaben zu dem Prozess, dessen id exakt mit der angegebenen {id} übereinstimmt. Wird kein Parameter angegeben entspricht der Aufruf ***/process-list*** ohne Parameterangabe. Beispiel in cURL: `curl -X GET -v http://<host>:<port>/omsinvoker/rest/v1/process/52123`

[POST] /processing?{resume | hold | hold/now} **[POST] /v1/processing?{resume | hold | hold/now}**	Ohne Parameter: Enthält den aktuellen Prozessstatus (processing, on-hold). Mit Parameter: Setzt die Verarbeitung auf den angegeben Wert (fortfahren, stoppen). ⚠ Benötigt Authentifizierung, weshalb hier nur POST-Aufrufe erlaubt sind, z.B. per cURL, z.B.: `curl -X POST -u <user>:<password> -v "http://<host>:<port>/omsinvoker/rest/processing/hold/now"` Zurückgegeben wird wie beim Aufruf ohne Parameter der aktuelle Prozesstatus nach Änderung durch den Aufruf.

[POST] /v1/process	Generiert einen neuen Prozess mit Status **Geplant *[scheduled]***. Beispiel in cURL: `curl -H "Content-Type: application/xml" -X POST -u <user>:<password> -d '<process> <priority>4</priority> <properties> <property name="eoms.process">Process 1</property> <property name="Operating-System">Windows x86 2003</property> </properties> <resources> <resource>Resource 1</resource> <resource>Resource 2</resource> <resource>Resource 3</resource> </resources> </process>' -v "http://<host>:<port>/omsinvoker/rest/v1/process"`

[PUT] /v1/process/{id}	Wie ***[POST] /omsinvoker/rest/v1/process***, aber es wird kein neuer Prozess erstellt, sondern der Prozess mit der angegebenen id abgeändert. Beispiel in cURL: `curl -H "Content-Type: application/xml" -X PUT -u <user>:<password> -d '<process> <protractTime>4500</protractTime> <properties> <property name="eoms.process">Process 1a</property> <property name="Operating-System">Mac OS X</property> </properties> <resources> <resource>Resource 1</resource> <resource>Resource 2</resource> <resource>Resource 3</resource> </resources> </process>' -v "http://<host>:<port>/omsinvoker/rest/v1/process/52123"`

[POST] /v1/process/{id}/activate	Aktiviert den Prozess mit der angegebenen id (Übergang von **Geplant** ***[scheduled]*** zu **Aktiviert** ***[activated]***). Beispiel in cURL: `curl -H "Content-Type: application/xml" -X POST -u <user>:<password> -d '<process><workerId>43435</workerId></process>' -v "http://<host>:<port>/omsinvoker/rest/v1/process/52123/activate"`
[DELETE] /v1/process/{id}	Entfernt den Prozess mit der angegebenen id. Beispiel in cURL: `curl -X DELETE -u <user>:<password> -v "http://<host>:<port>/omsinvoker/rest/v1/process/52123"`

Typische Fehler und Fehlerbehandlung

Hier finden Sie Lösungsstrategien für häufig auftretende Fehler. Dabei wird unterschieden zwischen:

- **Fehler beim Systemstart des EOMS-Core**
- **Fehler beim Zugriff auf den EOMS-Core-Monitor (grafische Oberfläche)**
- **Fehler beim Start eines EOMS-Workers**

Grundsätzlich sollten Sie bei Systemfehlern immer zunächst die Einträge in den Logfiles in *%EOMS-CORE_HOME%/logs* nach Auffälligkeiten durchsuchen.

Fehler beim Zugriff auf den EOMS-Core-Monitor (grafische Oberfläche)

Beim Aufruf des EOMS-Core-Monitors erscheint folgende Fehlermeldung: Die Webseite ist nicht verfügbar / konnte nicht gefunden werden (404).

Mögliche Ursachen:

A. Der Apache Tomcat Server ist nicht gestartet.

B. Der Apache Tomcat Server ist falsch konfiguriert.

C. Die Aufrufadresse ist falsch.

D. Der Port, auf dem der Tomcat Server läuft, wird von einer Firewall blockiert.

Lösung:

A. Überprüfen Sie, ob der Apache Tomcat Server erfolgreich gestartet wurde (unter Linux z.B. mit "*netstat -vatpn | grep java*"). Bei laufendem Tomcat wird der Port, der in der server.xml festgelegt wurde (Standard: 8080), belegt. Es ist auch möglich, dass der Port durch eine andere Java-Applikation belegt wird und Tomcat deshalb nicht gestartet werden konnte. Stellen Sie dann sicher, dass Tomcat nicht läuft (unter Linux durch "*%EOMS_HOME%/bin/catalina.sh stop*") und überprüfen Sie erneut die Portbelegung mit netstat. Ist der Port weiterhin belegt, muss die andere Applikation zuerst geschlossen werden, bevor Tomcat gestartet werden kann. Ist der Port nicht belegt und Tomcat kann trotzdem nicht gestartet werden, so liegt ein internes Problem vor. Nehmen Sie dann die Logfiles in *%EOMS_HOME%/logs* zur Hilfe.

B. Überprüfen Sie den Connector in der ***server.xml*** *(Anleitung dazu hier)* und stellen Sie sicher, dass der Connnector richtig konfiguriert ist. Dazu gehört u.a. die richtige Portangabe *(Standard: 8080)*. Falls Sie den EOMS-Core-Monitor vom Rechner, auf dem das EOMS-Core selbst installiert ist (gehostet wird) aufrufen können aber von keinem anderen Rechner, so müssen Sie den *Connector* um die Adressangabe erweitern, damit das EOMS-Core nicht nur lokal auf dem Rechner gehostet wird, sondern auch extern verfügbar ist. Ergänzen Sie den ***<Connector>***-Tag um das Attribut ***address="<ip>"***, wobei <ip> die IP-Adresse des Hostrechners ist, auf dem das EOMS-Core gehostet wird. Ob das EOMS-Core von externen Rechnern erreichbar ist lässt sich z.B. durch "***telnet <ip> <port>***" herausfinden. Mehr Informationen zur Connector-Konfiguration finden Sie hier.

C. Besteht das Problem weiterhin, so versuchen Sie eventuell, den EOMS-Core-Monitor unter einer falschen Adresse aufzurufen. Die Aufrufadresse hat die Form: **<protocol>://<ip>:<port>/omsinvoker**
<*protocol*> = Verwendetes Protokol (http oder https). Standard ist http. Wenn Tomcat in der server.xml mit aktiver SSL konfiguriert ist, müssen Sie https verwenden. Das Protokol sollte explizit angegeben werden.
<*ip*> = IP-Adresse des Hosts, auf dem das EOMS-Core gehostet wird (im <Connector>-Tag der server.xml angegebene IP-Adresse).
<*port*> = Port, auf dem Tomcat erreichbar ist (siehe A. und B., im <Connector>-Tag der server.xml angegebener Port). Standard ist 8080, falls Sie https verwenden 8443.

D. Können Sie trotz der Lösung in B. das EOMS-Core weiterhin nur von dem Hostrechner aus erreichen, so wird der Port eventuell von einer Firewall geblockt. Überprüfen Sie unter Linux mit "***iptables -nL | grep :<port>***" (z.B. "iptables -nL | grep :8080), ob der Port geblockt wird und entfernen Sie gegebenenfalls die entsprechende Regel, z.B. durch "***iptables -D <chain> <ruleNR>***", wobei <*chain*> = "INPUT" / "OUTPUT" und <*ruleNR*> = Die Nummer der betreffenden Regel ist.

Beim Aufruf im Browser wird nur eine leere Seite statt dem EOMS-Loginfenster angezeigt.

Mögliche Ursachen:

A. Die Datenbank ist inkonsistent.

Lösung:

A. Überprüfen Sie die Datenbank und stellen Sie sicher, dass die EOMS-Datenbank keine ungültigen Einträge enthält und dass das Datenbankschema alle benötigten Tabellen enthält. Gegebenenfalls müssen Sie die Datenbank neu anlegen, z.B. per vorgefertigtem Skript für ihren Datenbanktyp. Wenden Sie sich an den Support, falls Sie Hilfe benötigen.

Sie können sich nicht am EOMS-Core-Monitor anmelden, weil Ihre Anmeldedaten nicht akzeptiert werden.

Mögliche Ursachen:

A. Sie versuchen, sich mit ungültigen Anmeldedaten anzumelden.

B. Die Datenbank, in der die Passwörter gespeichert werden, ist beschädigt.

Lösung:

A. Stellen Sie sicher, dass Sie gültige Anmeldedaten verwenden und melden Sie sich zur Not mit den Standardbenutzer 'eoms' mit Passwort 'eoms' an. Falls Sie den Standardbenutzer geändert haben, müssen Sie *B.* a nwenden.

B. Möglicherweise ist die EOMS-Core-Datenbank, in der auch Passwörter gespeichert werden, beschädigt. Überprüfen Sie die Datenbank unter /conf (z.B. unter derby mit dem Tool ***ij)*** und stellen Sie sicher, dass der Datenbankserver gestartet und erreichbar ist *(siehe Datenbankkonfiguration)*. Spielen Sie gegebenenfalls ein Backup der Datenbank ein. Haben Sie keine Sicherungen der Datenbank, so müssen Sie als letzte Maßnahme die Datenbank neu aufsetzen.

Der Anmeldevorgang wird mit "Could not connect to server" abgebrochen.

Mögliche Ursachen:

A. Es wurde keine EOMS-Core-Datenbank angelegt oder die vorhandene Datenbank ist beschädigt.

B. Die Datenbank ist nicht erreichbar / Der Datenbankserver ist nicht gestartet.

C. Es liegt ein sonstiges Problem vor, z.B. ein Konfigurationsfehler.

Lösung:

A. Stellen Sie sicher, dass Sie die Datenbank korrekt angelegt und nach *%EOMS_HOME%/data/* kopiert haben *(für derby siehe hier).*

B. Möglicherweise kann das EOMS-Core nicht korrekt auf die Datenbank zugreifen. Stellen Sie sicher, dass die Datenbankkonfiguration in der *eoms.invoker.properties* korrekt ist. Eventuell benötigt das EOMS-Core während der Laufzeit einen gestarteten Datenbankserver, um auf Datenbanken zuzugreifen. Stellen Sie sicher, dass der entsprechende Datenbankserver gestartet wurde, bevor das EOMS-Core versucht, auf die Datenbank zuzugreifen. Für derby starten Sie den Datenbankserver mit dem Befehl "***java -jar %DERBY_HOME%/lib/derbyrun.jar server start***".

C. Wurde der Fehler nicht durch Datenbankprobleme verursacht, kann eine Vielzahl an Ursachen verantwortlich sein. Wir bitten Sie in diesem Fall Kontakt zu unserem Support aufzunehmen.

Fehler beim Start eines EOMS-Workers

Der Worker kann den EOMS-Core-Host nicht finden.

Mögliche Ursachen:

A. Das EOMS-Core ist nicht gestartet oder nicht erreichbar.

B. Die Workerkonfiguration enthält falsche Adressdaten.

Lösung:

A. Stellen Sie sicher, dass das EOMS-Core gestartet und erreichbar ist (z.B. durch "netstat" oder "ping"). Stellen Sie sicher, dass keine Firewall den Zugang blockt.

B. Überprüfen Sie in ***conf/spring/eoms.invoker.client.properties*** die Einträge ***eoms.invoker.host*** und ***eoms.invoker.port*** und stellen Sie sicher, dass ***eoms.invoker.host*** die richtige IP-Adresse (die IP des Hosts, auf dem das EOMS-Core läuft) und ***eoms.invoker.port*** den richtigen Port (in der server.xml des EOMS-Core festgelegten, Standard: 8080) zugewiesen bekommt.

Der Worker kann die Ressource vom Auftrags-System nicht empfangen: "Can not fetch resource"

Mögliche Ursachen:

A. Die in der Konfiguration des Workers eingestellte Übermittlungsmethode (indirekt / direkt) stimmt nicht mit den Einstellungen (bzw. übergebenen Parametern) des Auftrags-Systems überein oder der Worker kann die Datenübertragung zum EOMS-Core nicht erfolgreich abschließen, weil er sich nicht erfolgreich mit dem System / Auftragssystem verbinden kann.

Lösung:

A. Überprüfen Sie beim OMS-Worker, ob die Aufrufzeile im Spooler (bzw. im Auftrags-System) alle für die in der ***eoms.invoker.client.properties*** konfigurierte Übertragungsmethode benötigten Parameter enthält. Stellen Sie dann sicher, dass die gewählte Übertragungsmethode (ActiveMQ oder direkt) richtig konfiguriert und verfügbar ist. Stellen Sie außerdem sicher, dass der Worker über alle Berechtigungen zur Übertragung verfügt und die Verbindungskonfiguration (sowohl im Core als auch im Worker) korrekt ist, also die richtigen Ports und Hostadressen angegeben sind.

Beim Start des Workers tritt eine java.exception auf

Mögliche Ursachen:

Dies kann verschiedene Ursachen haben.

Lösung:

Stellen Sie zunächst sicher, dass Worker und Core ordnungsgemäß konfiguriert sind und vergleichen Sie die Logeinträge zum Zeitpunkt des Fehlers. Kontaktieren Sie gegebenenfalls unseren Support.

Tutorials

Hier finden Sie weitergehende Materialien, die Ihnen das Arbeiten mit dem EOMS erleichtern sollen.

Folgende Ressourcen stehen zur Verfügung:

Bitte beachten Sie, dass diese Seite im Aufbau ist und zur Zeit keine Tutorials zur Verfügung stehen.

Screenshots

Videos

JavaScript

JavaScript (kurz **JS**) ist eine Skriptsprache, die ursprünglich für dynamisches HTML in Webbrowsern entwickelt wurde, um Benutzerinteraktionen auszuwerten, Inhalte zu verändern, nachzuladen oder zu generieren und so die Möglichkeiten von HTML und CSS zu erweitern. Heute findet JavaScript auch außerhalb von Browsern Anwendung, so etwa auf Servern und in Microcontrollern.[1][2]

Der als **ECMAScript** (ECMA 262) standardisierte Sprachkern von JavaScript beschreibt eine dynamisch typisierte, objektorientierte, aber klassenlose Skriptsprache. Sie wird allen objektorientierten Programmierparadigmen unter anderem auf der Basis von Prototypen gerech t. In JavaScript lässt sich objektorientiert und sowohl prozedural als auch funktional programmieren. (laut Wikipedia)

JavaScript ist einfach zu erlernen und zu programmieren. Mit der Suchfunktion des Brwosers finden Sie hier schnell Anleitungen und Videos im Internet um mit kleinen Funktionen die Leistungsfähigkeit von RCML-Scripts zu erweitern. Hier einige gute Beispiel für JavaScript Tutorials:

- http://www.w3schools.com/js/
- http://www.kostenlose-javascripts.de/tutorials/

Screenshots

Bitte beachten Sie, dass diese Seite im Aufbau ist und zur Zeit keine Screenshots zur Verfügung stehen.

Videos

Eingeschränkte Verfügbarkeit von Videos

Leider ist die Verfügbarkeit von Videos über das EOMS z.Z. eingeschränkt.

Dies liegt einerseits daran, dass wir an der Erstellung arbeiten, andererseits vor allem auch daran,

dass die benötigten Bandbreiten zur verzögerungsfreien Anzeige von Videos derzeit nicht zur Verfügung stehen.

Wir arbeiten an einer Lösung und bitten dies zu entschuldigen.

Glossar

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

ACTIVATED (Status)

- Zeigt an, dass der Job momentan durch einen Worker bearbeitet wird.

ActiveMQ

- Messaging-Server für indirekte Kommunikation zwischen den einzelnen Systemen.

Adobe Flash Flex

- RIA-Framework. Basis der GUI des EOMS-Core-Monitor.

AKI-Worker

- Worker zur Interaktion mit AKI-Spoolingsystemen. Leitet SAP-Datenströme direkt an AKI-Systeme weiter und verhält sich dabei a Quelle wie ein SAP-System.
  ⚠ Wird in Version 1.3 nicht mehr unterstützt!

Apache Tomcat

- Webserver für Java Servlets. Dient als Basis für das EOMS-Core.

Attribut (RCML)

- Beschreibt ein Element näher.

Auftrags-System

- System, das an das EOMS angebunden ist, mit ihm interagiert und Daten austauscht.

nach oben

B

Binär-Paket

- Archivdatei, die ausführbare Programme für die Dokumentenaufbereitung im Redaktions-System enthält.

nach oben

C

Connector (Tomcat)

- Spezifikation in der server.xml des Tomcat, die einen Service an einen Port bindet.

Consumer

- Thread auf einem Worker.

nach oben

D

DERBY_HOME

- Umgebungsvariable, die auf das Homeverzeichnis des EOMS-Core zeigt. Muss gesetzt werden, wenn Tomcat als Applikations-S verwendet wird.

derby-Datenbank

- Standarddatenbanksystem bei Auslieferung des EOMS. Alternativen: MySQL und MSSQL.

docxworld

- Service zur Datenverarbeitung von profiforms, siehe hier.

nach oben

E

Element (RCML)

- Basisbausteine in RCML. Syntaktisch XML-Elemente.

eoms.invoker.client.properties

- Konfigurationsdatei des EOMS-Workers.

eoms.invoker.authentication

- Enthält Benutzerdatensätze für die Kontenverwaltung des EOMS-Core.

eoms.invoker.properties

- Konfigurationsdatei des EOMS-Core.

EOMS-Input Server

- Datenempfangsschnittstelle in docxworld.

EOMS-Input-Worker

- Worker zur Einlieferung von Daten in docxworld. Stellt u.a. Verbindung zum EOMS-Input Server her.

nach oben

F

Fileshare

- Veraltete Datenaustauschmethode, bei der der Worker direkten Zugriff auf ein Datenverzeichnis des Spoolers hat.

nach oben

G

GUI

- Steht für Grafische Oberfläche.

nach oben

H

Health Monitor

- Monitoring im UI-Client, der Echtzeit- und Verlaufsstatistiken übersichtlich darstellt.

HTTPS

- Sicheres Übertragungsprotokoll für Webanwendungen. Sollte wenn möglich für Spoolerkommunikation statt reinem http verwende werden.

nach oben

I

nach oben

J

JAVA_HOME

- Umgebungsvariable, die auf das Homeverzeichnis der JRE zeigt. Muss für das EOMS gesetzt sein.

JavaScript

- Skriptsprache. Kann in RCML eingebettet werden.

JMS

- Java Messaging Service. Bildet die Grundlage von ActiveMQ.

Job

- Verarbeitungseinheit.

Job-ID

- Eindeutige Identifikationskennung für Jobs.

nach oben

K

Kommunikation, direkte

- Läuft die Kommunikation zwischen EOMS-Core, Worker und Auftrags-System direkt über HTTP, spricht man von direkter Kommunikation.

Kommunikation, indirekte

- Läuft die Kommunikation zwischen EOMS-Core, Worker und Auftrags-System über den Messaging-Server ActiveMQ (JMS), spric von indirekter Kommunikation.

nach oben

L

List of Tasks

- Beinhaltet die Prozesse, die ein Worker ausführen kann.

localhost

- enstpricht 127.0.0.1 oder dem aktuellen Rechner.

Logdatei

- Logdateien enthalten Ablaufinformationen (Fehler, Warnungen, Informationen) über Programmausführungen und sollten bei Prob zu Rate gezogen werden.

nach oben

M

Messaging-Server

- Siehe JMS und ActiveMQ.

MSSQL

- SQL-Implementation von Microsoft. Wird vom EOMS unterstützt und kann als Datenbanksystem für das EOMS-Core verwendet v

MySQL

- SQL-Implementation. Wird vom EOMS unterstützt und kann als Datenbanksystem für das EOMS-Core verwendet werden.

nach oben

N

nach oben

O

OMS-Worker

- Worker zur Verarbeitung von Jobs aus dem Spooler.

nach oben

P

Produktions-Paket

- Archivdatei des Redaktions-Systems, das Dateien zur Dokumentenaufbereitung enthält.

Prozess (Process)

- In sich geschlossene Verarbeitungssequenz, ein Auftrag (Job) für das EOMS.

Prozess-Pool

- Zwischenpuffer des EOMS, aus dem Prozesse an die Worker vergeben werden.

nach oben

Q

nach oben

R

RCML

- Rich Client Markup Language. Beschreibungssprache für Worker-Prozesse, kann auch Skriptsprachen (JavaScript) enthalten.

Redaktions-System

- Siehe R-S-Dokumentation.

REST

- Web-Schnittstelle, mit der Sie über URI's Informationen abrufen und einfache Befehle ausführen lassen können.

Return-Code (RCML)

- Rückgabewert eines Prozesses / Elements, das seinen Status angibt.

nach oben

S

SCHEDULED (Status)

- Zeigt an, dass der Job noch nicht an einen Worker vergeben wurde und auf Verarbeitung wartet.

server.xml

- Konfigurationsdatei des Apache Tomcat.

Spooler

- Siehe Spooler-Dokumentation.

nach oben

T

Tag (RCML)

- wie Element (RCML).

TERMINATED (Status)

- Zeigt an, dass der Job beendet wurde.

nach oben

U

nach oben

V

nach oben

W

WORK

- Standard-Arbeitsverzeichnis des Workers. Enthält die Zwischenspeicherung der Jobdaten.

Worker

- Verarbeitungseinheit, die Jobs vom EOMS-Core entgegennimmt und verarbeitet.

WORKER

- Standard-Verzeichnis des Workers für temporäre Zwischenspeicherung empfangener Ressourcen.

nach oben

X

nach oben

Y

nach oben

Z

Weiterfuehrende Informationen

Die weiterführenden Informationen gliedern sich in folgende Unterkapitel (bitte auf den jeweiligen Link klicken):

Online-Archiv dieses Produkts

Ältere Versionen dieses Produkts in den Online-Dokumentationen

- EOMS 1.2.X

Ergaenzende Online-Dokumentationen

In diesem Bereich finden Sie Dokumentionen zu Aspekten oder korrespondierenden Software-Systemen des EOMS:

Datum	Link	Typ	Beschreibung
26.07.2011	Redaktions-System	Online	Vom R-S können über die EOMS-Worker Produktions-Pakete abgerufen werden, welche für die Aufbereitung von Druckdokumenten verwendet werden.
12.08.2014	Spooler 3.8	Online	Der Spooler kann als Auftrags-System für das EOMS verwendet werden.

ZUGFeRD-Modul

Sitemap

Was ist neu in Version 1.3

Einfuehrung

- Aufbau des EOMS
- Core - Worker
- Der Prozess-Pool
- EOMS als Basis fuer das Redaktions-System
- Hinweise zur Online-Dokumentation
    - Verwendete Symbole

Nutzer-Dokumentation

- Systemstart
- EOMS-Core
    - Erste Schritte
        - Anmelden
        - Nutzeroberflaeche und Navigation
            - Die obere Informationsleiste
            - Die Navigationsleiste
    - Bedienung
        - Prozesse
            - Zaehler Prozess-Tab
            - Monitor - Prozess-Tab
            - Verlauf - Prozess-Tab
        - Worker-Monitor
        - System-Monitor
            - Dashboard - System-Monitor-Tab
            - Verlauf - System-Monitor-Tab
        - Ueber (Eigenschaften)
    - Konfiguration und Struktur
- EOMS-Worker
    - Workertypen
        - OMS-Worker
        - EOMS-Input-Worker
    - Konfiguration und Struktur
    - RCML
- Fehlerbehandlung

Administrator-Dokumentation

- RCML-Elemente
  - <process>
    - <delete>
    - <destination>
    - <docxworld-fetch-production-environment>
      - <docxworld-contract>
      - <link-name>
      - <binary-bundles-home>
      - <production-bundles-home>
      - <merge-home>
      - <runtime-home>
    - <eoms-input-query-data>
    - <eoms-input-query-status>
    - <eoms-input-submit>
    - <error>
    - <exec>
      - <param>
      - <commandline>
      - <result>
    - <fetch-production-bundle>
    - <fetch-production-environment>
    - <fetchresource>
    - <if> - <else>
    - <message>
    - <releaseresource>
    - <rw-response>
    - <sleep>
    - <switch>
      - <case>
    - <update-variable>
    - <upload>
    - <while>
    - <workdir>
  - <script>

- Anbindung Auftrags-Systeme
  - Spooler Anbindung
- REST
- Typische Fehler und Fehlerbehandlung

Tutorials

- JavaScript
- Screenshots
- Videos

Glossar

Weiterfuehrende Informationen

- Online-Archiv dieses Produkts
- Ergaenzende Online-Dokumentationen
- Sitemap
- Download der Dokumentation

Download der Dokumentation

Bitte beachten Sie unsere rechtlichen Hinweise, bevor Sie die Dateien herunterladen!

Die Dateien entsprechen dem Datum in der Spalte "Erstellungsdatum". Die Online-Dokumentation ist nur zu diesem Zeitpunkt der Erstellung aktuell.

Wir weisen Sie darauf hin, dass interaktive Multimedia-Inhalte in der Online-Dokumentation im HTML- und PDF-Format nicht angezeigt werden können.

Format	Erstellungsdatum	Größe	Download-Datei	Hinweis
HTML	14.12.2015	31 MB	HTML-Datei	Nach dem Herunterladen der ZIP-Datei extrahieren Sie diese bitte. Anschließend öffnen Sie bitte die "index.html"-Datei in dem extrahierten Ordner. Es öffnet sich Ihr Browser und Sie können die Dokumentation lesen.
PDF	14.12.2015	2,7 MB	PDF-Datei	