



Software- und Organisations-Service GmbH

Job Scheduler

Event Verarbeitung

Technische Information
12. Oktober 2009

Impressum

Software- und Organisations-Service GmbH
Giesebrechtstr. 15
10629 Berlin
Deutschland

Telefon +49 30 864790-0

Telefax +49 30 8613335

Mail info@sos-berlin.com

Web <http://www.sos-berlin.com>

Letzte Änderung: 25.09.2009

1	Einleitung.....	4
1.1	Anwendungsbeispiele.....	5
2	Ereignisverarbeitung.....	7
2.1	Event Generator	8
2.2	Event Processor	15
2.3	In einer Job-Kette Ereignisse prüfen	17
3	Synchronisieren von Job-Ketten	18
4	Event Handler.....	19
4.1.1	Eigene Event Handler per Script	20
4.1.2	XSL Event Handler	22
4.1.3	XML Event Handler.....	23
5	Events überwachen.....	33
6	Ereignisverarbeitung einrichten.....	37
7	Beispiele für Ereignisbehandlungsroutinen.....	39
7.1	Split and Merge.....	39
7.2	Job A und Job B auf Job Scheduler 1, danach Job C auf Job Scheduler 2.....	40
7.3	Job A ist bis 5:00 Uhr nicht gelaufen	40
7.4	Benachrichtigung wenn bis 17:00 Uhr Datei nicht eingetroffen ist.....	41
8	Anhang	42

1 Einleitung

Der Job Scheduler verarbeitet standardmäßig Job-Ketten, die Jobs in einer vorhersehbaren Abfolge starten. Diese Serialisierung von Jobs ist für viele Anwendungsfälle zutreffend. Falls Jobs in Abhängigkeit nicht nur von einem, sondern von mehreren anderen Jobs oder von externen Ereignissen gestartet werden sollen bzw. von Jobs, die in anderen Job Scheduler Instanzen gestartet wurden, dann sind diese dynamischen Abhängigkeiten nicht durch Konfiguration von Job-Ketten abbildbar.

Job Scheduler stellt hierfür eine Lösung bereit, mit der Ereignisse ausgelöst werden können, die zentral verarbeitet werden. Ereignisse unterschiedlicher Herkunft werden gemeinsam ausgewertet und z.B. entsprechende Job-Starts oder Aufträge initiiert.

Ein Ereignis dient im Job Scheduler zur Steuerung des Ablaufs von Jobs und Aufträgen in Job-Ketten. Jobs und Job-Ketten werden gestartet, wenn ein oder mehrere bestimmte Ereignisse auftreten.

Ereignisse eignen sich besonders zur Implementierung von Abläufen, bei denen es mehrere Nachfolger und/oder mehrere Vorgänger im Ablauf gibt.

Für die Behandlung von Ereignissen werden drei unterschiedliche Methoden angeboten:

1. Verarbeitung in Event Handlern. Diese Lösung verfügt über die größten Freiheitsgrade in der Formulierung komplexer Bedingungen. Siehe --> „*XML Event Handler*“
2. Abfragen von Ereignissen und entsprechende Behandlung per Shell Script. In diesem Szenario können individuelle Shell Scripte sowohl Ereignisse erzeugen wie auch das Vorhandensein von Ereignissen prüfen. Siehe --> „*Eigene Event Handler im Script*“
3. Synchronisation mit dem Synchronizer-Job. Mit diesem Ansatz kann mit einem Minimum an Konfigurationsaufwand die Verarbeitung von mehreren Job-Ketten synchronisiert werden.
--> Siehe „*Synchronisieren von Job-Ketten*“

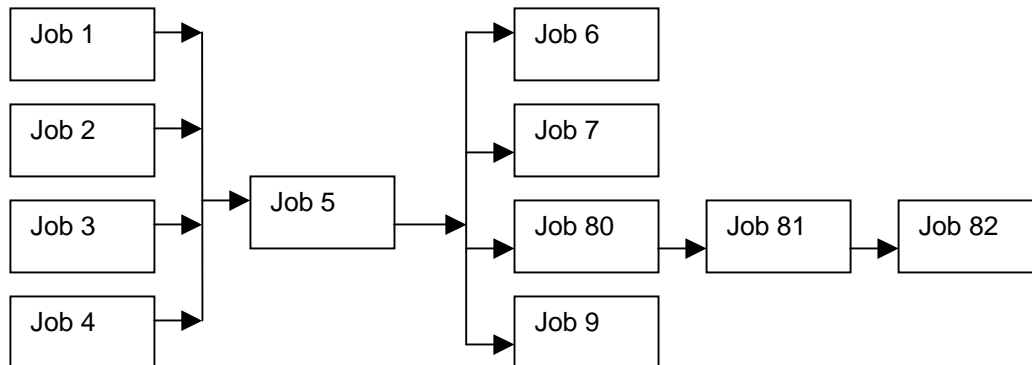
Anwendungsbeispiel: In Job-Kette A werden Dateiaufträge bearbeitet. In Job-Kette B werden Daten in eine Datenbank geschrieben. Es existiert ein persistenter Auftrag für die Job-Kette B, der täglich um 15:00 Uhr startet. Ein Eintrag in die Datenbank darf erst dann erfolgen, wenn eine bestimmte Datei für Job-Kette A eingetroffen ist. Die Schritte bis zum Synchronisationspunkt dürfen parallel bzw. in einer beliebigen Reihenfolge durchlaufen werden.

1.1 Anwendungsbeispiele

Hier als Beispiel einer *split & merge* Abfolge:

Eine Architektur wie in diesem Beispiel könnte z.B. gewünscht sein, weil man zeitintensive Jobs hat, die parallel ablaufen können. Damit die gesamte Verarbeitung beschleunigt wird, starten sie parallel (möglicherweise sogar auf verschiedenen Job Scheduler Instanzen).

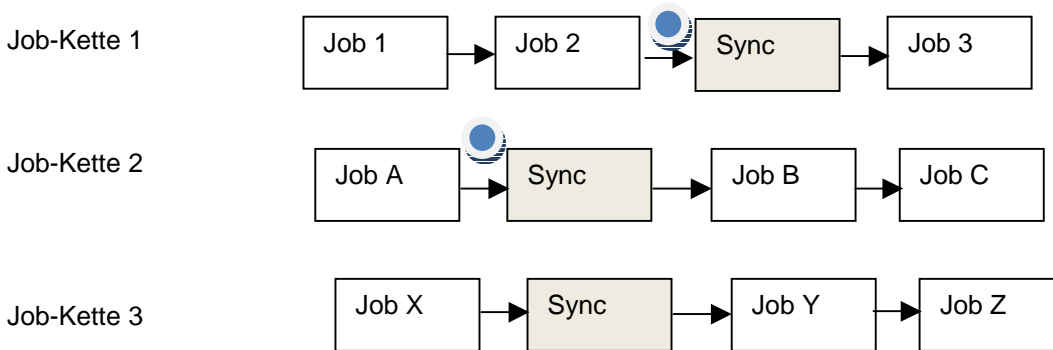
Ein weiterer Grund für solch eine Verarbeitung ist die zeitverzögerte Entstehung von Ereignissen. Bei einer Serialisierung in Job-Ketten würde die gesamte Verarbeitung ruhen.




In diesem Beispiel laufen die Jobs „Job 1“, „Job 2“, „Job 3“, „Job 4“ zu beliebigen Zeitpunkten ab. Erst wenn alle erfolgreich beendet sind, wird „Job 5“ gestartet. Anschließend sollen parallel „Job 6“, „Job 7“, „Job 9“ und die Job-Kette bestehend aus „Job 80“, „Job 81“, „Job 82“ ablaufen. Im weiteren Verlauf der Dokumentation wird darauf eingegangen, wie diese Abfolge mit der Ereignisverarbeitung realisiert werden kann.

Synchronisation mehrerer Job-Ketten

Es existieren mehrere Job-Ketten. An einem bestimmten Punkt innerhalb der Kette müssen diese synchronisiert werden, d.h. Aufträge werden erst dann weiter verarbeitet, wenn alle anderen beteiligten Job-Ketten ebenfalls Aufträge bis zu diesem Synchronisationspunkt verarbeitet haben.



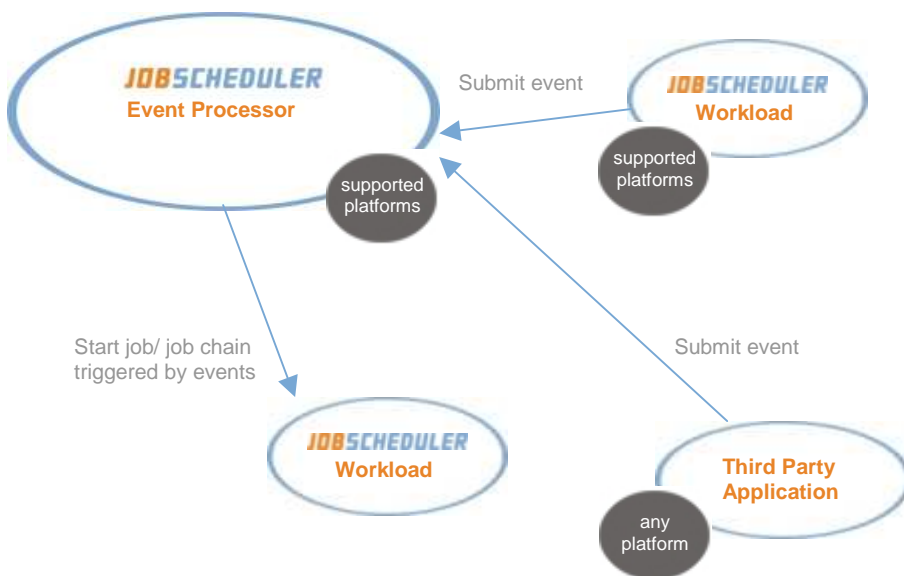
Die beiden Aufträge  warten an ihrem Synchronisationspunkt bis alle Job-Ketten einen Auftrag haben. Da Job-Kette 3 keinen Auftrag hat, wird die Verarbeitung in den Job-Ketten 1 und 2 angehalten.

2 Ereignisverarbeitung

In diesem Kapitel werden die Begriffe

- Event Generator
- Event Processor
- Event Handler

erläutert.



2.1 Event Generator

Damit Ereignisse verarbeitet werden können, müssen sie zunächst entstehen. Dabei gibt es unterschiedliche Ereignisquellen (Event Generators):

- Monitore eines Jobs in einer Job-Kette
- Standard-Jobs zur Erzeugung von Ereignissen
- Skripte
- Ereignisbehandlungsroutinen.

Allen Event Generators ist gemeinsam, dass sie einen Auftrag an den Event Processor via TCP/IP senden, der pro Auftrag einen Satz in die Ereignistabelle schreibt und sie außerdem in einer globalen → Job Scheduler Variablen hält. Der Auftrag enthält die Beschreibung des Ereignisses sowie die Aktion, die ausgeführt werden soll. Die Aktion `add` fügt das Ereignis hinzu. Die Aktion `remove` entfernt alle Ereignisse, die den angegebenen Attributen entsprechen (z.B. alle Ereignisse einer bestimmten Klasse).

Die Ereignisse können im Prinzip von jeder beliebigen Software produziert werden, indem sie einen Auftrag erzeugen und an den Supervisor Job Scheduler senden.

Ein Ereignis enthält folgende Attribute:

Name	Beschreibung	Default
event-id	Gemeinsam mit event-class eindeutig.	
exit-code	Der aktuelle Exit Code des laufenden Job Scripts (<script>)	Exit Code des Scripts
event-class	Zusammen mit event-id ergibt sich ein eindeutiger Bezeichner. Kann z.B. beim Entfernen von Ereignissen verwendet werden (Entfernen aller Ereignisse einer bestimmten Klasse)	
job-name	Der Name des gerade laufenden Jobs	Aktueller Job-Name
job-chain	Der Name der aktuellen Job-Kette	Aktuelle Job-Kette
order-id	Die Auftragskennung des aktuellen Auftrags	Aktuelle Auftragskennung (order id)
name	Der Name für das Ereignis	
creation-date	Zeitstempel der Erzeugung des Ereignisses	now
expiration-date	Zeitstempel, an dem das Ereignis automatisch abläuft. Fehlt diese Angabe, hat das Ereignis eine Lebensdauer entsprechend der Angabe im Event Processor (Default=24h)	00:00 des nächsten Tages

Mindestens die Event Class muss bei einem Ereignis angegeben werden.

Ereignisse werden von folgenden *Event Generators* erzeugt:

Event Generator Job

Der Standard-Job *JobSchedulerSubmitEventJob* kann verwendet werden, um ein Ereignis auszulösen. Das Ereignis wird an einen Supervisor Job Scheduler übertragen. Ist kein Supervisor konfiguriert, dann wird das Ereignis demselben Job Scheduler übergeben. Das Ereignis kann sowohl über Job-Parameter als auch über Auftragsparameter konfiguriert werden. Der Job kann in einer Job-Kette oder autark gestartet werden.

→ **Siehe Job-Dokumentation** *./jobs/JobSchedulerSubmitEventJob.xml*

Dieser Job erzeugt ein Ereignis mit der Klasse `myClass` und der ID `myId`.

```
<job name = "JobSchedulerSubmitEventJob"
      title = "Submit Events"
      order = "yes" >

  <description>
    <include file = "jobs/JobSchedulerSubmitEventJob.xml" />
  </description>

  <params>
    <param name = "scheduler_event_class" value = "myClass" />
    <param name = "scheduler_event_id" value = "myId" />
  </params>

  <script language = "java"
          java_class = "sos.scheduler.job.JobSchedulerSubmitEventJob" />
</job>
```

Beispiel: Das Ereignis "Die Uhrzeit 17:00 ist erreicht" soll ausgedrückt werden. Dazu wird ein Job konfiguriert, der als Startzeit 17:00 verwendet.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job name="sample"
  title="Submit Event &quot;17:00 Uhr&quot;">
  <description>
    <include file="jobs/JobSchedulerSubmitEventJob.xml"/>
  </description>
  <params>
    <param name="scheduler_event_class"
      value="myClass"/>
    <param name="scheduler_event_id"
      value="myId1"/>
  </params>
  <script language="java"
    java_class="sos.scheduler.job.JobSchedulerSubmitEventJob"/>
  <run_time>
    <period single_start="17:00"/>
  </run_time>
</job>
```

Der Job kann z.B. auch als Schritt innerhalb einer Job-Kette konfiguriert werden, um den Durchlauf eines Auftrags mit einem Ereignis zu dokumentieren. Die folgende Job-Kette besteht aus 3 Schritten. Wenn Schritt "100" abgeschlossen ist, soll ein Ereignis erzeugt werden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job_chain name="job_chain1"
  orders_recoverable="yes"
  visible="yes">
  <job_chain_node state="100"
    job="anyJob"
    next_state="200"
    error_state="error"/>
  <job_chain_node state="200"
    job="firstStepIsReady"
    next_state="300"
    error_state="error"/>
  <job_chain_node state="300"
    job="anyOtherJob"
    next_state="success"
    error_state="error"/>
  <job_chain_node state="error"/>
  <job_chain_node state="success"/>
</job_chain>

<?xml version="1.0" encoding="ISO-8859-1"?>
<job name="firstStepIsReady"
  title="Submit Event &quot;Step 100 is ready&quot;"
  order="yes">
  <description>
    <include file="jobs/JobSchedulerSubmitEventJob.xml"/>
  </description>
  <params>
    <param name="scheduler_event_class"
      value="myClass"/>

    <param name="scheduler_event_id"
      value="myId2"/>
  </params>
  <script language="java"
    java_class="sos.scheduler.job.JobSchedulerSubmitEventJob"/>
  <run_time/>
</job>
```

Event Generator Monitor

Beinhaltet die gleiche Funktionalität wie ein Job, allerdings implementiert als Monitor zur Verwendung mit beliebigen anderen Jobs. Siehe die Job-Dokumentation in *./jobs/JobSchedulerSubmitEventMonitor.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job title="Erzeugt einen Event mit einem Monitor"
      name="job2_event_by_monitoing">
  <params>
    <param name="scheduler_event_id"
           value="2" />

    <param name="scheduler_event_class"
           value="monitorEvent" />
  </params>
  <script language="javascript">
    <![CDATA[
function spooler_process(){
  spooler.log.info("Here is a event coming from a monitor");
  return false;
}
    ]]>
  </script>
  <monitor name="create_event"
          ordering="0">
    <script java_class="sos.scheduler.job.JobSchedulerSubmitEventMonitor"
           language="java" />
  </monitor>
  <run_time/>
</job>
```

Event Generator Script

Für Unix und Windows steht das Script `jobscheduler_event.sh` bzw. `jobscheduler_event.cmd` zur Verfügung. Der Aufruf dieses Scripts erzeugt ein Ereignis. Das Script kann z.B. in bestehende Shell Scripte aufgenommen werden oder als separater Schritt in einer Job-Kette bzw. als standalone Job aufgerufen werden.

Beispiel für einen Shell Job, der ein Ereignis der Klasse `example` erzeugt:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job order="yes"
  name="file_job">
  <script language="shell">
    <![CDATA[
      @echo off
      @rem submit event to Supervisor Job Scheduler
      %SCHEDULER_HOME%/bin/jobscheduler_event.cmd -x %ERRORLEVEL% -e example
    ]]>
  </script>
  <run_time/>
</job>
```

Folgende Parameter können angegeben werden

`jobscheduler_event.sh - submit event to Job Scheduler`

-x	exit-code	Um den Exit-Code eines vorhergehenden Kommandos zu substituieren wird die Zeichenfolge \$? verwendet.
-e	event-class	Ein allgemeiner Name für eine Reihe von Ereignissen. Damit ist es dem Event Handler möglich, mehrere Ereignisse einer Klasse simultan zu behandeln. Z.B. <code>daily_closing</code> könnte eine Klasse für Ereignisse sein, die starten sollen, wenn die Tagesverarbeitung beendet ist.
-i	event-id	Ein Bezeichner für das Ereignis. Die Event Handler können auf Ereignisse mit einer bestimmten ID reagieren. Die ID ist pro Event Klasse eindeutig.
-c	job-chain	Der Name der Job-Kette. Falls leer, ist es der Name der aktuellen Job-Kette.
-o	order-id	Falls der Job in einer Job-Kette ausgeführt wird, kann dessen Order ID verwendet werden, um das Ereignis zu identifizieren. Der Default ist der Wert der Umgebungsvariablen <code>SCHEDULER_ORDER_ID</code> .
-j	job-name	Der Name des aktuellen Jobs. Der Default ist der Wert der Umgebungsvariablen <code>SCHEDULER_JOB_NAME</code> . Die Umgebungsvariable wird automatisch vom Job Scheduler gesetzt
-h	workload-job-scheduler-host	Der Host der lokalen Job Scheduler Workload Instanz. Der Default ist der Wert der Umgebungsvariablen <code>SCHEDULER_HOST</code> .
-p	workload-job-scheduler-port	Der Port der lokalen Job Scheduler Workload Instanz. Der Default ist der Wert der Umgebungsvariablen <code>SCHEDULER_TCP_PORT</code> .
-s	supervisor-job-scheduler-host	Der Host der Job Scheduler Supervisor Instanz. Der Default ist der Wert der Umgebungsvariablen <code>SCHEDULER_SUPERVISOR_HOST</code> . Workload Instanzen registrieren automatisch eine Supervisor Instanz, um die Job Konfigurati-

		onen zu synchronisieren. Die Supervisor Instanz empfängt Ereignisse, führt die Event Handler aus und stößt die Ausführung von Jobs und Job-Ketten an.
-r	supervisor-job-scheduler-port	Die Port Nummer der Job Scheduler Supervisor Instanz. Als Standard wird der Wert der Umgebungsvariablen SCHEDULER_SUPERVISOR_PORT verwendet.
-v	supervisor-job-chain	Der Name der Job-Kette in der Job Scheduler Supervisor Instanz, die den Event Processor implementiert. Default ist scheduler_event_service.
-w	what	Ereignisse können hinzugefügt (add), gelöscht (remove) und geprüft (check) werden. Die Default-Aktion ist add. Mit <i>check</i> wird geprüft, ob ein oder mehrere bestimmte Ereignisse vorhanden sind. Es muss mindestens eine der Optionen -i -e -x oder -a angegeben sein, die Ereignisse bestimmen, nach denen gesucht wird.
-t -td	expiration-date	Ereignisse können automatisch nach Zeitablauf ungültig werden. Ihre Lebensdauer wird durch einen konfigurierbaren Wert im Event Processor bestimmt. Soll ein bestimmter Zeitpunkt angegeben werden, zu dem das Ereignis abläuft, kann dies mit diesem Parameter erfolgen. Der Zeitpunkt wird im ISO Datumsformat angegeben yyyy-MM-dd hh:mm:ss.. Der Wert <i>never</i> sorgt dafür, dass das Ereignis niemals abläuft. Die Optionen expiration-date, -cycle und -period können nicht gleichzeitig verwendet werden.
-tc	expiration-cycle	Bestimmt eine Uhrzeit des aktuellen bzw. des folgenden Tages, zu dem die Lebensdauer des Ereignisses abläuft. Der Zyklus ist im Format hh:mm[:ss] anzugeben. Kann nicht mit expiration-date oder -period verwendet werden.
-tp	expiration-period	Bestimmt einen Zeitraum nach dem die Lebensdauer des Ereignisses abläuft. Die Periode ist im Format hh:mm[:ss] anzugeben. Kann nicht mit expiration-date oder -cycle verwendet werden.
-d	name=value	Zusätzliche Parameter für den Event Handler. Parameter werden durch Name/Wert Paare erzeugt.
-a	xpath-expression	Wenn -w angegeben ist, werden alle Ereignisse gefunden, die dem angegebenen XPath-Ausdruck entsprechen. Es können beliebig komplexe Ausdrücke angegeben werden. Es stehen alle Attribute eines Ereignisses zur Verfügung. Dadurch können beliebig komplexe Abfragen erfolgen, die mit den Optionen -e -i und -x nicht möglich wären.
allowed-exit-code	...	Eine Liste von Exit Codes, die keinen Fehler signalisieren, sondern als erfolgreich angesehen werden sollen.

jobscheduler_event.sh sendet ein Ereignis zum Event Processor der Job Scheduler Supervisor Instanz. Die Ereignisse werden vom Event Handler verarbeitet und führen zu einer Ausführung von

Jobs oder Job-Ketten. Event Handler beinhalten Bedingungen über das Vorhandensein oder das Fehlen von bestimmten Ereignissen. Entsprechend dieser Bedingungen werden Jobs oder Aufträge gestartet

Fall kein Supervisor verfügbar ist, werden die Ereignisse in einer lokalen Datei gespeichert.

`./logs/scheduler.events` Diese Ereignisse werden vom Job `job_scheduler_dequeue_events` verarbeitet, sobald der Supervisor wieder zur Verfügung steht.

Beispiel

Erzeugen eines Ereignisses mit einer minimalen Ausstattung an Parametern:

```
./jobscheduler_event.sh -x $? -e daily_closing -i my_id -j my_job
```

Bestimme Hostnamen und Port eines anderen Job Scheduler Supervisors und definiere ein Parameter `proc1` mit dem Wert 42:

```
./jobscheduler_event.sh -x $? -e daily_closing -i my_id -j my_job -s master -r 4444 -d "proc1=42"
```

Environment

<code>SCHEDULER_HOME</code>	Installationsverzeichnis der Job Scheduler Workload Instanz.
<code>SCHEDULER_HOST</code>	Host der Job Scheduler Workload Instanz.
<code>SCHEDULER_TCP_PORT</code>	Port der Job Scheduler Workload Instanz.
<code>SCHEDULER_JOB_CHAIN</code>	Name der job chain die aktuell ausgeführt wird.
<code>SCHEDULER_ORDER_ID</code>	Id des aktuellen Auftrages
<code>SCHEDULER_SUPERVISOR_HOST</code>	Hostname der Job Scheduler Supervisor Instanz.
<code>SCHEDULER_SUPERVISOR_PORT</code>	Port der Job Scheduler Supervisor Instanz.

Diagnostics

Folgende Fehlermeldungen werden ggf. nach `stderr` geschrieben:

<code>ERROR-001: no host has been specified</code>	Der Host der Job Scheduler Supervisor Instanz wurde nicht angegeben
<code>ERROR-001: no port has been specified</code>	Der Port der Job Scheduler Supervisor Instanz wurde nicht angegeben.
<code>ERROR-010: could not connect to host</code>	Der angegebene Host oder IP ist ungpütig oder es ist kein Job Scheduler unter dieser Adresse erreichbar

Event Generator Event Handler (→ 2.3)

In Event Handlern können neue Ereignisse erzeugt werden. Dazu müssen diese Event Handler das Element `<add_event>` aufführen.

```
<commands>
  <add_event>
    <event event_id="7" event_class="sample" event_title="Neues Ereignis" />
  </add_event>
</commands>
```

2.2 Event Processor

Der Event Processor hat zwei Aufgaben.

1. Sammeln der Ereignisse und abspeichern in der Datenbank sowie in einer Job Scheduler Variablen.
2. Ausführen von Ereignisbehandlungsroutinen (Event Handlern) und Auswerten von Bedingungen.

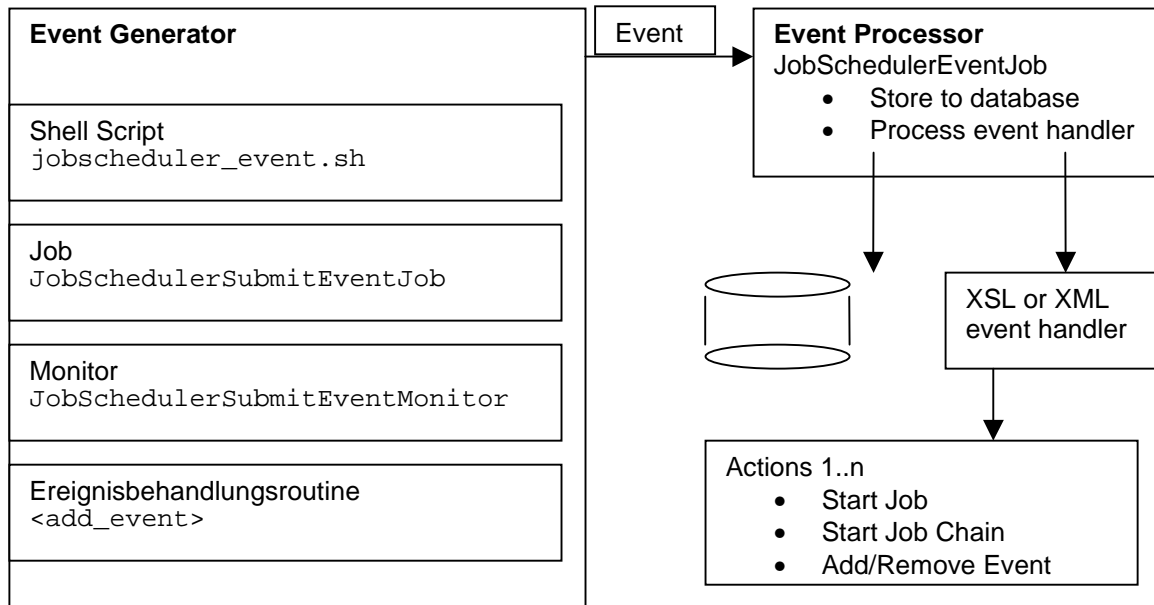
Ein Event Handler besteht aus:

- einem Regelwerk, bei dem das Vorhandensein von Ereignissen geprüft wird
- und einer Liste von Aktionen, die ausgeführt werden, wenn die Ereignisbedingungen zutreffen. Folgende Aktionen sind möglich
 - Starten eines Jobs
 - Starten einer Job-Kette
 - Erzeugen eines neuen Ereignisses
 - Entfernen von Ereignissen

Event Handler können in einem XML Format und als Stylesheets in XSL vorliegen. Ereignisbehandlungsroutinen im XML Format werden vom Job-Editor unterstützt und können dort bearbeitet werden.

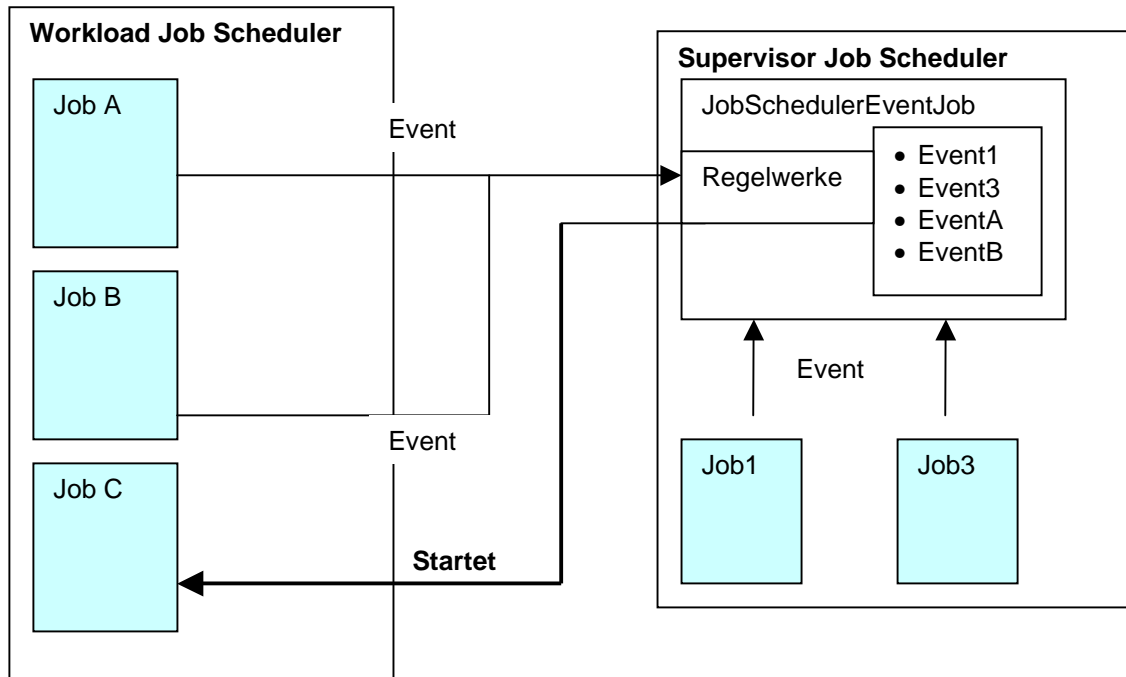
→ Siehe Kapitel 2.3

Event Generators erzeugen also Ereignisse. Diese werden im *Event Processor* gesammelt und bei jedem Eintreffen eines Ereignisses werden ein oder mehrere Event Handler ausgeführt. Bei der Ausführung der Event Handler werden Bedingungen geprüft, die das Vorhandensein eines oder mehrere Ereignisse testen. Im positiven Fall werden die zugehörigen Aktionen ausgeführt.



Der *Event Processor* besteht aus einer Job-Kette mit dem Job `scheduler_event_service` und der Job-Kette `scheduler_event_service`. Die Definition für einen Event Processor befindet sich in der Datei `./config/scheduler_event.xml`.

Die Job-Kette beinhaltet den Job `scheduler_event`. Die Parametrisierung dieses Jobs ist in der Job Dokumentation `scheduler/jobs/JobSchedulerEventJob.xml` beschrieben.



Überblick. Beispiel einer Architektur mit zwei Job Schemulern.

In einem Workload Job Scheduler starten die Jobs A und B. Die Jobs erzeugen jeweils ein Ereignis. Die Ereignisse werden vom Event Processor im Supervisor Job Scheduler verarbeitet, in dem die Regelwerke des Event Handlers ausgeführt werden. Im Supervisor Job Scheduler laufen die Jobs 1 und 3, die ebenfalls Ereignisse erzeugen. Der Event Processor startet z.B. Job C, wenn ein Regelwerk zu True evaluiert.

Ereignisse haben eine Lebensdauer, die beim Erzeugen des Ereignisses gesetzt werden kann. Standardmäßig laufen alle Ereignisse nach 24 Stunden Uhr ab. Innerhalb ihrer Lebensdauer können Ereignisse explizit in den Ereignisbehandlungsroutinen gelöscht werden. Alle Ereignisse werden in einer Datenbank gespeichert und sind nach einem Neustart des Job Schemulers wieder verfügbar.

2.3 In einer Job-Kette Ereignisse prüfen

Manchmal ist es notwendig innerhalb einer Job-Kette das Vorhandensein eines oder mehrere Ereignisse zu prüfen und von ihnen die Weiterverarbeitung abhängig zu machen. Zu diesem Zweck steht der Job *JobSchedulerExistsEventJob* zur Verfügung.

Dieser Job kann prüfen, ob bestimmte Ereignisse existieren. Dazu verarbeitet er Aufträge, die eine Spezifikation der Ereignisse enthalten. Je nachdem, ob passende Ereignisse existieren oder nicht, wird der Auftrag in den `next_state` oder in den `error_state` versetzt.

Die Spezifikation der zu prüfenden Ereignisse steht im Parameter `scheduler_event_spec`. Der Parameter gibt einen XPath Ausdruck an, der auf die XML Repräsentation der Ereignisse angewendet wird. Liefert der XPath Ausdruck ein Ergebnis, so wird der Auftrag in den `next_state` versetzt. Liefert der XPath Ausdruck kein Ergebnis, so wird der Auftrag in den `error_state` versetzt.

Ein Beispiel zur Verwendung dieses Jobs befindet sich im Anhang.

Die Job-Dokumentation kann unter `scheduler/jobs/JobSchedulerExistsEventJob.xml` eingesehen werden.

Beispiele:

```
//events/event[@event_class='foo']
```

Erfolgreich, wenn es ein Event der Event-Klasse "foo" gibt

```
//events[event/@event_class='foo' and event/@event_class='bar']
```

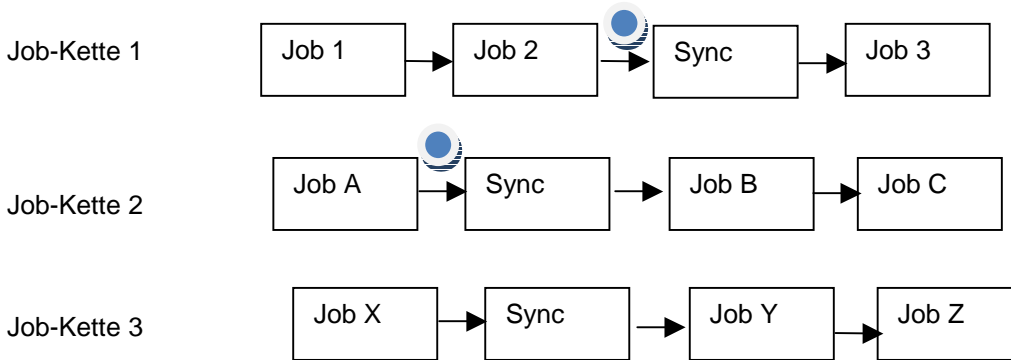
Erfolgreich, wenn es ein Event der Event-Klasse "foo" und ein Event der Event-Klasse "bar" gibt


```
//events[not(event/@event_class='foo')]
```

Erfolgreich, wenn es kein Event der Event-Klasse "foo" gibt

3 Synchronisieren von Job-Ketten

Ein Standardaufgabe, das Synchronisieren von Job-Ketten, kann vereinfacht ohne Events konfiguriert werden: Es existieren mehrere Job-Ketten. An einem bestimmten Punkt innerhalb der jeweiligen Kette müssen diese synchronisiert werden, d.h. Aufträge werden erst dann weiter verarbeitet, wenn alle anderen beteiligten Job-Ketten ebenfalls Aufträge an einem Synchronisationspunkt haben.



Die beiden Aufträge  warten an ihrem Synchronisationspunkt bis alle Job-Ketten einen Auftrag haben. Da Job-Kette 3 keinen Auftrag hat, wird die Verarbeitung in der Job-Kette 1 und 2 angehalten.

Es werden immer so viele Aufträge synchronisiert wie im Sync-Job angegeben (default=1). Kommt z.B. ein neuer Auftrag in die Job-Kette 2 (es sind dann 2 Aufträge in dieser Job-Kette) und anschließend ein Auftrag in Job-Kette 3, dann wird aus allen Job-Ketten jeweils nur ein Auftrag verarbeitet. Danach befindet sich noch ein Auftrag in Job-Kette 2 am Synchronisationspunkt.

Für dieses Verfahren können weitere Informationen unter http://jobscheduler.sourceforge.net/osource_scheduler_howto_split_merge_en.htm, nachgelesen werden

Konfiguration für den Job Sync

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<job order="yes"
  stop_on_error="no">
  <!--
  <params>
    <param name="ChainA_required_orders"
      value="2" />

    <param name="ChainB_required_orders"
      value="3" />
  </params>
  -->
  <script java_class="sos.scheduler.job.JobSchedulerSynchronizeJobChains"
    language="java" />
  <delay_order_after_setback setback_count="1"
    is_maximum="no"
    delay="00:02" />
  <delay_order_after_setback setback_count="200"
    is_maximum="yes"
    delay="00:02" />

  <run_time />
</job>
  
```

4 Event Handler

Event Handler beinhalten eine Abfrage nach Ereignissen. Alle vorhandenen Ereignisse werden in einer globalen Job Scheduler Variablen vorgehalten. Der Inhalt der Job Scheduler Variablen, d.h. die Liste der aktuell vorhandenen Ereignisse, kann in der Protokolldatei des Event Processors nachgeschlagen werden.

In der Datei *scheduler/config/factory.ini* kann der Log Level auf einen Wert `> debug2` eingestellt werden, damit erweiterte Ausgaben in die Protokolldatei *scheduler/logs/task.scheduler_event_service.log* geschrieben werden. Hierzu wird im Abschnitt `[spooler]` der Wert für `log_level` z.B. auf `debug3` gesetzt. Diese Einstellung wirkt für alle Jobs. Der Wirkungsbereich wird auf den Event Prozessor begrenzt, wenn der Wert für `log_level` im Abschnitt `[job_scheduler_event_service]` erfolgt.

In der Protokolldatei der Task (*scheduler/logs/task.scheduler_event_service.log*) des *Event Processors* wird die aktuelle XML Struktur ausgegeben. Die XML Struktur sieht z.B. so aus:

```
<events current_date="2008-05-16 11:14:01"
        expiration_date="2008-05-16 23:14:01">

<event created="2008-05-16 11:13:29"
        event_class="example"
        event_id="0"
        exit_code="0"
        expires="2008-05-16 23:13:30"
        job_chain=""
        job_name="simple_shell_job"
        order_id=""
        remote_scheduler_host="wilma"
        remote_scheduler_port="4444"
        scheduler_id="scheduler.supervisor"/>

<event created="2008-05-16 11:14:01"
        event_class="example"
        event_id="0"
        exit_code="0"
        expires="2008-05-16 23:14:01"
        job_chain="txt_chain"
        job_name="file_job"
        order_id="files/in/sample2.txt"
        remote_scheduler_host="wilma"
        remote_scheduler_port="4444" scheduler_id="scheduler.supervisor"/>
</events>
```

Der *Event Processor* ermittelt zunächst alle Event Handler und führt diese aus. Dabei kann es zwei Arten von Event Handler geben.

- XML Event Handler
- XSL Event Handler

Die XML Event Handler werden mit dem Job Editor erstellt. Sie definieren die Event Processing Rules und die dazugehörigen Aktionen.

Die XSL Event Hhandler werden mit einer XSLT Stylesheet Transformation verarbeitet. Das Ergebnis der Transformation sind Kommandos an einen oder mehrere Job Scheduler.

Alle Event Handler werden in einem zentralen Verzeichnis abgelegt. Der Name des Verzeichnisses steht im Job-Parameter *event_handler_filepath* des Event Processors.

4.1.1 Eigene Event Handler per Script

Ereignisse lassen sich in Shell Scripten abfragen. Dadurch ist es möglich, eigene Prüfungen in Shell Scripten zu implementieren und damit die Behandlung von Ereignissen mit Mitteln der Shell durchzuführen. Der Event Processor hat in diesem Fall lediglich die Aufgabe, die Ereignisse einzusammeln und zu speichern. Die Prüfung, ob ein bestimmtes Ereignis vorliegt, wird per Shell Script `jobscheduler_event.cmd` bzw. `jobscheduler_event.sh` durchgeführt. Dabei werden zwei Möglichkeiten der Abfrage angeboten:

1. Direkte Abfrage von Attributen `-e` und `-i` für Ereignisse
2. Abfrage mit XPath-Ausdrücken

Die Option `-w` wird auf den Wert `check` gesetzt. Zusätzlich können die Attribute angegeben werden, nach denen gesucht wird. Das Script schreibt die Anzahl der vorliegenden Ereignisse nach `stdout`. Unter Windows wird außerdem die Umgebungsvariable `%JOB_SCHEDULER_COUNT_EVENTS%` gesetzt.

Beispiele für Abfragen:

Wie viele Ereignisse der Klasse "foo" liegen vor?

```
call c:\scheduler\bin\jobscheduler_event.cmd -h localhost -p 4454 -w check -e foo
```

Wie viele Ereignisse der Klasse "foo" liegen vor (mit XPath)?

```
call c:\scheduler\bin\jobscheduler_event.cmd -h localhost -p 4454 -w check
-a "//events/event[@event_class='foo']"
```

Liegt jeweils ein Ereignis der Event Klasse "foo" und der Event Klasse "bar" vor?

```
call c:\scheduler\bin\jobscheduler_event.cmd -h localhost -p 4454 -w check
-a "//events[event/@event_class='foo' and event/@event_class='bar']"
```

Liegt kein Ereignis der Event Klasse "foo" vor?

```
call c:\scheduler\bin\jobscheduler_event.cmd -h localhost -p 4454 -w check
-a "//events[not(event/@event_class='foo')]"
```

Beispiel für Split & Merge

In diesem Beispiel werden innerhalb eines Scripts drei weitere Scripte parallel gestartet (Split). Erst wenn diese Scripte terminieren, wird die Verarbeitung fortgesetzt (Merge). Die gestarteten Scripte signalisieren ihre Terminierung durch das Erzeugen eines Ereignisses.

1. Haupt Script

Die Implementierung des Haupt Scripts wird in einem Job definiert. Die von dort gerufenen, weiteren Scripte müssen nicht im Job Scheduler als Job vorliegen.

```
@echo off
@echo hier Script A
@rem ----- Begin Split
@echo Jetzt erfolgt der Split
start b.cmd
start c.cmd
start d.cmd
@rem ----- Ende Split

@rem ----- Begin Merge
:merge
call c:\scheduler\bin\jobscheduler_event.cmd -h localhost -p 4444 -w check -e test >nul
@echo +%JOB_SCHEDULER_COUNT_EVENTS%+ Ereignisse der Klasse test gefunden
@rem Sleep 10 Sekunden
ping -n 11 localhost > nul
@rem Anzahl der vorliegenden Ereignisse ermitteln
if %JOB_SCHEDULER_COUNT_EVENTS% lss 3 goto merge

@rem ----- Ende Merge
```

```
@echo Merge ist vollbracht.
@rem Ereignisse löschen
call c:\scheduler\bin\jobscheduler_event.cmd -h localhost -p 4444 -w remove -e test
```

2.Script B (Script C und D entsprechend)

```
@echo off
@echo hier Script B
... (Verarbeitung)
@rem Ereignis erzeugen
call c:\scheduler\bin\jobscheduler_event.cmd -h localhost -p 4444 -w add -i B -e test
```

Das gleiche Beispiel kann mit 4 Jobs A,B,C,D konfiguriert werden. Alle Scripte werden als Jobs definiert. Die Implementierung von Job A sieht wie folgt aus.

```
<job>
  <script language="shell"><![CDATA[
    @echo off
    @echo hier Script A
    @rem ----- Begin Split
    @echo Jetzt kommt der Split
    call c:\sos\scheduler\bin\jobscheduler.cmd command "<start_job job='test/jobB'/">
    call c:\sos\scheduler\bin\jobscheduler.cmd command "<start_job job='test/jobC'/">
    call c:\sos\scheduler\bin\jobscheduler.cmd command "<start_job job='test/jobD'/">
    @rem ----- Ende Split

    @rem ----- Begin Merge
    :merge
    call c:\scheduler\bin\jobscheduler_event.cmd -w check -e test > nul
    @echo ++%JOB_SCHEDULER_COUNT_EVENTS%+ Ereignisse der Klasse test gefunden
    ping -n 11 localhost > nul
    if %JOB_SCHEDULER_COUNT_EVENTS% lss 3 goto merge

    @rem ----- Ende Merge
    @echo Merge ist vollbracht
    call c:\scheduler\bin\jobscheduler_event.cmd -w remove -e test
  ]]></script>
  <run_time/>
</job>
```

Die Implementierung von Job B und entsprechend Job C sieht so aus:

```
<job>
  <script language="shell"><![CDATA[
    @echo off
    @echo hier Script B
    call c:\scheduler\bin\jobscheduler_event.cmd -w add -i B -e test
  ]]></script>
  <run_time/>
</job>
```

4.1.2 XSL Event Handler

Alle Event Handler, deren Namen dem regulären Ausdruck im Job-Parameter *event_handler_filespec* entsprechen, werden ausgeführt.

Zusätzlich verarbeitet der Job Stylesheets für bestimmte Jobs, Job-Ketten oder Event Klassen nach der Namenskonvention:

- *[job_name].*.job.xsl* wird ausgeführt für Ereignisse des Jobs *[job_name]*
- *[job_chain_name].*.job_chain.xsl* wird ausgeführt für Ereignisse eines Jobs in der Job-Kette *[job_chain_name]*
- *[event_class].*.event_class.xsl* wird ausgeführt für Ereignisse der Klasse *[event_class]*

XSL Event Handler werden manuell mit einem Texteditor erfasst. Der Event Handler im Beispiel führt die Liste aller bekannten Ereignisse aus (s.o.).

Beispiel für einen Event Handler, der nach drei Ereignissen sucht: Ein Ereignis des Jobs *simple_shell_job* und jeweils ein Ereignis von zwei Job-Ketten.

```
<xsl:template match="events[
  event[@job_name='simple_shell_job'] and
  event[@job_chain='txt_chain'] and
  event[@job_chain='pdf_chain'] ]">
```

Wenn diese Ereignisse gefunden werden, dann werden zwei Kommandos an den Supervisor Job Scheduler gesendet. Das erste Kommando startet den Job *samples/events/done_job* im Workload Job Scheduler:

```
<xsl:call-template name="run_job">
  <xsl:with-param name="job">samples/events/done_job</xsl:with-param>
  <xsl:with-param name="host">localhost</xsl:with-param>
  <xsl:with-param name="port">4444</xsl:with-param>
</xsl:call-template>
```

Das zweite Kommando löscht die Ereignisse, um zu vermeiden, dass die Ereignisse ein weiteres Mal verarbeitet werden.

```
<remove_event><event event_class="example" /></remove_event>
```

4.1.3 XML Event Handler

Im Verzeichnis der Event Handler wird in der folgenden Reihenfolge nach Dateien gesucht:

- *[job_name].*.job.actions.xml* wird ausgeführt für Ereignisse des Jobs *[job_name]*
- *[job_chain_name].*.job_chain.actions.xml* wird ausgeführt für Ereignisse eines Jobs in der Job-Kette *[job_chain_name]*
- *[event_class].*.event_class.actions.xml* wird ausgeführt für Ereignisse der Klasse *[event_class]*
- *[name].actions.xml* wird ausgeführt

Wird das Ereignis z.B. durch den Job *job1* erzeugt und es existiert eine Datei mit dem Namen *job1.job.actions.xml*, dann wird diese verwendet.

XML Ereignisbehandlungsroutinen können mit dem Job Editor erfasst und bearbeitet werden. Um eine neue Ereignisbehandlungsroutine zu erstellen, klicken Sie auf den Menüpunkt „Neu“ und „Event Handler“.

XML Ereignisbehandlungsroutinen bestehen aus einem Regelwerk und einer Liste von Kommandos. Die Kommandos werden ausgeführt, wenn das Regelwerk zu TRUE evaluiert. Kommandos können für jeweils dieselben Job Scheduler Instanzen zu Gruppen zusammengefasst werden.

Beispiel:

Es gibt zwei Kommandogruppen: in der ersten Gruppe werden zwei Kommandos für den Job Scheduler *localhost:4454* definiert; in der zweiten Gruppe wird ein Kommando für den Job Scheduler *server:4444* definiert:

```
<commands>
  <command name="command_1" scheduler_host="localhost" scheduler_port="4454">
    <start_job job="job0" at="now" />
    <add_order job_chain="job_chain1" replace="yes" />
  </command>
  <command name="command_2" scheduler_host="server" scheduler_port="4444">
    <start_job job="job33" at="now" />
  </command>
</commands>
```

Regelwerk

Das Regelwerk besteht aus einer oder mehreren Ereignisgruppen. Eine Ereignisgruppe besteht aus einer Liste von Ereignissen, die mit einem booleschen Ausdruck verbunden werden. Die Ereignisgruppen werden evaluiert. Die Ergebnisse der Evaluierung der Ereignisgruppen werden untereinander mit einem booleschen Ausdruck verbunden. Dieses Ergebnis bestimmt, ob die Kommandos ausgeführt werden.

Verkürzte Schreibweise: Wenn alle Elemente einer Gruppe mit OR bzw. AND verknüpft werden sollen, reicht es aus, für die Logik *or* bzw. *and* zu definieren. Das gleiche gilt für die Verknüpfung der Gruppen untereinander.

Beispiel:

Gruppe 1: Ereignis A und Ereignis B
Logik Gruppe Test ist A OR B

Gruppe 2: Ereignis C und Ereignis D
Logik der Gruppe Sample ist C AND D

Die Gruppen werden mit Test AND Sample verknüpft

Wenn z.B. A vorhanden ist, B fehlt und sowohl C und D vorhanden sind, werden die Kommandos ausgeführt.

```
<events logic=Test and Sample>
  <event_group group="Test" logic="or">
    <event event_id="A"
      event_class="sample"
      event_title="Job1 ist gelaufen"
      job_name="job1" />
    <event event_id="B"
      event_class="sample"
      event_title="Job2 ist gelaufen"
      job_name="job2" />
  </event_group>
  <event_group group="Sample">
    <event event_id="C"
      event_class="sample"
      event_title="Job3 ist gelaufen"
      job_name="job3" />
    <event event_id="D"
      event_class="sample"
      event_title="Job4 ist gelaufen"
      job_name="job4" />
  </event_group>
</events>
```

Beispiel Es ist 17:00 Uhr oder später und ein bestimmter Schritt in einer Job-Kette wurde durchlaufen. In diesem Fall soll der Job *dailyPrint* gestartet werden.

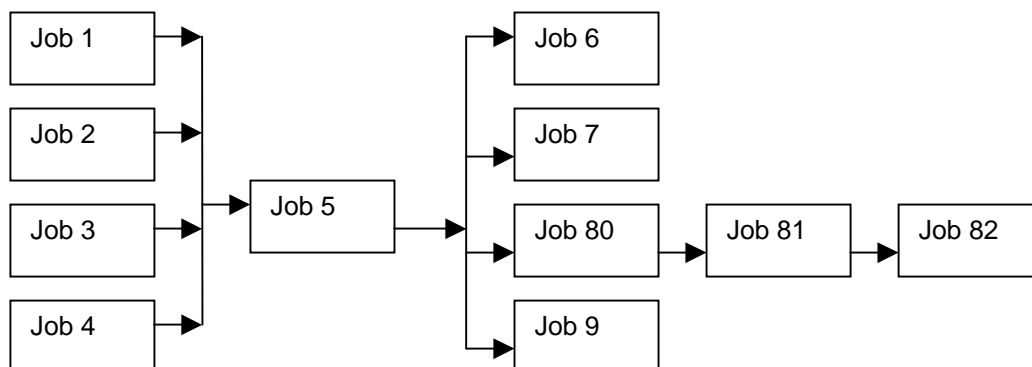
→Die Konfiguration des Event Generators für dieses Beispiel steht in Kapitel 2.1

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <action name="SampleAction">
    <events>
      <event_group group="beforePrinting" logic="and">
        <event event_id="myId1" event_class="myClass" event_title="17:00 reached" />
        <event event_id="myId2" event_class="myClass" event_title="Step100 is ready" />
      </event_group>
    </events>
    <commands>
      <command name="command_1" scheduler_host="localhost" scheduler_port="4444">
        <start_job job="dailyPrint" />
      </command>
    </commands>
  </action>
</actions>
```

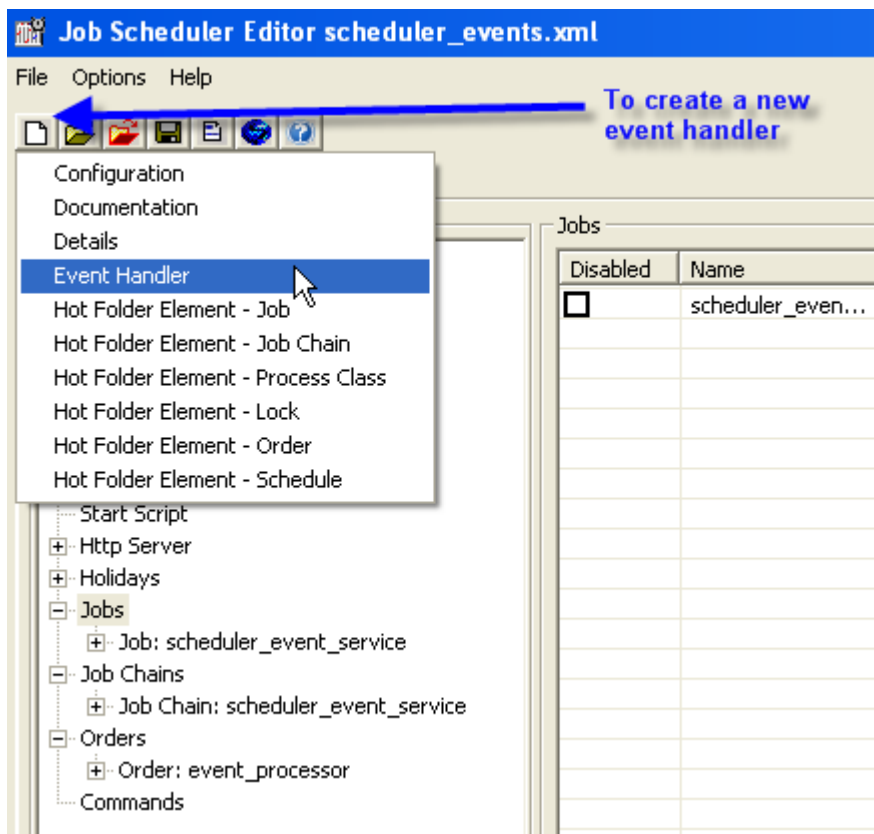
Beispiel für dieses Szenario aus Kapitel 1

In diesem Beispiel laufen die Jobs „Job 1“, „Job 2“, „Job 3“, „Job 4“ zu beliebigen Zeitpunkten ab. Erst wenn alle erfolgreich beendet sind, wird „Job 5“ gestartet. Anschließend sollen parallel „Job 6“, „Job 7“, „Job 9“ und die Job-Kette bestehend aus „Job 80“, „Job 81“, „Job 82“ ablaufen.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <action name="BeforeJob5">
    <events logic="and">
      <event_group group="pre" logic="and" event_class="pre">
        <event event_id="1" event_title="Job 1 is ready" job_name="job1" />
        <event event_id="2" event_title="Job 2 is ready" job_name="job2" />
        <event event_id="3" event_title="Job 3 is ready" job_name="job3" />
        <event event_id="4" event_title="Job 4 is ready" job_name="job4" />
      </event_group>
    </events>
    <commands>
      <command name="command_1" scheduler_host="localhost" scheduler_port="4444">
        <start_job job="job5" />
      </command>
    </commands>
  </action>
  <action name="AfterJob5">
    <events>
      <event_group group="post" event_class="post" logic="and">
        <event event_id="5" event_title="Job5 is ready" job_name="job5" />
      </event_group>
    </events>
    <commands>
      <command name="Start Job6" scheduler_host="localhost" scheduler_port="4444">
        <start_job job="job6" />
      </command>
      <command name="Start Job7" scheduler_host="localhost" scheduler_port="4444">
        <start_job job="job7" />
      </command>
      <command name="Start Job9" scheduler_host="localhost" scheduler_port="4444">
        <start_job job="job9" />
      </command>
      <command name="Start Jobchain" scheduler_host="localhost" scheduler_port="4444">
        <add_order job_chain="job_chain_sample" replace="yes" id="4711" />
      </command>
    </commands>
  </action>
</actions>
```



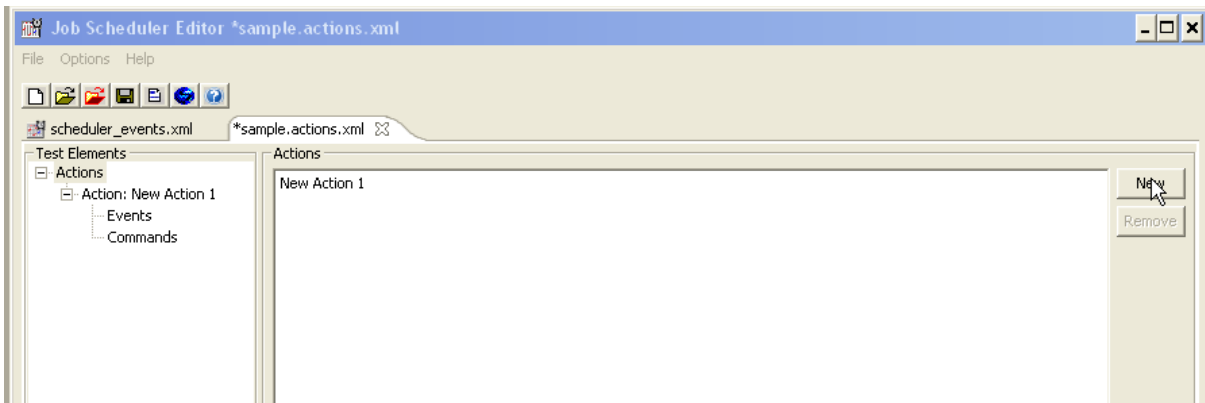
Die Kommandos und das Regelwerk lassen sich bequem mit dem Job-Editor bearbeiten:



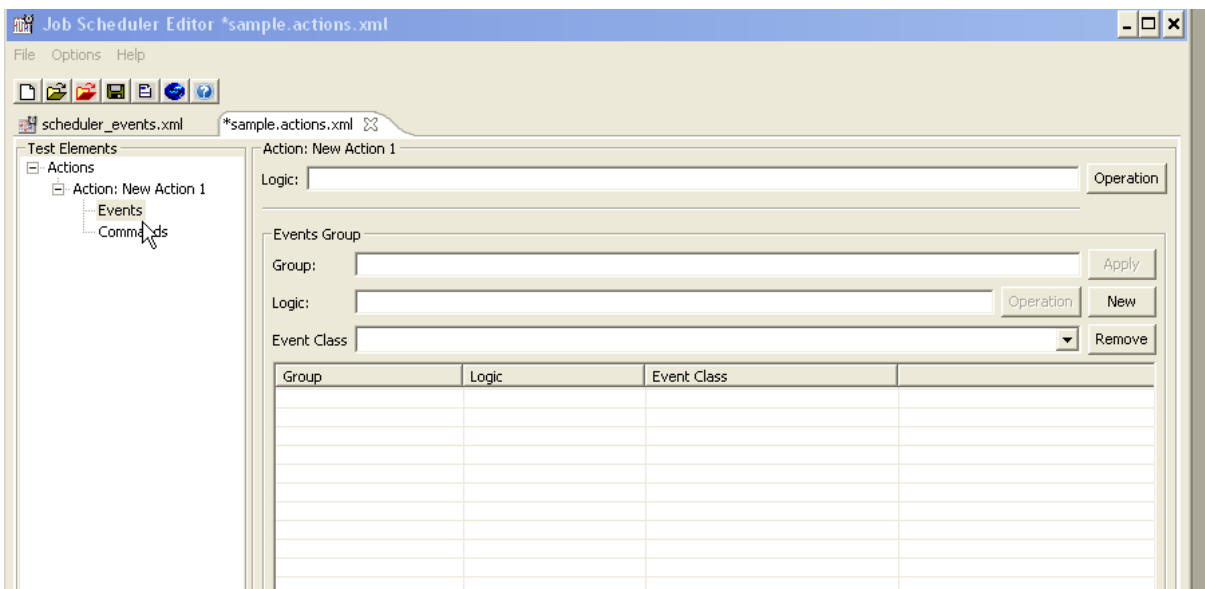
Eine Ereignisbehandlungsroutine besteht aus einer oder mehrere *Aktionen*. Eine Aktion besteht aus Ereignisgruppen und Kommandos. Die Ereignisgruppen werden entsprechend der angegebenen Bedingungen ausgewertet. Alle Gruppen untereinander werden ebenfalls mit der angegebenen Bedingung ausgewertet. Evaluieren alle Gruppen zu TRUE, dann werden die Kommandos ausgeführt. Wenn nur eine Gruppe angegeben ist, muss diese zu TRUE evaluieren. Wenn keine Bedingung angegeben ist, muss eine der Gruppen zu TRUE evaluieren.

Beispiel mit einer Gruppe und zwei Ereignissen. Wenn Ereignis 1 oder Ereignis 2 vorhanden ist, dann sollen die Kommandos ausgeführt werden.

1. Schritt: Aktion anlegen



2. Schritt: Ereignisgruppe anlegen



Eine Gruppe besitzt einen Namen und eine Logik und besteht aus einem oder mehreren Ereignissen. Es kann eine Ereignisklasse für alle Ereignisse der Gruppe vorgegeben werden. Der Standardwert für die Logik ist "OR". Für die Logik kann angegeben werden

- *or*: Eins der Ereignisse muss vorliegen
- *and*: Alle Ereignisse müssen vorliegen
- Bedingung, die die Ereignisse der Gruppe verknüpft z.B. *event1 and not event2*

In der Logik werden die Ereignisse mit ihrem Namen angesprochen. Ist der Name des Events leer, wird der Name aus `<class>.<id>` gebildet.

3. Schritt: Ereignisse angeben

*sample.actions.xml

Action: New Action 1 Group: Test

Event Title:

Event Class:

Event Id:

Job Name:

Job Chain:

Order Id:

Comment:

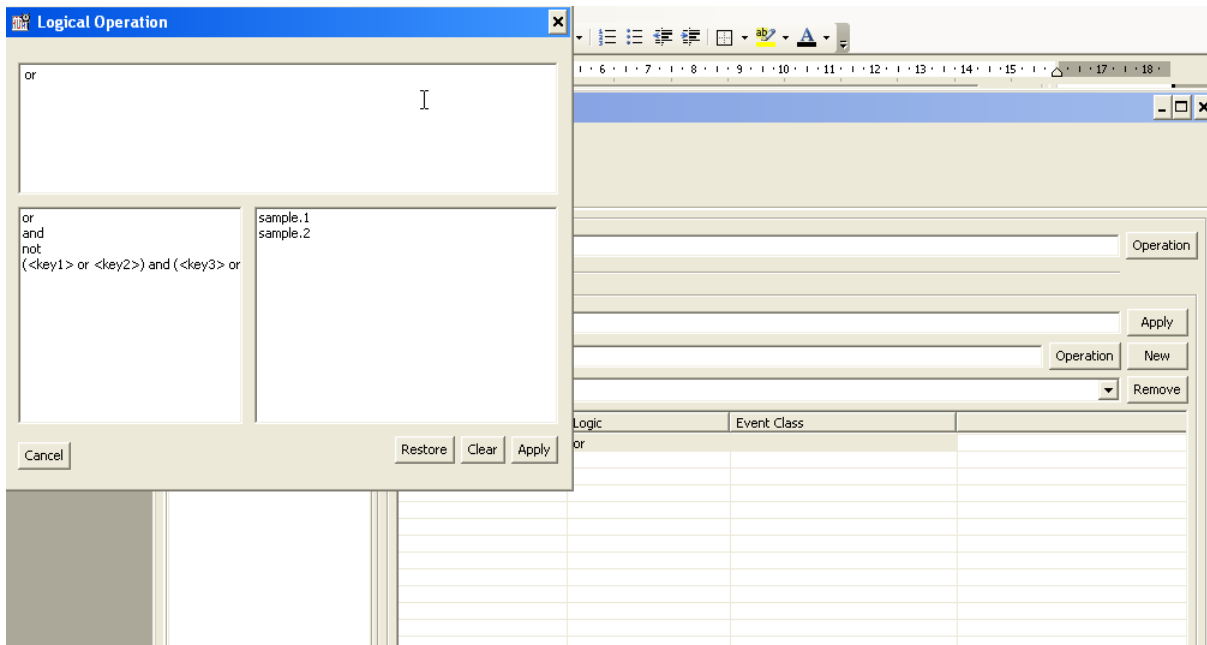
Event Id	Event Title	Event Class	Jobname	Job...	Order Id	Comment	
1	Job1 ist ...	sample	job1				<input type="button" value="Remove"/>

Ein Ereignis beinhaltet:

- Einen Titel. Wird im Event Monitor der Konsole angezeigt.
- Einen Namen. Wird verwendet, um das Ereignis in der Logik ansprechen zu können (z.B. `event1 and not event2`). Wenn in der Logik lediglich `or` oder `and` angegeben wird, muss kein Name angegeben werden. Allerdings wird empfohlen, einen Namen zu definieren, da z.B. in der Konsole im Event Monitor der Name ebenfalls ausgegeben wird.
- Eine Klasse. Wird verwendet, um zu prüfen, ob das Ereignis in der Liste der aktuell vorhandenen Ereignisse enthalten ist. Wenn der Name des Ereignisses leer ist, wird ein interner Name aus `klasse.id` gebildet.
- Eine ID. Wird verwendet, um zu prüfen, ob das Ereignis in der Liste der aktuell vorhandenen Ereignisse enthalten ist. Wenn der Name des Ereignisses leer ist, wird ein interner Name aus `klasse.id` gebildet.
- Einen Job-Namen. Wird verwendet, um zu prüfen, ob das Ereignis in der Liste der aktuell vorhandenen Ereignisse enthalten ist.
- Einen Job-Kettennamen. Wird verwendet, um zu prüfen, ob das Ereignis in der Liste der aktuell vorhandenen Ereignisse enthalten ist.
- Eine Auftragskennung (`order id`). Wird verwendet, um zu prüfen, ob das Ereignis in der Liste der aktuell vorhandenen Ereignisse enthalten ist.

4. Schritt: Logik angeben

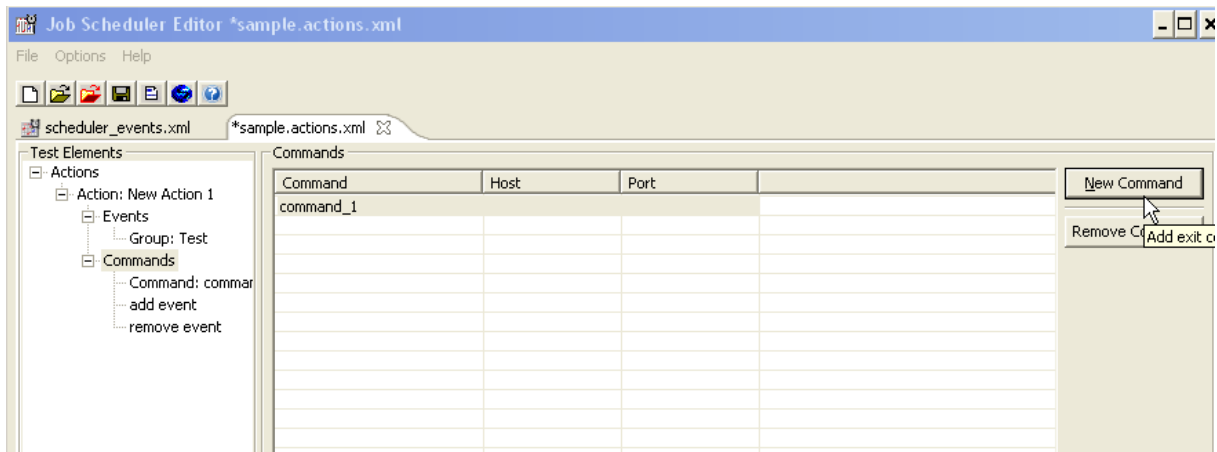
Für die Angabe der Logik kann ein Wizzard verwendet werden oder die Logik wird direkt in das vorgesehene Eingabefeld eingegeben. In diesem Feld lautet die Logik "or"



5. Schritt: Eventuell weitere Gruppen erfassen

Mehrere Gruppen werden über das Gruppenlogikfeld verknüpft. Der Standardwert für die Gruppenlogik lautet "or".

6. Schritt: Kommandos erfassen



Für alle Kommandos innerhalb eines Blocks wird der Host und ein Port angegeben. An diesen Job Scheduler wird das Kommando gesendet.

Ein Kommando wird mit "New Command" hinzugefügt. Es gibt die Kommandos

- start job: Startet einen Job. Es wird die Startzeit angegeben.
- order: Startet eine Job-Kette
 - Job-Kette: Die Job-Kette für den Auftrag
 - Order ID: Ein Auftrag hat eine eindeutige ID
 - Startzeit: Aufträge können verzögert gestartet werden
 - Priorität: Die Priorität des Auftrages
 - Titel: Ein beliebiger Titel
 - Status: Der Status in der Job-Kette, mit dem der Auftrag startet.
 - End-Status: Der Status in der Job-Kette, bis zu dem der Auftrag laufen soll
 - Replace: Soll der Auftrag einen vorhandenen überschreiben?

- add event: Erzeugt ein neues Ereignis
- remove event: Löscht Ereignisse. Es können z.B. alle Ereignisse einer Klasse oder auch eines Jobs gelöscht werden.

The screenshot shows a web application window titled 'sample.actions.xml'. It contains a form for adding a new event. The form has the following fields and buttons:

- Action: Group:** (Label)
- Event Class:** (Dropdown menu)
- Event Id:** (Text input)
- Job Name:** (Text input)
- Job Chain:** (Text input)
- Order Id:** (Text input)
- Buttons:** 'Apply' and 'New' (located to the right of the Event Class field)

Below the form is a table with the following columns: Event Id, Event Title, Event Class, Jobname, Job..., Order Id, Comment. A 'Remove' button is located to the right of the table.

Der Event Handler wird als XML-Datei gespeichert.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <action name="New Action 1">
    <events>
      <event_group group="Test" logic="or">
        <event event_id="1"
          event_class="sample"
          event_title="Job1 ist gelaufen"
          job_name="job1" />
        <event event_id="2"
          event_class="sample"
          event_title="Job2 ist gelaufen"
          job_name="jobd2" />
      </event_group>
    </events>
    <commands>
      <command name="command_1" scheduler_host="localhost" scheduler_port="4454">
        <start_job job="job0" at="now" />
        <add_order job_chain="job_chain1" replace="yes" />
      </command>
    </commands>
  </action>
</actions>
```

Auswerten des Regelwerkes

Bei der Auswertung des Regelwerkes wird geprüft, ob die in der Logik einer Event Group angegebenen Ereignisse in der Liste der aktuell vorhandenen Ereignisse gefunden werden können. Die so ermittelten TRUE/FALSE Werte werden mit den booleschen Operatoren verknüpft.

Ein Event ist in der Liste der Events vorhanden, wenn für die Attribute, die im Event Handler angegeben sind, Übereinstimmung gefunden wird. Nicht beachtet werden die Attribute `title`, `expires`, `creation`.

Beispiel

Diese `event_group` evaluiert zu `true`, wenn `event1` vorliegt und `event2` fehlt.

- `event1` liegt vor, wenn es ein Event mit *der* `event_class="sample"`, der `event_id="1"` und dem `job_name="job1"` gibt
- `event2` liegt vor, wenn es ein Event mit der `event_id="2"` gibt.

```
<event_group group="Test" logic="event1 and not event2">
  <event event_id="1"
    event_name="event1"
    event_class="sample"
    event_title="Job1 ist gelaufen"
    job_name="job1" />
  <event event_id="2"
    event_name="event2"
    event_title="Job1 ist gelaufen" />
</event_group>
```

5 Events überwachen

Mit der Job Scheduler Konsole können die Ereignisse überwacht werden. Die Job Scheduler Konsole steht mit der kommerziellen Lizenz zur Verfügung.

Die Konsole wird mit einer Konfigurationsdatei eingestellt. Dort werden die zu beobachtenden Job Scheduler eingestellt. Pro Job Scheduler wird ein host/port Paar angegeben, die jeweils fortlaufend nummeriert sind.

Für das Plugin der Event Handler ist eine Einstellung für das Verzeichnis mit den Event Handler notwendig. Im Abschnitt `[plugin_action_show_dialog]` wird dafür das Verzeichnis angegeben

Die Konsole wird mit `scheduler/bin/console` gestartet.

Beispiel für eine Konfiguration

```
[Console.Supervisor]
host1=kyrill.sos
port1=9000

host2=kyrill.sos
port2=9001


host3=kyrill.sos
port3=9002

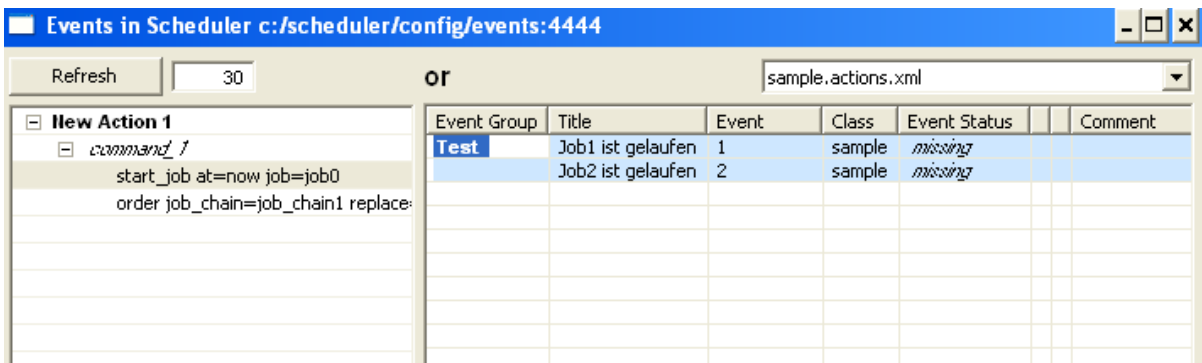
host4=localhost
port4=4474

host5=localhost
port5=4454

[plugin_action_show_dialog]
configuration_directory=c:/scheduler/config/events
```


- Liste der Event Handler mit aktuellen Auswertungsstatus

Die Liste der Event Handler wird aufgerufen, wenn auf  geklickt wird.



The screenshot shows a window titled "Events in Scheduler c:/scheduler/config/events:4444". It features a "Refresh" button, a numeric input field with "30", and a dropdown menu showing "sample.actions.xml". On the left, a tree view under "New Action 1" shows a sub-item "command_1" with associated commands: "start_job at=now job=job0" and "order job_chain=job_chain1 replace:". The main area is a table with the following data:

Event Group	Title	Event	Class	Event Status	Comment
Test	Job1 ist gelaufen	1	sample	missing	
	Job2 ist gelaufen	2	sample	missing	

Aus der Liste wird die gewünschte Ereignisbehandlungsroutine ausgewählt. Es werden dann die Aktionen angezeigt. Im linken Fenster stehen die Kommandos an den Job Scheduler, im rechten Fenster werden die erforderlichen Ereignisse unterteilt in Ereignisgruppen angezeigt. Die Logik, mit der die Ereignisse einer Ereignisgruppe verknüpft sind, wird angezeigt, wenn auf den Namen der Ereignisgruppe geklickt wird.

Die Ereignisse im der rechten Fenster werden farblich markiert.

Beispiel

Event Group	Title	Event	Class	Event Status	Commer
3		0	example	<i>missing</i>	
		0	test	active	2 2
4		event3	example	<i>missing</i>	
		event4	example	<i>missing</i>	
5	simple_shell_job	0	example_job	active	2 2
	sample_job	0	example_job	<i>missing</i>	

Die gelbe Hintergrundfarbe bedeutet:

"Alle Ereignisse der Gruppe müssen vorhanden sein (Logik=AND)"

Die hellblaue Hintergrundfarbe bedeutet:

"Eines des Ereignisse der Gruppe muss vorhanden sein" (Logik=OR)

Die grüne Hintergrundfarbe bedeutet: "Das Ereignis ist vorhanden"

Die Aktionen im rechten Fenster werden mit rot markiert, wenn das Regelwerk zu TRUE evaluiert.

6 Ereignisverarbeitung einrichten

1 Event Processor installieren

Der Event Processor wird in einem Supervisor Job Scheduler installiert.

Die Datei `scheduler_event_service` befindet sich In der Auslieferung des Job Schedulers im Verzeichnis `config` und definiert die Job-Kette `scheduler_event_service`. Diese Job-Kette implementiert den Event Processor.

Die Konfiguration wird dem Supervisor Job Scheduler bekannt gegeben, indem sie als `<base file="..."/>` eingebunden wird.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<config>
  <!-- -->
  <base file="scheduler_events.xml"/>
  <process_classes>
    <process_class max_processes="10"/>
  </process_classes>
</config>
```

Die Datei kann unter

http://www.sos-berlin.com/download/scheduler/samples/events_supervisor.zip

heruntergeladen werden.

Anpassungen:

Der Parameter `"expiration_period"` sollte auf den Wert gestellt werden, an dem standardmäßig alle Ereignisse verfallen (z.B. `<param name="expiration_period" value="00:00"/>`)

2 Mindestens einen Event Handler definieren

Der Event Handler kann mit dem Job Editor erstellt werden. Alle Event Handler werden im Verzeichnis `scheduler/config/events` abgelegt. Das Verzeichnis für die Event Handler kann im Event Processor mit dem Parameter `<param name="event_handler_filespec" value="scheduler_events.xml"/>` eingestellt werden

Ein Beispiel für einen sehr einfachen Event Handler, der nach Job A den Job B startet, sieht so aus.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <action name="sample">
    <events>
      <event_group group="Main">
        <event event_id="0" event_title="Job A" job_name="job_a" />
      </event_group>
    </events>
    <commands>
      <command name="start_Job_B" scheduler_host="localhost" scheduler_port="4444">
        <start_job job="job_b" />
      </command>
    </commands>
  </action>
</actions>
```

3 Event Generator für alle im Event Handler aufgeführten Ereignisse einrichten

Um das Ereignis "Job A ist gestartet" zu erzeugen, gibt es mehrere Möglichkeiten:

- Ist es ein Shell Job, dann kann das Shell Script `jobscheduler_event.cmd` aufgerufen werden
- Ist es ein Java Job, dann kann der Monitor eingebunden werden.

Beispiel für die Verwendung als Shell Script:

```
<job>
  <script language="shell">
    <![CDATA[
      # any commands
      # submit event to Supervisor Job Scheduler
      ${SCHEDULER_HOME}/bin/jobscheduler_event.sh -x $? -e "sample"
    ]]>
  </script>
  <run_time/>
</job>
```

7 Beispiele für Ereignisbehandlungsroutinen

7.1 Split and Merge

Das Beispiel besteht aus zwei Aktionen und zwar der Aktion "Split" und der Aktion "Merge".

Die Aktion "Split" startet die Jobs *job_B_1* und *job_B_2* wenn das Ereignis *job_A* eintrifft. Außerdem wird das Ereignis nach Verarbeitung der Aktionen entfernt.

Die Aktion „Merge“ wartet auf die Ereignisse *job_B_1* beendet und *job_B_2* beendet sowie auf die Beendigung des manuell zu startenden Jobs *job_B_4*. Die Aktion startet dann den Job *job_C* und entfernt die beteiligten Ereignisse.

```
<actions>
  <action name="Split">
    <events>
      <event_group group="A">
        <event event_class="example_split" event_id="0"
          event_title="Job A" job_name="job_A"/>
      </event_group>
    </events>

    <commands>
      <command name="Start B1" scheduler_host="localhost" scheduler_port="4474">
        <start_job job="samples/splitAndMerge/job_B_1"/>
      </command>

      <command name="Start B2" scheduler_host="localhost" scheduler_port="4474">
        <start_job job="samples/splitAndMerge/job_B_2"/>
      </command>
      <remove_event debug="example_split">
        <event event_class="example_split" event_id="0" job_name="job_A"/>
      </remove_event>
    </commands>
  </action>

  <action name="And Merge">
    <events>
      <event_group logic="and" group="A">
        <event event_class="example_andmerge" event_id="0"
          event_title="Job B_1" job_name="job_B_1"/>
        <event event_class="example_andmerge" event_id="0"
          event_title="Job B_2" job_name="job_B_2"
          comment="Wenn nicht bis 18::00 Uhr aktiv, 01728388383 anrufen"/>
        <event event_class="example_andmerge" event_id="0" event_title="Job B_4"
          job_name="job_B_4" comment="Wird manuell gestartet"/>
      </event_group>
    </events>

    <commands>
      <command name="Start C" scheduler_host="localhost" scheduler_port="4474">
        <start_job job="samples/splitAndMerge/job_C"/>
      </command>
      <remove_event debug="example_andmerge">
        <event event_class="example_andmerge" event_id="0"/>
      </remove_event>
    </commands>
  </action>
</actions>
```

7.2 Job A und Job B auf Job Scheduler 1, danach Job C auf Job Scheduler 2

Job A und Job B laufen in Job Scheduler 1 ab und erzeugen zwei Ereignisse. Liegen beide Ereignisse vor, dann wird Job_C in Job Scheduler 2 gestartet.

```
<actions>
  <action name="New Action 1">
    <events>
      <event_group group="Main">
        <event event_id="1" event_class="sample" event_title="Job A" job_name="job_A" />
        <event event_id="2" event_class="sample" event_title="Job B" job_name="job_B" />
      </event_group>
    </events>
    <commands>
      <command name="command_1" scheduler_host="Scheduler2" scheduler_port="4444">
        <start_job job="job_C" />
      </command>
    </commands>
  </action>
</actions>
```

7.3 Job A ist bis 5:00 Uhr nicht gelaufen

In diesem Beispiel wird davon ausgegangen, dass *job_A* ein Verzeichnis überwacht. Der *job_1700* dient ausschließlich dazu, um 17:00 Uhr zu laufen und ein Ereignis zu senden.

Der *job_B* wird gestartet, wenn *job_1700* abgeschlossen ist und *job_A* nicht ausgeführt wurde:

```
<actions>
  <action name="Missing File">
    <events logic="A">
      <event_group logic="example_missing.time and Not example_missing.file" group="A">
        <event event_class="example_missing" event_id="file"
          event_title="File is detected" job_name="job_A" />
        <event event_class="example_missing" event_id="time"
          event_title="Its Teatime now" job_name="job_1700" />
      </event_group>
    </events>
    <commands>
      <command name="Start B1" scheduler_host="localhost" scheduler_port="4474">
        <start_job job="job_B1" />
      </command>
      <remove_event>
        <event event_class="example_missing" event_id="0" />
      </remove_event>
    </commands>
  </action>
</actions>
```

7.4 Benachrichtigung wenn bis 17:00 Uhr Datei nicht eingetroffen ist

Ereignis "Es ist 17:00" erzeugen

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job name="sample" title="Submit Event &quot;17:00 Uhr&quot;">
  <description>
    <include file="jobs/JobSchedulerSubmitEventJob.xml"/>
  </description>
  <params>
    <param name="scheduler_event_class" value="file_arrived"/>
    <param name="scheduler_event_id" value="1"/>
  </params>
  <script language="java"
    java_class="sos.scheduler.job.JobSchedulerSubmitEventJob"/>
  <run_time>
    <period single_start="17:00"/>
  </run_time>
</job>
```

Ereignis "File has arrived" erzeugen

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<job name="sample" title="Submit Event &quot;File has arrived&quot;">
  <description>
    <include file="jobs/JobSchedulerSubmitEventJob.xml"/>
  </description>
  <params>
    <param name="scheduler_event_class" value="file_arrived"/>
    <param name="scheduler_event_id" value="2"/>
  </params>
  <script language="java"
    java_class="sos.scheduler.job.JobSchedulerSubmitEventJob"/>
</job>
```

Event Handler

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <action name="FileIsMissing">
    <events>
      <event_group group="File Missing" logic="Its17_00 and not File"
        event_class="file_arrived">
        <event event_name="File" event_class="file_arrived"
          event_title="File has arrived" event_id="2" />
        <event event_name="Its17_00"
          event_title="A specific time is reached" event_id="1" />
      </event_group>
    </events>
    <commands>
      <command name="notify" scheduler_host="localhost" scheduler_port="4444">
        <start_job job="job_notify">
          <params>
            <param name="to" value="admin@host.de" />
            <param name="subject" value="File is missing" />
          </params>
        </start_job>
      </command>
      <remove_event>
        <event event_class="file_arrived" />
      </remove_event>
    </commands>
  </action>
</actions>
```

8 Anhang

Konfiguration eines Event Processors

Die Job-Kette beinhaltet den Job `scheduler_event`. Die Parametrisierung dieses Jobs ist in der Job Dokumentation `scheduler/jobs/JobSchedulerEventJob.xml` beschrieben.

```
<config>
  <jobs>

    <job name="scheduler_event_service"
        title="Process Events"
        order="yes"
        stop_on_error="no"
        timeout="120">
      <description>
        <include file="jobs/JobSchedulerEventJob.xml"/>
      </description>

      <params>
        <param name="event_handler_filepath"
            value="./config/events"/>
        <param name="event_handler_filespec"
            value="scheduler_events.xsl"/>
        <param name="expiration_period"
            value="12:00"/>
      </params>

      <script java_class="sos.scheduler.job.JobSchedulerEventJob"
            language="java"/>

      <run_time/>
    </job>
  </jobs>
  <job_chains>
    <job_chain name="scheduler_event_service"
        orders_recoverable="no"
        visible="yes">
      <job_chain_node state="start"
          job="scheduler_event_service"
          next_state="end"
          error_state="error"/>

      <job_chain_node state="end"/>

      <job_chain_node state="error"/>
    </job_chain>
  </job_chains>
  <commands>
    <add_order id="event_processor"
        job_chain="scheduler_event_service">
      <params>
        <param name="action"
            value="process"/>
        <param name="event_handler_filespec"
            value=".xsl$"/>
      </params>

      <run_time let_run="yes"
          repeat="300"/>
    </add_order>
  </commands>
</config>
```