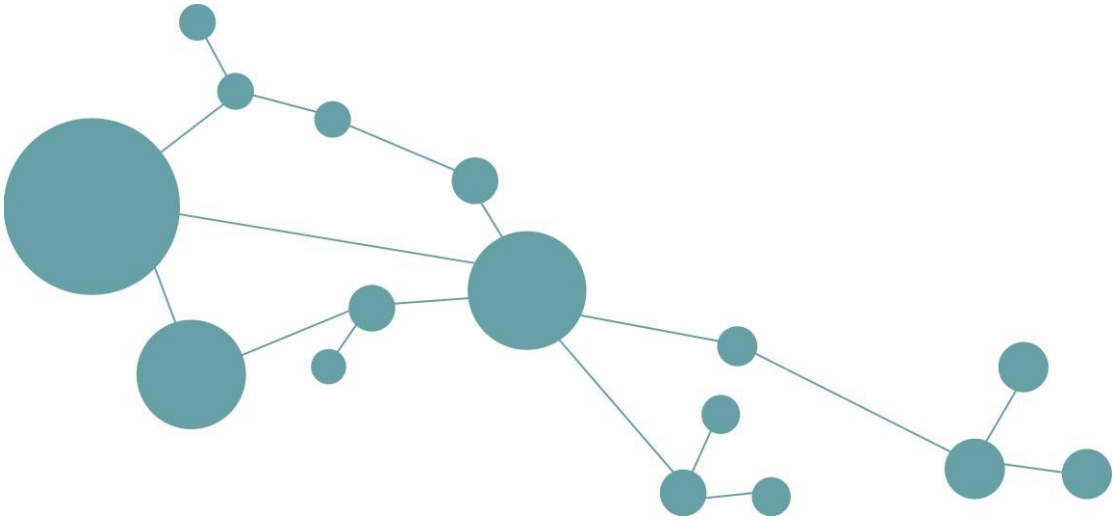

Job-Client 3.2

i·views

Inhaltsverzeichnis

1 Allgemeines zum Job-Client **3**

2 Starten des Joblicents **3**

2.1 Kommandozeile 3

2.2 Installation als Windows-Dienst 4

3 Konfiguration des Job-Clients **5**

3.1 Konfigurationsdatei "jobclient.ini" 5

3.1.1 Indexjobs 8

3.1.2 KBrainbotJob 9

3.1.3 KExternalCommandJob 9

3.1.4 KExtractBlobTextJob 13

3.1.5 KQueryJob 13

3.1.6 KScriptJob 13

3.2 Beispiel für eine Ini-Datei 13

3.3 Erweiterungen ab K-Infinity-Core 3.2 13

1 Allgemeines zum Job-Client

Der Job-Client erbringt zum einen Dienste für andere K-Infinity-Clients, um diese von rechenzeit- oder datenintensiven Aufgaben zu entlasten. Zum anderen dient er als Brücke zwischen K-Infinity-Clients und externen Systemen.

Zu seinen wichtigsten Aufgaben gehört die Ausführung aller Arten von Suchen sowie die Auslieferung der Suchergebnisse an die Clients (Sortierung, textuelle Aufbereitung, Rechtefilterung).

Im Normalfall wartet der Client auf die Fertigstellung eines Auftrags (Synchronbetrieb).

Für die Ausführung komplexer Suchen, das Erstellen von Statistiken, Batch-Abgleiche, Datenaufbereitungen, Datenbereinigungen, etc. muss der Client nicht auf die Fertigstellung warten (Asynchronbetrieb). Das Ergebnis wird vom Service bereitgestellt und der Client wird benachrichtigt. Das Ergebnis kann dann beliebige Zeit später eingesehen werden. Da das Ergebnis auch persistent gemacht wird, ist es auch nach einem Neustart des Systems bzw. im Falle eines Fail-Overs weiterhin verfügbar.

Funktionsweise:
In dem vom K-Infinity-Mediator bereitgestellten geteilten Objektraum werden die Aufträge der Clients an die Services in sogenannten Pools abgelegt. Alle K-Infinity Job-Clients werden über neue Aufträge notifiziert und bewerben sich - sofern sie aktuell frei sind - für die Bearbeitung des neuen Auftrags. Nach Bearbeitung des Auftrags wird das Resultat wieder im geteilten Objektraum bereitgestellt, der beauftragende Client wird benachrichtigt und das Ergebnis kann abgerufen und zur Anzeige gebracht werden. Somit beauftragt der Client zwar logisch einen Job-Client, physikalisch läuft die Kommunikation aber immer über den K-Infinity Server. Für den Client ist es transparent, welcher Job-Client seinen Auftrag ausführt, sowie es für den Job-Client transparent ist, wo der Auftrag herkommt und wie viele parallele Job-Clients zurzeit aktiv sind. Für Administratoren ist die Installation und Wartung der Job-Clients daher sehr einfach und flexibel. Job-Clients lassen sich beliebig skalieren, auf verschiedene Rechner verteilen und dynamisch zu- und abschalten. Eine externe Clusterung oder sonstige Orchestrierung ist nicht erforderlich.

Technische Daten:
Multi-Platform Executable auf Basis der VisualWorks Smalltalk Virtual Machine (jobclient.exe bzw. jobclient.im)
Benötigt eine TCP/IP-Verbindung zum K-Infinity Server
Automatische Lastverteilung zwischen den Services
Job-Clients können zu jeder Zeit zugeschaltet oder heruntergefahren werden
Standby-Modus bei zeitweiliger Nicht-Verfügbarkeit benötigter Ressourcen

2 Starten des Joblicents

Der Job-Client kann entweder direkt in der Kommandozeile gestartet werden oder als Windows-Dienst installiert werden.

2.1 Kommandozeile

Wird der Job-Client ohne jegliche Parameter gestartet, so werden die erforderlichen Parameter aus der Ini-Datei jobclient.ini gelesen und die Fehlermeldungen in die Datei jobclient.log geschrieben.

Falls es zu einem Aufrufparameter auch einen Eintrag der Ini-Datei gibt, hat der der Aufrufparameter höhere Priorität.

`-inifile <Dateiname>, -ini <Dateiname>`

Name der Ini-Datei, die statt der Standard-Ini-Datei verwendet wird.

`-clientTimeout <sec>`

Setzt die Zeit, innerhalb der sich ein Client automatisch melden muss, auf <sec> Sekunden. Der Wert sollte mindestens auf 600 gesetzt werden.

`-host <Servername:Port>`

Name und ggf. IP-Adresse des K-Infinity Servers.

`-volume <Volumename>`

Name des Wissensnetzes, auf dem gearbeitet werden soll.

Kommandozeilen-Parameter für das Logging:

`-nolog`

Schaltet Logging ab

`-loglevel <Integer>`

Konfiguriert, welche Meldungen im Log erscheinen sollen:

- 0: Alle Meldungen inklusive Debug-Ausgaben
- 10 (Standardwert): Alle Meldungen außer Debug-Ausgaben
- 20: Nur Warnungen und Fehlermeldungen
- 30: Nur Fehlermeldungen

`-logfile <Dateiname>, -log <Dateiname>`

Name der Log-Datei, die statt der Standard-Log-Datei verwendet wird. Dieser Parameter muss auf jeden Fall verändert werden, wenn mehrere Clients im selben Arbeitsverzeichnis gestartet werden sind.

`-debug`

Schaltet das Logging auf debug-mode

`-log <logname>`

Setzt die Logdatei auf <logname>.

2.2 Installation als Windows-Dienst

Der Job-Client kann als Windows-Dienst installiert werden:

`jobclient -installAsService <Name> [<Dienst_1> ... <Dienst_n>]`

„Name“ gibt den Namen des Windows-Dienstes an. Das aktuelle Verzeichnis, aus dem die Installation als Dienst gestartet wird, wird als Arbeitsverzeichnis verwendet

Die (optional) angegebenen Dienste werden als benötigte Dienste eingetragen. Insbesondere der K-Infinity Server-Dienst bietet sich hier an.

Der Starttyp des Dienstes kann danach noch bei Bedarf in der Windows-Diensteverwaltung auf „Automatisch“ geändert werde.

Der Dienst kann mit

`jobclient -deinstallService <Name>`

wieder entfernt werden. Vorsicht: Es wird nicht überprüft ab der angegebene Dienst tatsächlich ein Job-Client oder ein anderer Dienst ist. Alternativ kann der Dienst auch mit dem Service Control-Tool von Microsoft deinstalliert werden:

`sc delete <Name>`

3 Konfiguration des Job-Clients

3.1 Konfigurationsdatei "jobclient.ini"

Die Konfiguration des Job-Clients wird in der Ini-Datei vorgenommen. Falls nicht durch den Aufrufparameter "-inifile" spezifiziert wird "jobclient.ini" als Konfigurationsdatei verwendet.

`host=<Hostname:Portnummer>`

Name / IP-Adresse des Servers

`volume=<Volumename>`

Der Name des Wissensnetzes, auf dem gearbeitet werden soll.

`jobPools=Jobname1 [,Jobname2, ...]`

Angabe, welche Jobs der Job-Client abarbeiten soll. Die Namen der zu startenden Job-Pools sind hier kommagetrennt anzugeben.

Beispiel:

`jobPools=KExpertQueryJob,KSearchJob,index`

Die möglichen Typen werden in den Unterkapiteln dargestellt.

`cacheDir=<Verzeichns>`

Beschreibung des Ortes, an dem der Cache für den Job-Client angelegt wird.

`volumeAccessor=CatBSBlockFileVolumeAccessor oder CatCSVolumeFileStorageAccessor`

Beschreibung der Speicherart des Caches. Wenn nicht angegeben wird CatBSBlockFileVolumeAccessor verwendet. Diese Speicherart ist vor allem bei großen Netzen zu empfehlen, da CatCSVolumeFileStorageAccessor eine große Anzahl an Dateien anlegen würde.

`maxCacheSize=<Größe in MB>`

Zielgröße des Caches

`shutDownTimeout=<Sekunden>`

Dauer, auf die beim Herunterfahren des Job-Clients auf die Beendigung der aktiven Jobs gewartet wird. Nach Ablauf werden die Jobs abgebrochen. Der Standardwert ist 10 Sekunden.

`enableLowSpaceHandler=true/false`

Mit dieser Option wird der LowSpaceHandler eingeschaltet. Dieser sollte auf jedenfall bei großen Netzen eingeschaltet werden.

`useProxyValueHolder=true/false`

Mit dieser Option kann gesteuert werden, ob der Job-Client Indexzugriffe per RPC durchführt (true), oder Indizes in den Speicher lädt (false). Diese Option sollte ausgeschaltet werden, wenn der Mediator entlastet werden soll. Dabei sollte allerdings darauf geachtet werden, dass der Job-Client genug Speicher zur Verfügung hat. Falls der Job-Client für schreibende Jobs konfiguriert wurde, hat diese Option keinen Effekt, da dann der Indexzugriff immer per RPC durchgeführt wird. Es wird beim Start im Log eine Meldung ausgegeben, falls man den Wert auf false gesetzt hat.

`name=<Job-Client-Name>`

Dieser Name wird verwendet, um den Job-Client im AdminTool in der Übersichtsliste aller Job-Clients zu identifizieren.

Logging-Einstellungen:

`loglevel = <Integer>`

Konfiguriert, welche Meldungen im Log erscheinen sollen:

- 0: Alle Meldungen inclusive Debug-Ausgaben
- 10 (Standardwert): Alle Meldungen ausser Debug-Ausgaben
- 20: Nur Warnungen und Fehlermeldungen
- 30: Nur Fehlermeldungen

`debug = true/false`

Setzt den Log-Level bei true auf 0, bei false auf 10

`channels = <Channel1> [,<Channel2>,...]`

Namen von Channelfiltern. Mit Hilfe der Channelfilter werden nur Log-Meldungen ausgegeben, die zu den angegebenen Channelfiltern gehören. Der Name eines Channelfilters deutet darauf hin, zu welchem Themengebiet die Log-Ausgaben gehören. Welche Channelfilter möglich sind, erfährt man in der Kommandozeile mit Hilfe des Parameters -availableChannels.

`logfile = <Dateiname>`
`log = <Dateiname>`

Name der Log-Datei, die statt der Standard-Log-Datei verwendet wird.

`maxLogSize = <size>`

die maximale Grösse des Logfiles, ab der die alte Logdatei archiviert wird und eine neue angebrochen wird. Bei Werten kleiner als 1024 wird die Angabe als in MB verstanden

`notifiers = <Name1> [,<Name2>,...]`

Namen von Notifiern. Ein Notifier wird bei Log-Ausgaben ab einem bestimmten Level aktiviert (typischerweise bei Fehlermeldungen). Siehe Abschnitt „Mail-Notifier“.

`logprefix = <Prefix1> [, <Prefix2>,... ]`

Zusätzliche Daten, die bei jeder Log-Ausgabe hinzugefügt werden

- $timestamp$: Zeitstempel der Log-Ausgabe
- pid : Prozess-ID der Anwendung
- $proc$: ID des aktuellen Smalltalk-Threads
- $alloc$: belegter Speicher der VM (in Megabyte)
- $free$: Freier Speicher der VM (in Megabyte)
- $incGC$: Status inkrementelle GCs
- os : Information über OS
- cmd : Kommandozeile
- $build$: Build-Version
- $coast$: COAST-Version

Bei einem Prefix, der nicht in dieser Liste enthalten ist, wird der Prefix unverändert ausgegeben

Speicher-Einstellungen:

Die folgenden drei Parameter dienen zur Konfiguration der Speicherzuteilung und -nutzung. Erlaubt ist die Angabe von Werten entweder in Megabyte oder in tatsächlichen Byte, wobei die Annahme gilt, dass sich Werte kleiner als 1048576 auf Megabyte-Angaben beziehen.

`maxMemory=<Integer, in MB>`

Maximal erlaubte Hauptspeicherbelegung. Mininmal 50 MB, standardmäßig gesamter physikalisch vorhandener Hauptspeicher (unter Windows) bzw. 512 MB.

`growthRegimeUpperBound=<Integer, in MB>`

Hauptspeicherbelegung ab der verstärkt versucht wird, Speicher freizugeben. Standardmäßig 0.6 * maxMemory.

`freeMemoryBound=<Integer, in MB> [10]`

Die folgende Einträge sind veraltet und sollten aus der Konfigurationsdatei entfernt werden:

- unloadStrategy
- minAge
- unloadInterval
- unloadSize
- keepSize

Mail-Notifier Einstellungen:

Log-Ausgaben können über Mail-Notifier zusätzlich zum Eintrag in die Log-Datei per Mail versendet werden. Sinnvollerweise wählt man einen nicht zu geringen Schwellwert (Log-Level = 30), ab dem eine Mail versendet wird.

Für jeden angegebenen Namen aus der Notifier-Liste muss eine Konfiguration angegeben werden, die im Abschnitt [Name] angegeben wird.

Folgende Angaben sind für alle Notifier gleich:

- class = <Klassenname> : Art des Notifiers (z. Zt. nur COAST.CoastMailNotifier)

- level = <Integer> : Log-Level, ab dem der Notifier aktiv wird

Spezifische Angaben für COAST.CoastMailNotifier;

- attachLog = <Boolean> : Wenn true wird ein Error-Log als Attachment an die Mail angehängt
- sender = <Mailadresse> : Absender der Mail
- recipient = <Mailadresse> : Empfänger der Mail
- smtpHost = <Hostname> : Mailserver
- smtpPort = <Integer> : Port des Mailservers. Standardwert = 25 / 465 (TLS/SSL)
- retries = <Integer> : Anzahl der Versuche, die Mail bei einem Fehlschlag nochmals zu versenden
- retryDelay = <Integer> : Wartezeit zwischen zwei fehlgeschlagenen Versuchen, eine Mail zu versenden
- username = <Nutzername> : optionaler Anmeldename, benötigt für Authentifizierung
- password = <Password> : optionales Passwort, benötigt für Authentifizierung
- tls = true/false : gesicherte Verbindung (TLS/SSL) verwenden. Standardwert = false. Falls true müssen username und password gesetzt sein.

TLS/SSL ist für Authentifizierung nicht zwingend erforderlich, aber dringend zu empfehlen. Die Kompatibilitätserweiterung STARTTLS wird nicht unterstützt.

Beispiel:

```
notifiers = errorMailNotifier

[errorMailNotifier]
class = COAST.CoastMailNotifier
level = 30
;Spezifische Angaben für Mail-Versand
attachLog = true
sender = sender@hostname.de
recipient = receiver@hostname.de
smtpHost = mailserver.hostname.de
retries = 3
retryDelay = 30
```

Die angegebene Konfiguration definiert einen Mail-Notifier, die alle Fehlermeldungen per Mail verschickt.

3.1.1 Indexjobs

- Kategorie(n): **index**

Werden für den Jobpool die unten angezeigten Jobklassen oder **index** angegeben, dann werden die Indexierungsaufträge vom Job-Client ausgeführt. Die Indexierungsaufträge sollten nur von einem einzigen Job-Client durchgeführt werden. Statt alle Jobklassen einzeln im Job-Pool aufzuzählen, kann auch der symbolische Name **index** verwendet werden.

KAddAllToIndexJob

- Bezeichnung: Attribute zum Index hinzufügen

KLightweightIndexJob

- Bezeichnung: Externen Index aktualisieren

  Ein externer Index wird über den KLightweightIndexJob gepflegt.

KLuceneAdminJob

- Bezeichnung: Lucene Verwaltungsaufgabe

  Der KLuceneIndexJob verwaltet einen extern aufgebauten Lucene-Index.

KRemoveIndexJob

- Bezeichnung: Attribute aus dem Index entfernen

KSyncIndexJob

- Bezeichnung: Index synchronisieren

KAddAllToIndexJob, KRemoveIndexJob und KSyncIndexJob werden benötigt, um die internen Indizes zu pflegen.

3.1.2 KBrainbotJob

- Kategorie(n): <keine>
- Bezeichnung: KBrainbotJob

Der KBrainbotJob führt Aktionen zur Pflege eines Brainbot-Indexes aus.

Falls innerhalb der Konfiguration im AdminTool angegeben wird, dass Pflegeaktionen von einem Jobclient ausgeführt werden sollen ("Jobclient benutzen"), so muss ein Jobclient gestartet werden, damit die Pfege des externen Index ausgeführt wird.

Der KBrainbotJob hat keine weiteren Konfigurationsparameter in der ini-Datei, da die gesamte Konfiguration im AdminTool stattfindet.

3.1.3 KExternalCommandJob

- Kategorie(n): <keine>
- Bezeichnung: Externer Aufruf

Mit Hilfe des **KExternalCommandJobs** ist es möglich ausführbare Programme, die sich mit der Abarbeitung oder Veränderung von Dateien beschäftigen oder einfach nur aufgerufen werden sollen, anzusteuern. Eine Konfiguration in der INI-Datei des JobClients ist nicht notwendig. Der Job wird durch einen Skriptaufruf eingeworfen.

Das Hauptelement des Skriptaufrufes ist das Element ***ExternalCommandJob***. Mit dem Attribut *execution* kann eingestellt werden, ob der Job lokal ohne JobClient (Wert: *local*) oder mit JobClient (Wert: *remote*) ausgeführt werden soll. Der Standardwert ist *remote*.

Anmerkung zur Remote-Ausführung:

Eine Kontrolle über den Zugriff auf lokale Programme findet über den Aufruf einer Batchdatei statt. Bevor sich der JobClient einen KExternalCommanJob zur Ausführung nimmt, überprüft er, ob er diesen Job ausführen kann. Das ist der Fall, wenn im aktuellen Verzeichnis des JobClients die Batchdatei vorhanden ist, die im Element *Command*angegeben ist. Wird der aktuell anstehende Job von keinem JobClient zur Bearbeitung angenommen, ist die Job-Warteschlange für den Benutzer, der den Job eingeworfen hat, blockiert. Dieser Job muss von Hand gelöscht werden.

Das notwendige, erste Unterelement im Skript:

- **Command**: gibt an welche Batchdatei aufgerufen werden soll

```
<Command>convert.bat</Command>
```

In dem Element *Command* wird der Name der Batchdatei angeben. In der Batchdatei ist das Verzeichnis und das auszuführende Programm selbst angegeben. **Wichtig:** Die Batchdatei muss auf der gleichen Ebene wie das Programm (z.B. JobClient oder KB) liegen. Verzeichnisangaben im Element *Command* werden ignoriert.

Die weiteren Unterelemente werden von oben nach unten abegarbeitet. Falls die Reihenfolge der Parameter im externen Programm eine Rolle spielt, sollte dies berücksichtigt werden.

Skriptelemente, die die Parameter für den Aufruf bilden:

- **OptionString**: kann mehrfach verwendet werden. Es werden Parameter des aufzrufenden externen Programms als Zeichenketten angegeben. Die Parametereinträge müssen vollständig angegeben werden.

  ```
  <OptionString>-size 100x100</OptionString>
  ```

- **OptionPath**: der angegebene Path-Audruck wird ausgewertet und als Zeichenkette in den Kommandoaufruf eingebaut

  ```
  <OptionPath path="./topic()/concept()/@$size$"/>
  ```

Skriptelemente, die sich mit dem Handling von Attributen beschäftigen

- **SourceBlob**: Angabe des Blobattributes, das als Datenquelle verwendet wird

  ```
  <SourceBlob><Path path="$bild$"/></SourceBlob>
  <SourceBlob path="$bild$"/>
  ```

- **ResultAttribute**: Angabe der Parameter für die Erzeugung eines neuen oder die Veränderung eines bestehenden Blobattributs mit dem Inhalt der Datei bzw. der Datei selbst, die das Ergebnis des extern aufgerufenen Programms ist.
  Attributwerte:
  name: Name bzw. interner Name des anzulegenden Attributes
  topic: Zielindividuum des anzulegenden Attributes
  modifyExisting: verändern (*true*) oder neu anlegen (*false,* Standardwert)
  filename: Dateiname des anzulegenden Blobattributes

  ```
  <ResultAttribute
      name="$bild2$"
  ```

```
        topic="./topic()"
        modifyExisting="true"
        filename="convert_ +./valueString()"/>
    <ResultAttribute
        name="$bild2$"
        topic="./topic()"
        modifyExisting="true"
        filename="convert_ +./valueString()">
        <Path path="$bild2$"></ResultAttribute>
```

Beispiel 01:

Skript:

```
<Script>
    <ExternalCommandJob execution="local">
        <Command>convert.bat</Command>
        <OptionString>-size 100x100</OptionString>
        <SourceBlob><Path path="."/></SourceBlob>
        <OptionString>-geometry +5+10</OptionString>
        <SourceBlob><Path path="."/></SourceBlob>
        <OptionString>-geometry +35+30</OptionString>
        <OptionString>-composite</OptionString>
        <ResultAttribute
                    name="$bild2$"
                    topic="./topic()"
                    modifyExisting="true"
                    filename="convert_ +./valueString()"/>
    </ExternalCommandJob>
</Script>
```

Inhalt der Batchdatei unter Windows:

```
"C:\Program Files\ImageMagick-6.2.6-Q16\convert.exe" %*
exit /B %ERRORLEVEL%
```

Inhalt der Batchdatei unter Linux:

```
#!/bin/bash
convert $*
```

Beispiel 02:

Skript:

```
<Script>
    <ExternalCommandJob execution="local">
        <Command>convert2.bat</Command>
        <SourceBlob path="."/>
        <SourceBlob path="."/>
        <ResultAttribute
            name="$bild3$"
```

```
            topic="./topic()"
            modifyExisting="true"
            filename="convert2_ + ./valueString()"/>
    </ExternalCommandJob>
</Script>
```

Inhalt der Batchdatei unter Windows:

```
"C:\Program Files\ImageMagick-6.2.6-Q16\convert" -size 100x100 %1
-geometry +5+10 %2 -geometry +35+30 -composite %3
exit /B %ERRORLEVEL%
```

Inhalt der Batchdatei unter Linux:

```
#!/bin/bash

convert -size 100x100 $1 -geometry +5+10 $2 -geometry +35+30 -composite $3
```

Anmerkung: Die beiden Beispiele liefern als Ergebnis die gleiche Datei. In den Windows-Batchfiles dient der Exit-Befehl dazu, den Exit-Code von "convert" an den Aufruf zurückzuliefern.

Hier noch ein Beispiel für ein erweitertes Konvertierungsscript, welches mit den Parametern "Quelldatei", "Bildbreite" und "Zieldate" aufgerufen werden kann und welches nur breitere Bilder auf die angebene Breite verkleinert. Das Script schreibt außerdem eine Protokolldatei über die Konvertierung wobei auch Fehlermeldungen von Image Magick in die Logdatei geschrieben werden:

```
set MONTH_YEAR=%DATE:~-8%
echo Converting %1 to %3 (width: %2) >> convert%MONTH_YEAR%.log
convert.exe %1 -resize "%~2>" %3 2>> convert%MONTH_YEAR%.log
echo Conversion finished with exit code %ERRORLEVEL% >> convert%MONTH_YEAR%.log
exit /B %ERRORLEVEL%
```

Und hier noch die Version für Linux (Bash):

```
#!/bin/bash
FULLDATE=‘date +%c‘
MONTH_YEAR=‘date +%m.%Y‘
LOGFILE="convert.$MONTH_YEAR.log"
echo "$FULLDATE: Converting $1 to $3 (width: $2)">>$LOGFILE
convert "$1" -resize "$2>" "$3" 2>>$LOGFILE
EXITCODE="$?"
echo $FULLDATE: Conversion finished with exit code $EXITCODE>>$LOGFILE
exit $EXITCODE
```

3.1.4 KExtractBlobTextJob

- Kategorie(n): <keine>
- Bezeichnung: Blob in ein Textattribut umwandeln. Aus dem Blobattribut wird mithilfe der im Admin-Tool auf dem Reiter "Indexkonfiguration -> Externer Volltextfilter" angegebenen Batch-Datei der Textinhalt extrahiert und in einem neuen Attribut des angegebenen Textattributtyps abgelegt. Weitere mögliche Parameter für den Job sind das Topic, an dem der Extrakt angelegt werden soll, sowie die Sprache des anzulegenden Attributs, in dem Fall, dass das angegebene Textattribut mehrsprachig ist. Dieser Job wird von einem Trigger eingeworfen, der so angelegt sein sollte, dass er auf Erzeugen und Modifizieren von Blobattributen reagiert. Die dabei anzugebende KScript-Regel lautet "ExtractBlobText" und gestattet die Angabe der oben genannten Parameter.

3.1.5 KQueryJob

- Kategorie(n): query
- Bezeichnung: Suche

Dient der ausgelagerten Ausführung von einfachen und Expertensuchen auf einem Jobclient. Wird je nach Bedürfnissen der betrachteten Suche ausgestattet und ausgeführt.

3.1.6 KScriptJob

- Kategorie(n): <keine>
- Bezeichnung: KScriptJob

Mithilfe des KScriptJobs lassen sich KScripts aus KScript heraus so aufrufen, dass sie auf dem Job-Client ausgeführt werden. Dabei geschieht das Erzeugen des Jobs durch die KScript-Regel "ScriptJob", welche ausgestattet mit Script und den zu diesem Zeitpunkt errechneten Startobjekten als Ausgangspunkt den resultierenden KScriptJob in die Job-Queue einstellt. So lassen sich Arbeiten asynchron auf Job-Clients verteilen. Anwendungsbeispiele sind die Auslagerung von Tätigkeiten, die bei sequenzieller Ausführung den aufrufenden Klienten zu lange blockieren würden.

3.2 Beispiel für eine Ini-Datei

```
volume=MeinNetz host=localhost
jobPools=KSearchJob,KIndexJob,KExpertQueryJob cacheDir=jobcache
logfile=jobclient01.log
name=jobclient01
useProxyValueHolder=false
enableLowSpaceHandler=true maxMemory=400 ;debug=true
```

3.3 Erweiterungen ab K-Infinity-Core 3.2

Vorglühen des JobClients

Die JobClients lassen sich vorglühen, d.h. dass beim Hochfahren der JobClients gewisse,

durch die Konfiguration auswählbare Strukturen geladen werden. Durch diesen Vorgang steigt der Speicherbedarf des JobClients. Im Gegenzug kann der JobClient Fragen schneller beantworten, ohne viele Cluster nachladen zu müssen.

In die Ini-Datei des JobClients muss der Eintrag **keepClusterIDs** angegeben werden. Mögliche Werte für diesen Eintrag sind:

- **index** - Bei den Einstellungen der zusammensteckbaren Indexern gibt es die Möglichkeit, das Häkchen bei *Jobclient soll Index in den Hauptspeicher laden* zu setzen. Für die aktivierten Indexer wird ein Teil Ihrer Indexstruktur geladen.
- **protoOfSizes** - Die Anzahl der Individuen für jedes Konzept werden bereits beim Start ermittelt.
- **accessRights** - Das Root-Objekt des Rechtesystems wird in den Speicher geladen.

Wichtig: Für den Eintrag *useProxyValueHolder* muss der Wert *false* gesetzt sein. Sonst versucht der JobClient RPCs (Anfragen, die der Mediator beantworten kann) an den Mediator abzusetzen. Der Client soll jedoch die Cluster selber laden und unter Umständen auch im Speicher behalten.

Anmerkung: Es ist ebenfalls von Vorteil, für eine Performanceverbesserung den Festplattencache für den JobClient einzuschalten.

Beispiel für die Einträge in der INI-Datei:

```
[Default]
...
useProxyValueHolder=false
keepClusterIDs=index,protoOfSizes,accessRights
cacheDir=jobcache
maxCacheSize=1000
...
```

Clusterhalten des JobClients

Für die Jobs *KSearchJob* (heisst in 3.2 *KQueryJob*), *KExpertQueryJob*, *KTableRenderJob*, *KTableSortJob* und *KTableQueryJob* kann in der INI-Datei des *JobClients* der Eintrag *keepClusterIDs* auf *true* gesetzt werden. So werden nach dem Ausführen dieser Jobs die Cluster betrachtet, die bei der Abarbeitung geladen werden mussten. Alle Cluster, die nicht die Suchergebnisse enthalten, sondern die Indexstrukturen, werden der Menge, der im Speicher zu haltenden Cluster, hinzugefügt.

Wichtig: Auch hier muss für den Eintrag *useProxyValueHolder* der Wert *false* gesetzt sein.

Anmerkung: Ebenfalls ist es zu empfehlen den Festplattencache für den JobClient einzuschalten.

Beispiel für die Einträge in der INI-Datei:

```
[Default]
...
jobPools=KInfinity.KExpertQueryJob, KInfinity.KTableQueryJob, ...
...
```

[KInfinity.KExpertQueryJob]
keepClusterIDs=true

[KInfinity.KTableQueryJob]
keepClusterIDs=true