# Job Java Job

Mit diesem Job können komplexe Java-Programmierungen am Webdesk durchgeführt werden.

Da hierfür Webdesk-Pakete geladen werden müssen ist es empfehlenswert sich hierfür mit Workflow in Verbindung zu setzen.

Der Job kompiliert den im Konfigurationsdialog hinterlegten Java-Quellcode zur Laufzeit und führt ihn aus.

## Konfiguration

- **Java File Name**: Vollqualifizierter Klassenname (z. B. at.workflow.webdesk.custom.jobs.TestSystemOut). Dieser Name bestimmt Pfad und Klassenname des Quellcodes.
- **Java Code**: Vollständiger Java-Quellcode der Klasse. Die Klasse muss CustomJavaJob implementieren.
- **Write Source File**: Schreibt den Quellcode als Datei nach WEB-INF/work/runtime (Pfad entsprechend dem Klassennamen). Nützlich für Debugging.
- **Use Transaction**: Führt den Job in einer Datenbank-Transaktion aus; bei Fehlern erfolgt ein Rollback.

## Unterbrechen (Interrupt)

Ein Unterbrechen ist nur möglich, wenn die Klasse CustomInterruptableJavaJob implementiert. Andernfalls kann der Job nicht zuverlässig abgebrochen werden.

## Beispiel: Java-Job mit Info-Log Ausgabe

Dieses Beispiel zeigt ein einen minimalen Job, der eine Log-Ausgabe erzeugt (sowohl im Job-Logger, als auch im Standard Out)

### Beispiel-Java-Code (Skeleton)

```
package at.workflow.webdesk.custom.jobs;

import java.util.Date;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import at.workflow.webdesk.po.jobs.customJavaJob.CustomJavaJob;
import at.workflow.webdesk.tools.WebdeskLoggerUtil;

public class TestSystemOut implements CustomJavaJob {

    private static final Logger logger =
 WebdeskLoggerUtil.getJobLogger(TestSystemOut.class);

    public void run() {
        System.out.println("Some Output on the Console from this Custom Java Job at " + new
 Date());

        logger.info("Some output in the log");
    }

}
```

## Beispiel: Java-Job WriteActiveUserSessionsToCache

**Zweck:** Dieser Job liest aktuell aktive Benutzer-Sitzungen aus dem Sitzungs- bzw. Session-Service aus und schreibt zusammengefasste Metadaten in einen Cache. Ziel ist eine schnelle Verfügbarkeit von Informationen über angemeldete Benutzer für Monitoring und Dashboard-Funktionen.

**Funktionsweise (Kurz):**

1. Der Job holt sich einen Service, der aktive Sessions liefert (z. B. eine PoSessionService-Bean).
2. Für jede gefundene Session werden relevante Informationen extrahiert (z. B. Personen-ID, Benutzername, letzte Aktivität, Client-IP).
3. Die aggregierten Daten werden an einen Cache-Service übergeben (z. B. CacheService oder einen Applikations-Cache) und dort abgespeichert.
4. Der Job führt sinnvolle Fehlerbehandlung und Logging durch; bei Bedarf kann er als Probelauf ausgeführt oder mit zusätzlichen Filterkriterien (z. B. nur bestimmte Mandanten) konfiguriert werden.

## Beispiel-Java-Code (Skeleton)

Hinweis: Die dargestellten Bean-Namen und Methoden sind beispielhaft und müssen an Ihre Webdesk-API/ Version angepasst werden. Vor dem produktiven Einsatz in Testumgebungen prüfen und anpassen.

```
package at.workflow.webdesk.custom.jobs;

import java.util.List;

import org.apache.logging.log4j.Logger;

import at.workflow.webdesk.po.jobs.customJavaJob.CustomJavaJob;
import at.workflow.webdesk.tools.WebdeskLoggerUtil;

/**
 * Beispiel-Job: Liest aktive Benutzer-Sessions aus und schreibt eine kompakte
 * Repräsentation in den Applikations-Cache.
 */
public class WriteActiveUserSessionsToCache implements CustomJavaJob {

    private static final Logger logger =
 WebdeskLoggerUtil.getJobLogger(WriteActiveUserSessionsToCache.class);

    public void run() {
        logger.info("Start WriteActiveUserSessionsToCache");

        try {
            // Hinweis: Bean-Namen und Typen an Ihre Installation anpassen
            // Beispiel: Object sessionService = SpringContext.getBean("PoSessionService");
            Object sessionService = null; // TODO: app-spezifische Session-Bean holen
            Object cacheService = null;   // TODO: Cache-Bean holen (z. B. "CacheService")

            // Pseudocode: aktive Sessions abfragen
            // List sessions = ((PoSessionService)sessionService).findActiveSessions();
            List sessions = null; // TODO: Ersetzen durch echte Abfrage

            if (sessions == null || sessions.isEmpty()) {
                logger.info("Keine aktiven Sessions gefunden");
            } else {
                for (Object s : sessions) {
                    // TODO: Session-Objekt in eine kompakte Struktur uübersetzen
                    // Beispiel:
                    // String personId = ((Session)s).getPersonId();
                    // String userName = ((Session)s).getUserName();
                    // Date lastAccess = ((Session)s).getLastAccess();
                    // String clientIp = ((Session)s).getClientIp();

                    // Ergebnis-Objekt bauen und in Cache schreiben
                    // ((CacheService)cacheService).put("activeSession:" + personId,
 summaryObject);
                }
```

```
                logger.info("Gespeichert {} Sessions in Cache", sessions.size());
            }

        } catch (Throwable t) {
            logger.error("Fehler beim Schreiben der aktiven Sessions in den Cache", t);
            // Je nach Job-Framework: Fehler weiterwerfen oder behandeln
            throw new RuntimeException(t);
        }

        logger.info("Ende WriteActiveUserSessionsToCache");
    }
}
```

Weiteres Vorgehen:

- Passen Sie die Bean-Namen und Typsignaturen an die in Ihrer Umgebung verfügbaren Services an (z. B. PoSessionService, PoCacheService).
- Testen Sie den Job zuerst in einer Staging-Umgebung; prüfen Sie Logausgaben und Cache-Inhalte.
- Falls gewünscht, erweitern Sie den Job um Konfigurationsparameter (z. B. Mandant-Filter, Probelauf-Flag, E-Mail-Reporting).

# Felder

| Name | Wert |
|---|---|
| Modul | Portal & Organisation (po) |
| Webdesk Actionname | Java Job |
| Artefakt-Typ | Job |