Connecting External Job Management Systems to the SAP NetWeaver Scheduler for Java

Interface

SAP

JAVA-JXBP 7.1

Version 1.1

Java External Interface for Background Processing

Copyright

Contents

Symbols

Symbol	Meaning
⚠	Warning
💬	Example
💡	Tip
🧭	Recommendation
<syn>	Syntax

1. Introduction

This document describes the connection of an external job management system (often called an *external scheduler)* to the SAP NetWeaver Application Server Java (AS Java). For this purpose, SAP has defined a public interface called JXBP, which stands for Java eXternal interface for job Background Processing. JXBP is one of a range of open interfaces, which SAP makes available for system management tasks.

2. Overview

Chapter	Contents
3	General description of the external interfaces function.
4	Short introduction to SAP NetWeaver AS Java background processing. In this chapter you can find details for the Java Scheduler concept, architecture, and terminology.
5	Description of the requirements for using the JXBP interface. This chapter contains information of how to configure external schedulers in the SAP NetWeaver Administrator and the necessary security settings.
6	Certification requirements. This chapter describes, what you need, in order to certify your usage of the JXBP interface.
7	JXBP interface description

3. The Function of External Interfaces

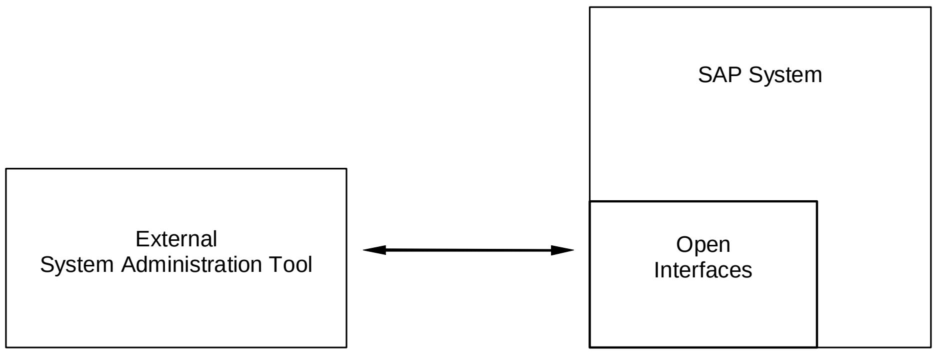


Fig. 3.1: Location of external interfaces

The external interfaces provide simple and seamless integration of the SAP system into both local administrative tools and business-wide system management infrastructures. The benefit from the integration is to provide the customer with a homogenous information infrastructure. The role of the interface is to facilitate the flow of information between the SAP system and the external tools.

4. A Short Introduction to AS Java Background Processing

The SAP NetWeaver Scheduler for Java is a part of the SAP NetWeaver AS Java systems version 7.1 Enhancement Package 1 SP4 or higher. It implements an object-oriented approach for jobs development and scheduling.. It provides low-level job scheduling capabilities for applications running on the AS Java, such as enabling the automated execution of tasks that applications can perform in the background.

For simplicity, in this documentation we refer to the SAP NetWeaver Scheduler Java as the Java Scheduler.

The scope and the features of the Java Scheduler are similar to those offered by the CCMS ABAP Scheduler (transactions SM36 and SM37) in AS ABAP.

The Motivation section below explains why background processing in general has a place in a dialog-oriented standard application, while the Architecture section explains the concept and the terminology concerning the Java Scheduler.

4.1 Motivation

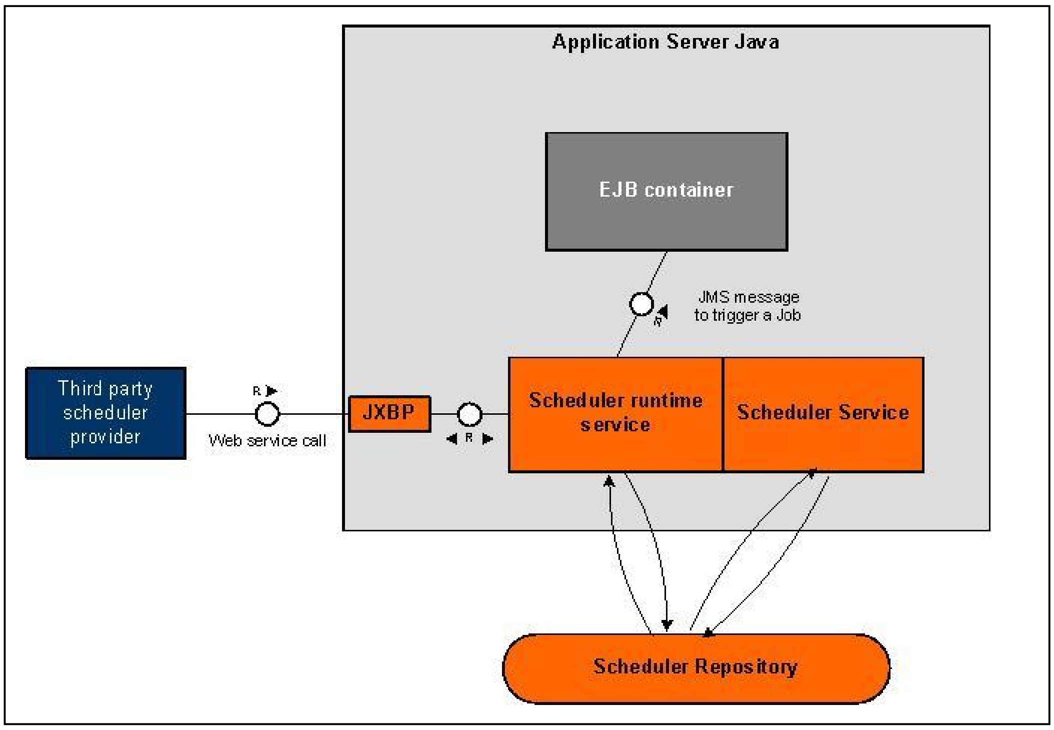
SAP NetWeaver is a platform used for interactive applications. In other words, the vast majority of tasks are carried out in dialog with the user. However, there are also good reasons for the inclusion of a background processing system in AS Java.
Besides the tasks carried out in a dialog, there are tasks processing large amounts of data and requiring lots of resources that do not need user interaction. With the help of the background processing system, such tasks are normally scheduled for times when the load on the system is low (during the nights, weekends, and so on), in order to avoid resource conflicts with the dialog users.
At the scheduled point in time, these tasks are started by the background processing system and executed without user interaction.
This mechanism is especially useful for tasks that have to be carried out periodically,

for example each week or each month. In the background processing system, these tasks – including the period - have to be specified once only. No further action is required from the user with respect to the regular execution.

4.2 Architecture and Concepts of the SAP NetWeaver Scheduler for Java

4.2.1 Architecture

The figure below outlines the architecture of the Java Scheduler:



Jobs are implemented on the basis of message-driven beans. A message-driven bean containing a job is called a JobBean. The execution of JobBeans is handled by the EJB container. A JobBean is executed when it receives a Java Messaging Service (JMS) message from the scheduler runtime service. In cluster environment, the JMS is responsible for load balancing: it decides which JobBean instance on which node gets the request to run. For a description of the job and job definition concept see section 4.2.2.

The Java Scheduler defines two services:

- Scheduler Runtime Service

  Controls all runtime aspects of a job. It handles the execution of jobs on the server node where it is running, provides error handling, and maintains job definitions and job runtime information, such as job parameters and log files.

- Scheduler Service

  Schedules jobs deployed on the application server and submits them to the scheduler runtime service. The scheduler service accepts requests for rescheduling, canceling and deleting jobs.

The two services store the complete state of the Java Scheduler in the database.

In cluster environment, the scheduler runtime service is deployed and runs on every cluster node. The scheduler service is also deployed on all cluster nodes, but it runs only on a single node at a time. The node where it runs is designated as the singleton node. If the singleton node goes down, the scheduler service gets activated on another cluster node. This mechanism ensures the scheduler service failover.

4.2.2 Jobs and Job Definition

The abstract logic of the work to be performed is stored in a job definition. The job definition is deployed on the server and implements the business logic of a job. When a job definition is scheduled with specific parameters and start conditions, a concrete instance of this job definition, called job, is executed. The same job definition can be scheduled with various parameters and start conditions.

Thus, a job runs once with particular parameter values at a particular point in time or upon a particular event and performs a certain amount of work. One job runs in one thread.

The Java Scheduler supports child jobs (see section 4.2.4) and job chains.

As a runtime object, a job has a life cycle characterized by job statuses (see section 4.2.3 below). Jobs also write logs (see section 4.2.5).

4.2.3 Job Statuses

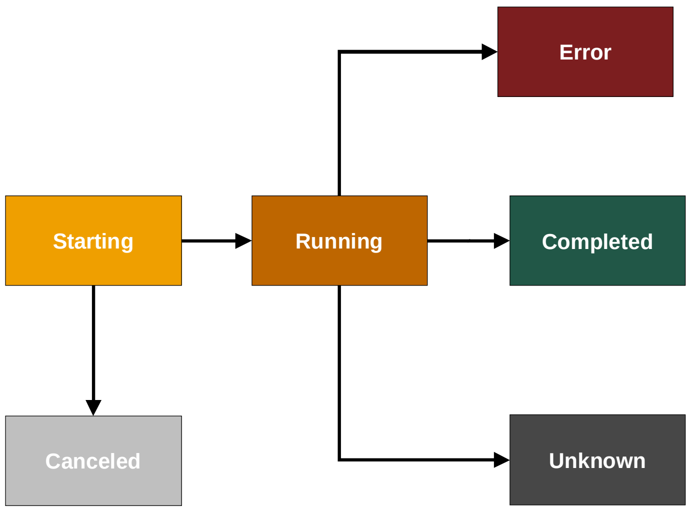
The status of a scheduled job signifies the job condition at a certain point in the job's life cycle. A job can be only in one status at a time. A job can be in any of the six job statuses outlined in the table below.

In its life cycle, a job always has exactly one of the following statuses:

Status	Description
Starting	The job is currently being started. The status *starting* is possible when a JMS message was sent to trigger the job but the job has not yet received it. This delay in the JMS message receipt is possible if currently there are not enough threads to run a job.
Running	The job is currently performing its unit of work.
Completed	The job has finished its unit of work.
Error	The job has completed its unit of work but threw an exception during execution or indicated failure by invoking an API method.
Unknown	The state of the job is not known. The status *unknown* is possible when a node, during its start up, detects that there are jobs currently running on it.
Canceled	The execution of the job was canceled while the job was in status *starting*. If you try to cancel a job in status *running*, it gets canceled cooperatively. The job acknowledges the request for cancellation but may or may not get canceled. • If the job gets canceled, it changes its status from *running* to either *completed*, or *error*. The job does

	not change to status *canceled*. • If the job does not get canceled, it remains in status *running*.

The figure below shows the possible job statuses and their transitions.



4.2.4 Parent/Child Functionality

The business process carried out by a job or by a collection of jobs, does not consist only of static jobs, which are known in advance and shown right away in the job overview. It also comprises jobs that are created at runtime, for example, to dynamically distribute workload. A job that is released by another job is called a child job, and the releasing job is called a parent job.

By using JXBP methods, the external scheduler can find out whether or not a job has child jobs. The SAP background processing system stores the parent/child data of jobs automatically and offers functions to access these data. JXBP offers functions to access the parent/child data of jobs. See also the method overview in section 6.2.

4.2.5 Events

The Java scheduler has an event concept. When starting a job, the external scheduler marks a job as running. When, for example, the job has been finished, the external scheduler needs to be informed for the status change. This information exchange takes place via events. An event is generated, whenever the job changes its status. Unhandled events are the new events. Events that have been already read are called 'handled' event. See also the method overview in section 6.2.

4.2.6 Logging and Tracing

The NetWeaver Scheduler for Java uses the standard SAP Logging framework to log messages on two levels:

- Job level (job logs)

  Logs at job level, or job logs, are logged in the database by every job. The following rules apply for job logs:

  - A job log is always associated with the job instance that logged it. The lifetime of the job log matches the lifetime of the job.

- The log for a job is deleted when the corresponding job is deleted, for example when the job’s retention period has expired.
- Job logs are not overwritten by a “Rotating log file set” strategy.
- You can retrieve a log written by a particular job no matter on which server process the job ran, or whether the node where the job ran is still part of the cluster.

The Java Scheduler allows managing the size of job records in the database. A job has a retention period denoting the number of days that a job record is persisted in the database. To prevent database overflow caused by too many job logs, in the job definition’s deployment descriptors job’s retention period can be configured.

- Scheduler level (Java Scheduler logs)

  By default, the Java Scheduler logs are logged under the `/System/Server` category, at SYS_SERVER.

5. Requirements for Using the JXBP Interface

The following requirements must be met so that an external scheduler can schedule a job in an AS Java system.

5.1 Configuring the External Scheduler in SAP NWA

You have to make the external job scheduler acquainted to the SAP Java system. This configuration task is executed in the SAP NetWeaver Administrator of the SAP Java system.

Proceed as follows to configure the external scheduler in the SAP NetWeaver Administrator:

1. Logon to the SAP NetWeaver Administrator, and choose *Operation Management -> Jobs -> Java Scheduler*.

   You see now the users interface of the SAP NetWeaver Java Scheduler in the SAP NetWeaver Administrator.

2. Choose *External Schedulers -> Add* to enter the configuration data for the external scheduler you want to make acquainted to the SAP Java system.

   The dialog window `Add External Scheduler` is displayed.

3. Enter the following data:

   - Name and description of the External Scheduler.
   - User

     The external scheduler accesses the Java system via a certain user. With this user the Java scheduler recognizes the external scheduler.

     The user entry depends on if there has already been a user defined and if the AS Java system is a standalone system or a double-stack system.

     - If it is a standalone system you can reuse existing user by mark option *Existing User* and entering his user ID. The alternative is to create a new user by enter a user name and enter and confirm a password for the new user.

     - In a dual-stack system, it is not possible to create a new user in the Java stack of the system. Therefore, in this case, you have to mark *Existing User* option and entering his user ID.

     The user can access the scheduler only via JXBP. And the external scheduler needs to know this user and password, so that it can connect to the Java

Scheduler. With the user name the scheduler authenticates itself, so that SAP NetWeaver knows which external scheduler connects.

- Inactivity Grace period

  As soon as an external scheduler is registered, it occupies resources on the server. And if this external scheduler does not connect to the Java system for a longer period, it would unnecessarily occupy resources that could be used otherwise. To prevent this, you can enter here an interval. If the external scheduler does not connect again within this interval, the resources are released for other purposes

  It is recommended to configure this interval to be non-infinite in order to avoid keeping redundant resources.

4. Choose *Add*.

5.2 Security settings

The client should invoke the JXBP Web service with the right credentials. In addition, the provided credentials must belong to a user, who is associated with the particular external scheduler.

6. Certification Requirements

If you would like to certify your usage of the JXBP external interface, there are some requirements to be fulfilled:

- You should have at least one valid SAP NetWeaver Developer User license. You can obtain a license at http://www.sap.com/community/survey/index.epx?SurveyID=1089
- You need SAP NetWeaver 7.1 Application Server Java installation. The supported version is 7.1 Enhancement Package 1 SP 4 or higher.
- The test catalogue, which is part of the same certification package (JAVAJXBP71TC.pdf), should be executed successfully.

The latest state of the requirements can be found in SAP Note 1396620.

7. JXBP - External Job Scheduling Interface (external job API)

Java eXternal Interface for Background Processing (JXBP) is a public API exposed as a Web service. It provides third-party scheduler providers with access to the Java Scheduler.

External schedulers cannot schedule jobs by using the scheduling capabilities of the Java Scheduler. They can use the scheduler only for job execution. For scheduling jobs, external schedulers should rely on their own scheduling capabilities.

Why does SAP offer the JXBP interface?

Many customers do not process their data with just one SAP system. They usually have a landscape consisting of one or more SAP systems as well as non-SAP systems. The non-SAP systems usually also have some kind of a background processing system.

There are interdependencies between the systems of such a landscape.

The non-SAP system A creates data using a background job. The SAP system B then processes this data in a job. This means that there is a job Y in SAP system B, which can only start after job X in non-SAP system A has finished.

Such a scenario demonstrates the need for a central job management system. The SAP background processing system cannot monitor jobs of non-SAP systems. In addition, the interdependencies between jobs even in a single system are sometimes so complex that they cannot be described with the functions of the internal batch API.

A central job management system (often referred to as 'external scheduler') connects to the SAP ABAP system via the XBP interface and to the SAP Java system via the JXBP interface.

In order to manage jobs centrally in a system landscape containing non-SAP systems, the non-SAP systems also have to provide an interface to which the external scheduler can connect.

7.1 What Is Required

In order to be able to work in the AS Java system, an external job scheduling system must be able to carry out the following activities within the AS Java system:

- Start jobs (immediately)
- Cancel running jobs
- Delete jobs
- Access information about jobs (status, log, and so on)

There are JXBP methods for carrying out all these activities.

7.2 JXBP Interface Description

The JXBP interface specifies the Java eXternal Batch Processing (JXBP) API. It can be used by external job scheduler in order to run jobs inside the SAP NetWeaver AS Java. The external scheduler needs to be registered with the scheduler execution runtime in order to use this service.

The interface methods are grouped into the following categories:

General Information Methods

Method	Description
String getSystemTimeZone() throws JXBPException	Returns the time zone where the cluster is located. It might throw `JXBPException` when the user is not authenticated or authorized. *Note that in cluster environment Java Scheduler expects the whole cluster to be run in a single time zone.*
getVersion()	Gets the version of the JXBP interface.

Job Definition Methods

Method	Description

Method	Description
JobDefinitionWS[] getJobDefinitions() throws JXBPException	Returns all job definitions known to the AS Java. It might throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
JobDefinitionWS getJobDefinitionByName (String jobName) throws JXBPException	Return the JobDefintion object for a given job definition name, where: • `jobName` - the name of searched job If there is no such job definition, return null. It might throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
JobDefinitionWS getJobDefinitionById (JobDefinitionID id) throws JXBPException	Gets a JobDefinition object by its job definition ID, where: • `id` – an identifier of searched job definition If there is no such job definition, return `null`. It can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.

Job Methods

Method	Description
JobID executeJob(JobDefinitionID jobDefId, JobParameterWS[] jobParametersWS, Integer retentionPeriod) throws ParameterValidationException, NoSuchJobDefinitionException, JXBPException	Submits the job for immediate execution, where: • `jobDefId` – an identifier of the job definition, which instance will be triggered for execution • `jobParametersWS` – an array of JobParameterWS consisting of all parameters required for execution. • `retentionPeriod` – overriding the default period, after which the job artifacts (job itself, job logs, job parameters, etc) can be automatically removed(by CleanJob) from scheduler database. Returns JobID of the submitted job. Can throw an exception when the provided parameters are not valid, or there is no job definition with provided `jobDefId` or when the user is not authenticated or authorized or when there is resource (like database) problem.
JobID executeJob(JobDefinitionID jobDefId, JobParameterWS[] jobParametersWS, Integer retentionPeriod, String vendorData) throws ParameterValidationException, NoSuchJobDefinitionException, JXBPException	Submits the job for immediate execution, where: • `jobDefId` – an identifier of the job definition, which instance will be triggered for execution • `jobParametersWS` – an array of JobParameterWS consisting of all parameters required for execution. The parameters can be obtained from the corresponding job definition. • `retentionPeriod` – overriding the default period, after which the job artifacts (job itself, job logs, job parameters, etc) can be automatically removed(by CleanJob) from scheduler database. • `vendorData` – data string to associate with the executing job. The maximum length allowed for

Method	Description
	vendor data is 200 characters. Returns JobID of the submitted job. Can throw an exception when the provided parameters are not valid, or there is no job definition with provided `jobDefId` or when the user is not authenticated or authorized or when there is resource (like database) problem or when vendor data sting is too long.
void cancelJob (JobID jobid) throws JXBPException, JobIllegalStateException, NoSuchJobException	Cancels a job, where: • `jobid` – an identifier of the job, which will be canceled. If the job has not been started it will immediately go into CANCELLED state. If it has been started, it will cooperatively try to abort the job. This method will just return with no indication whether the job was successfully cancelled or not. Can throw `JobIllegalStateException` if the job is not in status RUNNING, `NoSuchJobException` – if there is no such job or `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
void removeJob(JobID jobid) throws NoSuchJobException, JobIllegalStateException, JXBPException	Removes all information about this job instance from the SAP NetWeaver AS Java (including logs), where: • `jobid` – an identifier of the job to remove Can throw `JobIllegalStateException` if the job is not in status COMPLETED or ERROR or UNKNOWN or CANCELLED. Can throw also `NoSuchJobException` if a job with the given job id does not exist. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
void removeJobs (JobID[] jobids) throws JXBPException	Removes all records of the given job instances from the SAP NetWeaver AS Java (including logs), where: • `jobids` – identifiers of the jobs to use Can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem. This is a convenient method. Logical errors (e.g. one or more jobs in an illegal state) will be ignored.
JobWS getJob(JobID jobid) throws JXBPException	Returns the job for the given job ID, where: • `jobid` – the job identifier to use If there is no job with that `jobid`, return `null`. Can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
JobParameterWS[] getJobParameters (JobID jobid) throws JXBPException	Returns all parameters for the given job, where: • `jobid` – the job identifier to use Can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem or when there is no job with

Method	Description
	given `jobid`.
JobWS[] getJobs (JobID[] jobids) throws JXBPException	Return all jobs for the given job IDs, where: • `jobids` – the jobs identifiers to use Returns array with size equal to the count of found jobs, or null if no any jobs has found. Can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.

JobIteratorWS getJobs(JobFilterWS filterWS JobIteratorWS iterWS, int fetchSize)
throws JXBPException
	This method will return all the jobs which match the provided filter criteria. If the result set is bigger than the provided `fetchSize` parameter, the remaining entries can be retrieved using the returned iterator. The following code snipped shows how to retrieve the result from the call:
	`JobIteratorWS iter = jxbp.getJobs(myFilter, null, 1000);`
	`List<JobWS> jobs = iter.getJobs();`
	`//do something with the returned //job objects`
	
	`//if you want to get more jobs`
	`iter = iter.getJobs(myFilter, iter, 1000);`
	`List<JobWS> jobs = iter.getJobs();`
	
	Parameters:
	• `jobFilterWS` – the pre-initialize filter object (may be used to filter by status, start/end time, job id etc..)
	• `iterWS` – custom job iterator, which can be returned from the previous call. For first call it can be *null*.
	• `fetchSize` – indicate the maximum count of the records to be fetched.
	Return JobIteratorWS, which contains the result.
	Can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.

String getJobStatus(JobID jobid) throws NoSuchJobException, JXBPException	Returns the status of a job, where: • `jobid` – the job identifier to use The method can throw `NoSuchJobException` if there is no job for given id. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
JobIteratorWS getJobsByStatus(String s, JobIteratorWS iterWS, int fetchSize) throws JXBPException	Returns all jobs with particular status, where: • s – the status of searched job (e.g. STARTING, COMPLETED, ERROR etc.) • iterWS – custom iterator, which can be returned from the previous call. For first call it can be *null*. • `fetchSize` – indicate the maximum count of the records to be fetched.

Method	Description
	Return JobIteratorWS, which contains the result. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.

Child Job Methods

Method	Description
JobWS[] getChildJobs(JobID jobid) throws NoSuchJobException, JXBPException	Gets all child jobs of given job, where: • `jobid` –identifier of job, which is checked for child jobs Returns an array of JobWS with all the child jobs. Can throw `NoSuchJobException` if a job with the given job id does not exist. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
boolean hasChildJobs(JobID jobid) throws NoSuchJobException, JXBPException	Returns true if this job has at least one child job, where: • `jobid` – an identifier of the job, which is checked for child jobs Can throw `NoSuchJobException` if a job with the given job id does not exist. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem
boolean[] haveChildJobs(JobID[] jobids) throws JXBPException	Checks whether several jobs represented by their job IDs have child jobs, where • `jobids` – an identifiers of the jobs, which is checked for child jobs Return array of booleans. Each element indicates the result of check whether the corresponding `jobid` (from `jobids` array) has a child job.

Job Log Methods

Method	Description

LogIteratorWS getJobLog (JobID jobid,
LogIteratorWS it,
int fetchSize)
throws NoSuchJobException,
JXBPException
	This method will return the log for the given job in chunks. If the result set is bigger than the provided `fetchSize` parameter the remaining entries can be retrieved using the returned iterator. The following code snipped shows how to retrieve the result from the call.
	`LogIteratorWS iter =`
	`jxbp.getJobLog(jobId, null, 1000);`
	`String log = iter.nextChunk();`
	`//do something with the returned`
	`//log String`
	
	`//if you want to get more job log`
	`iter = jxbp.getJobLog(jobId, iter,`
	`1000);`
	`log = iter.nextChunk();`
	Parameters:

Method	Description
	• `jobid – an` identifier of job for which the log is retrieved. • `it` – custom job log iterator, which can be returned from the previous call. For first call it can be *null*. • `fetchSize –` indicate the maximum count of the records to be fetched. Return LogIteratorWS, which contains the result. Can throw `NoSuchJobException` if a job with the given job id does not exist. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
void removeJobLog(JobID jobid) throws NoSuchJobException, JobIllegalStateException, JXBPException	Removes the job log for the given job, where: • `jobid –` an identifier of job for which the log will be removed Can throw `NoSuchJobException` if a job with the given job id does not exist. Can throw also `JobIllegalStateException` if the job is not in status COMPLETED or ERROR or UNKNOWN or CANCELLED. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.

Vendor Data Methods

Method	Description
void setVendorData (JobID jobid, String data) throws NoSuchJobException, JXBPException	Associate vendor data with the given job, where: • `jobid` – an identifier of job for which the vendor data will be associated • `data` – the vendor data The maximum length allowed for vendor data is 200 characters. Can throw `NoSuchJobException` if a job with the given job id does not exist. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem or when data string is too long.
void setVendorData (JobID[] jobids, String data) throws NoSuchJobException, JXBPException	Associate vendor data with the given jobs, where: • `jobids` – an identifiers of jobs for which the vendor data will be associated • `data` – the vendor data The maximum length allowed for vendor data is 200 characters. The method ignores if there is no job for one or more of provided job ids. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem or when data string is too long.
String[] getVendorData (JobID[] jobIds) throws JXBPException	Returns vendor data associated with provided jobs ID, where: • `jobids` – an identifiers of job, which vendor data we are interested in. Can throw and `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.

Event Methods

Method	Description
EventWS[] getUnhandledEvents(int fetchSize) throws JXBPException	Returns all unhandled events, where: • `fetchSize` – indicate the maximum count of the records to be fetched. Returns all events for the current logged-in subscriber (external scheduler). The events which have been queued for this subscriber are marked as consumed and will not be returned anymore. Can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem or when `fetchSize` is not a positive number.
void clearEvents() throws JXBPException	Clears all events which have been returned by thw method `getUnhandledEvents`.

	Can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.
String[] getJXBPRuntimeEventTypes()	Returns string array of runtime event type names, which are used by the scheduler runtime.
void setFilter (String[] eventType) throws JXBPException	Allows a scheduler to specify which events it is interested in. If `setFilter` has not been called for a particular external scheduler, that scheduler will not be subscribed for any events. Parameters: • `eventType –` an array of event type names (e.g. `com.sap.scheduler.runtime.JobStarting, com.sap.scheduler.runtime. JobFinished)` Can throw `JXBPException` when the user is not authenticated or authorized or when there is resource (like database) problem.