IBM Workload Automation

Developer's Guide: Driving IBM Workload Automation

Version 9.5 Fix Pack 7

Note

Before using this information and the product it supports, read the information in Notices on page xl.

This edition applies to version 9, release 5, modification level 0 of IBM Workload Scheduler (program number 5698-WSH) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this guide

Provides an overview of the guide, with information about changes made to it since the last release, and who should read it. It also supplies information about obtaining resources and support from IBM.

Developer's Guide: Driving IBM Workload Automation introduces you to the application programming interfaces available to drive IBM Workload Automation products from your own applications.

What is new in this release

Learn what is new in this release.

For information about the new or changed functions in this release, see *IBM Workload Automation: Overview*, section *Summary of enhancements*.

For information about the APARs that this release addresses, see the IBM Workload Scheduler Release Notes at IBM Workload Scheduler Release Notes and the Dynamic Workload Console Release Notes at Dynamic Workload Console Release Notes. For information about the APARs addressed in a fix pack, refer to the readme file for the fix pack.

New or changed content is marked with revision bars.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully.

With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

For full information, see the Accessibility Appendix in the *IBM Workload Scheduler User's Guide and Reference*.

Technical training

Cloud & Smarter Infrastructure provides technical training.

For Cloud & Smarter Infrastructure technical training information, see: http://www.ibm.com/software/tivoli/education

Support information

IBM provides several ways for you to obtain support when you encounter a problem.

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

- Searching knowledge bases: You can search across a large collection of known problems and workarounds, Technotes, and other information.
- Obtaining fixes: You can locate the latest fixes that are already available for your product.
- Contacting IBM Software Support: If you still cannot solve your problem, and you need to work with someone from IBM, you can use a variety of ways to contact IBM Software Support.

For more information about these three ways of resolving problems, see the appendix about support information in *IBM Workload Scheduler: Troubleshooting Guide*.

Chapter 1. Introduction to driving IBM Workload Automation

Provides an overview of the entire publication.

Developer's Guide: Driving IBM Workload Automation describes the application programming interfaces which you can use to drive IBM Workload Automation products from your own applications.

Use the REST API application programming interface to create your own GUI or command-line interface to perform all the functions of the command-line programs composer, conman, and planman and the Dynamic Workload Console. This includes performing the following tasks in IBM Workload Scheduler and IBM Z Workload Scheduler:

- Modifying objects in the database
- Submitting workload
- Monitoring the plan
- Performing actions on the plan, such as remedial actions in the event that a job fails

The information about the application programming interfaces is organized as follows:

- Driving IBM Workload Scheduler with REST API on page 8

Chapter 2. Driving IBM Workload Scheduler with REST API

IBM Workload Scheduler provides a set of fully functional APIs that are implemented based on Representational State Transfer (REST) services. The REST API helps you easily integrate workload scheduling capabilities with external products and solutions. The same product functionality covered by the existing J2EE API is available with the REST API. The REST API is programming language independent and favors easier network configuration and firewall traversal. With the APIs, you can exploit heterogeneous environments and provide new automation opportunities with direct impact on productivity. The following are some examples or scenarios where the APIs can be implemented:

- Create your own graphical interface to create and modify scheduling definitions and update objects in the plan.
- Update definitions or plan objects within a script for integration or automation.
- When a specific event occurs within an external product, you can automatically submit a batch workload through IBM Workload Scheduler.
- In a managed file transfer solution, when a specific file arrives, you can submit one or more job flows that elaborate the file, closing the loop on your business process, whether it be bank transactions, a payroll process, or report generation. Your external managed file transfer product starts the business process and IBM Workload Scheduler takes care of the processing, assuring that it be monitored with the rest of the processes from a single point of control and eventually linked with other processes.

The IBM Workload Scheduler REST API provides several services to administer engines, event rules, workload modelling, plans, and security.

After installing your master domain manager or backup master domain manager, you can access the available REST API services by connecting to the following URL:

```
https://hostname:port_number/twsd
```

where,

hostname

The hostname of the master domain manager or the backup master domain manager.

port_number

The HTTPS port number of the master domain manager or backup domain manager. The default is 31116.

You can also access some IBM Workload Scheduler REST API samples here: REST API samples.

Chapter 3. Driving IBM Workload Automation with the Java™ API

This chapter describes the J2EE Application Programming Interface (API), which uses Enterprise Java™ Beans to drive IBM Workload Scheduler and IBM Z Workload Scheduler.

You can use the Java API to run all the tasks available in:

- the Dynamic Workload Console
- composer
- conman
- planman

Naming conventions

The naming conventions for the Java™ objects are quite straightforward. For example, to determine if a specific job definition in the database uses a command or a script, you use a method called `isCommand` in a class called `JobDefinition`.

The most important convention to remember is that an object in the database is differentiated from an object in the plan by the suffix "InPlan" to the object class name.

API detailed specification

Gives information on how the Javadoc API reference help can be accessed.

The full specification for the Java™ beans can be found in the Javadoc API reference online help. This is where all classes and methods are specified in detail. The full specification for the Java beans can be consulted in one of these ways:

- From the help of the **IBM Workload Scheduler Integration Workbench**, expand **Reference** and select **API reference**
- Open the following HTML file: `<`*`TWA_home`*`>/TWS/APIs/doc/Javadoc/index.html`

To obtain a description of the use of the different panes of the Javadoc API panel, click **Help**.

Classes in the *Deprecated* category should not be used.

IBM Workload Scheduler API projects

Describes API projects.

API projects

The projects here described are intended to connect to an instance of the corresponding version of IBM Workload Scheduler and interact with it using methods provided by Java™ API.

Structure of an API project

API project Wizards provide a structure containing everything you must need to connect to the required IBM Workload Scheduler instance:

Java source tree (src) on page 10

Separate directories for the source and class files.

Java libraries on page 11

A JRE System library and separate libraries are available for the IBM Workload Scheduler object and runtime jars.

A keys directory on page 11

A directory containing *.jks file needed to access through the IBM Workload Scheduler secure login.

A config directory on page 11

A directory containing all the configuration files you need to specify to connect to IBM Workload Scheduler.

One or more Java™ compilation units on page 10

A compilation unit includes a class that implements the Java™ interface for the connection to IBM Workload Scheduler. Another is an empty compilation unit with the classpath already configured and ready to be completed with the program logics you need.

build.xml on page 12

A standard ANT build file that you modify to suit your requirements.

Creating API projects

You create API projects in one of two ways:

Creating a project from scratch on page 12

You run a wizard, supplying information about what sort of project you want to create.

Creating a project from an API example on page 12

From a list of examples you select an API project similar to the project you want to create.

You then edit the new project so that it carries out the required task.

Java™ source tree (src)

Describes the Java™ source tree.

When you create an API project, it is set up as a Java™ project with separate folders for source and class files. The source folder is named `src`. It contains the Java™ code of the application.

A few Java™ classes are also created together with the new project.

Follow the IBM Workload Scheduler Java API reference to understand which method you need to implement. For logging and tracing in your code, use standard JSR-047 Java™ Logging APIs.

Java™ libraries

Describes the Java™ libraries.

IBM Workload Scheduler API projects are created with the following libraries:

Default JRE System library

Even if the default JRE is set by default, remember that the IBM Workload Scheduler event processor runs using IBM JDK version 1.5.

Use of IBM JDK version 1.5 is recommended for IBM Workload Scheduler API projects.

IBM Workload Scheduler library

This library contains all the IBM Workload Scheduler jars needed to implement IBM Workload Scheduler plug-ins or to use IBM Workload Scheduler APIs.

The library also defines the access rules for the classes in the jars: public APIs are defined as Accessible, while internal classes are defined as Discouraged.

IBM Workload Scheduler Runtime library

This library contains all the IBM Workload Scheduler jars needed at runtime by API based applications

Use of discouraged classes will be marked with compiler warnings by default. In any case the use of these classes is not supported.

Additional libraries needed for the plug-in Java™ code can be copied into the `lib` folder and added to the Java™ build path.

More details of these libraries are given in the Integration Workbench help.

Related links

Other project folders on page 11

Other project folders

Describes the other project folders.

config

Use this folder to store additional configuration files (such as property files) that the IBM Workload Scheduler administrator will need to edit for operation.

keys

This folder is used to store the *.jks key files needed to connect using the IBM Workload Scheduler secure login.

Java build path on page 11

build.xml

Describes the `build.xml` file created for an API project.

This is a standard ANT build file. You can modify it according to your needs.

Collected links

Other project folders on page 11

Creating a project from scratch

Describes how to create an API project from scratch.

With the Integration Workbench follow these steps to create API projects from scratch:

1. From the Integration Workbench select **Help →; Help Contents**
2. Expand **IBM Workload Scheduler Integration Workbench** and then **Integration workbench**
3. Expand **Driving IBM Workload Automation**
4. Select **Generating an API project**
5. The steps required to create the project are listed. Read them to understand what to do.
6. Follow the instructions to create the project. Integration Workbench creates a library containing all the product jars. Use your knowledge of Java™ products to create all the necessary coding and infrastructure to run the API.

Creating a project from an API example

Describes how to create an API project from an example.

Using the Integration Workbench you can create projects based on provided examples. Using this method you avoid the need to create the full API project structure from scratch. You choose an example which most approximates your requirements and then modify it accordingly.

To read how to use this facility, follow these steps:

1. From the Integration Workbench select **Help →; Help Contents**
2. Expand **IBM Workload Scheduler Integration Workbench** and then **Integration workbench**
3. Expand **Driving IBM Workload Automation**
4. Select **Creating a API project by example**
5. Create the project, following the instructions. Integration Workbench creates a project containing the full infrastructure and code for the chosen API.
6. To understand more about the API, open the project, select **Doc** and double-click **index.htm**

The examples you can choose from

Details the API example projects you can use as a template.

The examples you can select from are as follows:

AddEventRule

Work with event rules in the database.

MakeQueryJobsOnPlan

Work with dynamic scheduling job instances in the plan.

MakeQueryOnPlan

Work with objects in the plan.

MakeZOSQueryOnPlan

Work with objects in the plan for z/OS®.

RerunJobInPlan

Submit a job stream instance into the current plan rerun.

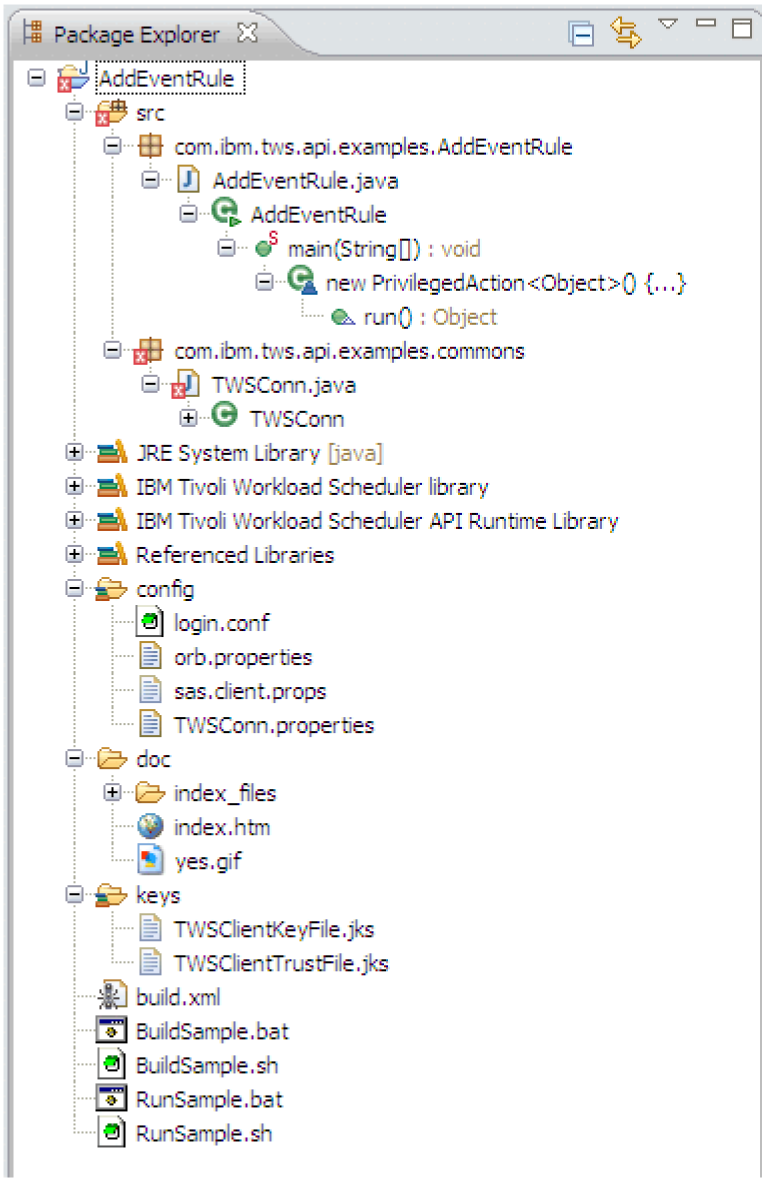
SubmitJobInPlan

Submit a job stream instance into the current plan.

Example IBM Workload Scheduler API project

Provides an example of an IBM Workload Scheduler API project.

The following figure shows the typical structure of a new IBM Workload Scheduler Java API project:



```
AddEventRule
  src
    com.ibm.tws.api.examples.AddEventRule
      AddEventRule.java
        AddEventRule
          main(String[]) : void
            new PrivilegedAction<Object>() {...}
              run() : Object
    com.ibm.tws.api.examples.commons
      TWSConn.java
        TWSConn
  JRE System Library [java]
  IBM Tivoli Workload Scheduler library
  IBM Tivoli Workload Scheduler API Runtime Library
  Referenced Libraries
  config
    login.conf
    orb.properties
    sas.client.props
    TWSConn.properties
  doc
    index_files
    index.htm
    yes.gif
  keys
    TWSClientKeyFile.jks
    TWSClientTrustFile.jks
  build.xml
  BuildSample.bat
  BuildSample.sh
  RunSample.bat
  RunSample.sh
```

The structure shown is for a Java API project named `AddEventRule`.

The Java™ classes contained in the `src` folder are described in Java™ source tree (src) on page 10.

The Java™ Build Path of the plug-in project is described in Java™ build path on page 11.

The role and contents of other project folders are described in Other project folders on page 11.

The ANT `build.xml` file is described in build.xml on page 12.

Java™ source tree (src) on page 10

Java™ build path on page 11

Other project folders on page 11

build.xml on page 12

Connecting to the products

Describes how to implement a connection to the IBM Workload Automation products using the API.

To connect to the products, you need to set the Connection Parameters to connect either to the IBM Workload Scheduler master domain manager (where there is a Connector installed) or to the IBM Z Workload Scheduler connector.

If you have created your project from an example, take the following steps, depending on which engine you are connecting to:

Connecting to the IBM Workload Scheduler master domain manager

1. From the Integration Workbench select your project
2. Expand **Config**
3. Edit the TWSConfig.properties file to obtain the correct parameters for your environment. The parameters are as follows:

```
TWSConfig.serverName=
TWSConfig.serverPort=
TWSConfig.userID=
TWSConfig.password=
TWSConfig.useSecureConnection=
TWSConfig.serverSecurePort=
```

where the parameters are as follows:

TWSConfig.serverName

The network name of IP address of the system where the IBM Workload Scheduler master domain manager is running (where there is a Connector installed). The default is "localhost".

TWSConfig.serverPort

The port used by the Connector on the master domain manager. The default is 31115.

TWSConfig.userID

The user ID with which the plug-in must authenticate. The default is "twsuser".

TWSConfig.password

The password of that user ID.

TWSConfig.useSecureConnection

Enter "true" to use a secure connection.

TWSConfig.serverSecurePort

If `TWSConfig.useSecureConnection` is set to "true", the secure port used by the Connector on the master domain manager.

Connecting to the IBM Z Workload Scheduler connector

1. From the Integration Workbench select your project
2. Expand **Config**
3. Edit the TWSConn.properties file to obtain the correct parameters for your environment. The parameters are as follows:

```
TWSConn.serverName=
TWSConn.serverPort=
TWSConn.userID=
TWSConn.password=
TWSConn.remoteServerName=
```

where the parameters are as follows:

TWSConn.serverName

The network name of IP address of the system where the z/OS® connector is installed. The default is "localhost".

TWSConn.serverPort

The port used by the z/OS® connector. The default is 31115.

TWSConn.userID

The user ID with which the plug-in must authenticate. The default is "twsuser".

TWSConn.password

The password of that user ID.

TWSConn.remoteServerName

The name of the z/OS® engine that you want to connect to.

If you have created a project from scratch, create an analogous structure of connection parameters.

Examples for IBM Workload Scheduler

Provides an overview of the examples available of using the Java™ API for IBM Workload Scheduler.

The following examples help you understand how Java beans are used. The examples are annotated with explanatory comments. In the javadoc reference (see API detailed specification on page 9), look up the objects used in the examples to see full details.

The examples are available in these groupings:

Working with objects in the database

Provides examples of using the Java™ API to work with objects in the database.

The following examples indicate how you use the classes to work with objects in the database:

Example 1: Adding a workstation to the database

```
//Object definition
String wksName = "MYWS";
Workstation wks = new Workstation();
wks.setName(wksName);
wks.setType(WorkstationType.FTA);
wks.setOs(OperatingSystem.UNIX);
wks.setAutoLink(true);
wks.setNodeName("node.abc.com");
wks.setSecurityLevel(SecurityLevel.NONE);



ConnModel myModel;
//Get an instance of ConnModel interface...
...

//Add the object
try
{
   myModel.addTWSObject(wks, null);
}
catch (ConnException e)
{
   //Do something to recover...
}
```

Example 2: Retrieving a workstation from the database

```
Workstation wksRead = new Workstation();
//Get the same workstation from the DB
try
{
wksRead = (Workstation) myModel.getTWSObject(Workstation.class,
      new FlowTargetKey(wksName), false, null);
}
catch (ConnException e)
{
   //Do something to recover...
 }
```

Example 3: Removing a workstation from the database

```
//Remove a workstation from the DB
 try
{
   myModel.removeTWSObject(Workstation.class, wksRead.getId(), null);
}
catch (ConnException exc)
{
   //Do something to recover...
}
```

Example 4: Defining a native job definition

```
//Job Definition creation

jobOk = new DistJobDefinition();
```

```
jobOk.setDescription("All values provided");
jobOk.setFlowTargetKey(wks);
jobOk.setName("JOB1");
jobOk.setTaskType("UNIX");
jobOk.setTaskString("ls");
jobOk.setUserLogin("tws_user");
jobOk.setRecoveryOption(RecoveryOption.STOP);
```

Example 5: Defining a job definition by using Jsdl

```
//Job Definition creation with Jsdl

jobPred4 = new DistJobDefinition();
jobPred4.setDescription("All values provided");
jobPred4.setFlowTargetKey(wksAgt);
jobPred4.setName("JSDLJOB");
jobPred4.setDefinedByJsdl(true);
jobPred4.setTaskString("<?xml version=\"1.0\" encoding=\"UTF-8\"?>" + "
<jsdl:jobDefinition xmlns:jsdl=\"http://www.abc.com/xmlns/prod/scheduling
  /1.0/jsdl\" " + "xmlns:jsdle=\"http://www.abc.com/xmlns/prod/scheduling/1.0/
  jsdle\">" + "<jsdl:application name=\"executable\">" +
    "<jsdle:executable interactive=
    \"false\">" + "<jsdle:script>hostname</jsdle:script>" + "</jsdle:executable>"
    + "</jsdl:application>" + "</jsdl:jobDefinition>");
jobPred4.setRecoveryOption(RecoveryOption.STOP);
```

Example 6: Adding a job stream definition with dependencies, jobs, and runcycle:

```
//Job Stream creation

JobStream js = null;
String jsName = "SBSCDBF1_1S";
String alias = "SBSCDBF1_1S";
js = new JobStream();
js.setName(jsName);
js.setFlowTargetKey(wks);
List<Job> joblist = js.getJobs();

//Jobs creation

Job jobPredecessor = new Job();
jobPredecessor.setName(jobPredName);
jobPredecessor.setJobDefinition(jobPred4);
jobPredecessor.setJobStreamKey((JobStreamKey) js.getKey());
Job jobSuccessor1 = new Job();
jobSuccessor1.setName(jobSucc1Name);
jobSuccessor1.setJobDefinition(jobOk);
jobSuccessor1.setJobStreamKey((JobStreamKey) js.getKey());

//Add Jobs into Job Stream

joblist.add(jobPredecessor);
joblist.add(jobSuccessor1);

//Add Dependencies to jobSuccessor1 from jobPredecessor

InternalDependency depend = new InternalDependency();
```

```
depend.setJobKey(new JobKey(jobOkName,(JobStreamKey) js.getKey()));
jobSuccessor1.getInternalDependencies().add(depend);

//Add a run cycle

RunCycle rcy = new RunCycle();
rcy.setCalendarKey(null);
rcy.setDescription("runCycleDescription");
rcy.setFreeDaysRule(FreeDaysRule.NEAREST_AFTER);
rcy.setICalendar("iCalendar");
rcy.setInclusive(false);
rcy.setName("runCycleName");
rcy.setOffsetType(null);
rcy.setOffsetValue(0);
rcy.getTimeRestrictions().setTimeDependent(true);
rcy.getTimeRestrictions().setStartOffset(43200000L);
rcy.getTimeRestrictions().setDeadlineOffset(14400000L);
rcy.getTimeRestrictions().setLatestStartOffset(3600000L);
rcy.getTimeRestrictions().setLatestStartAction(LateAction.CONTINUE);
rcy.setType(RunCycleType.SIMPLE);
rcy.setValidFrom(new Date(time - 86400000L));
rcy.setValidTo(new Date(time + 86400000L));
js.getRunCycles().add(rcy);
```

Working with objects in the plan

Provides examples of using the Java™ API to work with objects in the plan.

The following examples indicate how you use the classes to work with objects in the plan:

Example 4: Submitting a job stream instance into the current plan

This procedure requires the following main steps:

1. Obtain the required job stream definition from the database

   ```
   ConnPlan myPlan;
   //Get an instance of ConnPlan interface...
   ...

   String alias = "SBJBF1_1";
   JobStream js = null;
   JobStreamInPlan jsip = null;
   //If you already have a JobStream in the DB with Identifier jsDbID...
   try
   {
      //get it from the DB
      js = (JobStream)(myPlan.getTWSObject(JobStream.class,jsDbID,false,null));
   ```

2. Transform it into a JobStreamInPlan:

   ```
   //Transform it in a JobStreamInPlan.
   //TODAY is a variable representing the scheduled time
      jsip = myPlan.makeJobStreamInPlan(jsDbID, TODAY, alias, null);
   }
   catch (ConnException e)
   ```

```
{
              //Something went wrong...
}
catch (ConnEngineNotMasterException e)
{
   //Since the makeJobStreamInPlan is available also on FTAs
   //(it's on the Plan interface), an exception must be thrown
   //if it is called on an engine that is not the master
}
```

3. Add the JobStreamInPlan to the plan:

```
List idList = new ArrayList();
try
{
   //Add the job stream to the plan.
   //This method returns a list of Identifiers because the job stream can be
   //defined on a Workstation class, so you have an ID for each workstation
   //of the class
   idList = (ArrayList)myPlan.addJobStreamInstance(jsip, null);
}
catch (ConnException e)
{
   //...
}
catch (ConnEngineNotMasterException e)
{
   //...
}
```

Example 5: Making a query on the plan

The following example lists the first five jobs that begin with the letter "A":

```
String nameFilter = "A*";
int howMany = 5;

QueryFilter qf = new QueryFilter();
qf.setFilter(JobInPlanFilters.JOB_NAME, nameFilter);

QueryResult qr = null;
 try
{
          qr = myPlan.queryPlanObject(JobInPlan.class, qf, howMany, null);
}
catch (ConnException e)
{
   //...
}
```

Working with event rules in the database

Provides examples of using the Java™ API to work with event rules in the database.

The following examples indicate how you use the classes to work with event rules in the database:

Example 6: Adding an event rule to the database

Follow these steps:

1. Define the event rule:

```
String eventRuleName = "SampleEventRule";
String eventRuleDescription =
   "Define Event Rule; test MessageLoggerPlugIn and TWSObjectsMonitorPlugIn";
Date today = new Date(System.currentTimeMillis());
Date tomorrow = new Date(System.currentTimeMillis() + 86400000L);

//EventRule definition

EventRule er = new EventRule();
er.setName(eventRuleName);
er.setDescription(eventRuleDescription);
er.setRuleType(EventRuleType.FILTER);
er.setDraft(false);
er.setValidFrom(today);
er.setValidTo(tomorrow);
```

2. Define the event condition. In this case the condition is a job submission:

```
EventCondition evCond = new EventCondition();
evCond.setPluginName(TWSObjectsMonitorPlugIn.PLUGIN_NAME);
evCond.setEventType(JobUtil.EVENT_JOB_SUBMIT);
```

3. Define the conditions that the event condition has to satisfy to trigger the rule action (the filtering predicate):

```
String filterPred =   "<attributeFilter name=\"JobStreamWorkstation\"
                                          operator=\"eq\">"
                                          + "<value>MYWS</value>"
                                          + "</attributeFilter>"

+ "<attributeFilter name=\"JobStreamName\" operator=\"eq\">"
+ "<value>JS1</value>"
    + "</attributeFilter>"

+ "<attributeFilter name=\"JobName\" operator=\"eq\">"
+ "<value>JOB1</value>"
+ "</attributeFilter>"

+ "<attributeFilter name=\"Workstation\" operator=\"eq\">"
+ "<value>MYHOST</value>"
+ "</attributeFilter>"

+ "<attributeFilter name=\"Priority\" operator=\"range\">"
+ "<value>10</value>"
+ "<value>30</value>"
+ "</attributeFilter>"

+ "<attributeFilter name=\"Monitored\" operator=\"eq\">"
+ "<value>TRUE</value>"
+ "</attributeFilter>"

+ "<attributeFilter name=\"EstimatedDuration\" operator=\"ge\">"
 + "<value>400</value>"
 + "</attributeFilter>"

+ "<attributeFilter name=\"Login\" operator=\"eq\">"
+ "<value>TWSUser</value>"
+ "</attributeFilter>"

+ "<attributeFilter name=\"EveryFrequency\" operator=\"ge\">"
+ "<value>400</value>"
+ "</attributeFilter>";
```

4. Complete the event condition:

```
evCond.setFilteringPredicate(filterPred);
```

5. Add the event condition to the event rule:

```
er.getTriggerEvents().add(evCond);
```

6. Define the rule action. In this example, the rule action logs a message in the database:

```
RuleAction action = new RuleAction();
action.setPluginName(MessageLoggerPlugIn.PLUGIN_NAME);
action.setActionType(MessageLoggerPlugInConstants.ACTION_TYPE_MESSAGE_LOG);
action.setDescription("Adding the Message logger Plugin");
action.setResponseType(RuleResponseType.ON_DETECTION);
```

7. Define the value for the rule action parameter:

```
Map parameterMap = new HashMap();
parameterMap.put(MessageLoggerPlugInConstants.MESSAGE, "message");
parameterMap.put(MessageLoggerPlugInConstants.OBJECT_KEY, "object key");
```

8. Complete the rule action:

```
action.getParameterMap().putAll(parameterMap);
```

9. Add the rule action to the event rule:

```
er.getActions().add(action);
```

10. Add the event rule to the ConnModel interface:

```
ConnModel myModel = null;
//Get an instance of ConnModel interface...
//...

//Add the object

Identifier erId = null;
try
{
 erId = myModel.addTWSObject(er, null);
}
catch (ConnException e)
{
 //Do something to recover...
}
```

Example 7: Retrieve an event rule from the database by ID

Follow these steps:

1. Obtain the event rule ID to be retrieved by any means appropriate to your interface
2. Retrieve the event rule:

```
EventRule eRuleRead = new EventRule();
try
{
 eRuleRead =
      (EventRule) myModel.getTWSObject(EventRule.class, erId, false, null);
}
catch (ConnException e)
{
 //Do something to recover...
}
```

Example 8: Retrieve an event rule from the database by key (name)

Follow these steps:

1. Obtain the event rule key (name) to be retrieved by any means appropriate to your interface
2. Retrieve the event rule:

```
EventRule eRuleRead = new EventRule();
try
{
 eRuleRead =
    (EventRule) myModel.getTWSObject(EventRule.class,
                                new EventRuleKey(eventRuleName), false, null);
}
catch (ConnException e)
{
 //Do something to recover...
}
```

Example 9: Delete an event rule from the database by ID

Follow these steps:

1. Retrieve by ID the event rule to be deleted, as shown in example 7.1
2. If the event rule has been successfully retrieved, delete it:

```
{
 myModel.removeTWSObject(EventRule.class, eRuleRead.getId(), null);
}
catch (ConnException exc)
{
 //Do something to recover...
}
```

Example 10: Delete an event rule from the database by key (name)

Follow these steps:

1. Retrieve by key, the event rule to be deleted, as shown in example 7.2
2. If the event rule has been successfully retrieved, delete it:

```
{
 myModel.removeTWSObject(EventRule.class,
                                     new EventRuleKey(eventRuleName), null);
}
catch (ConnException exc)
{
 //Do something to recover...
}
```

Examples for IBM Z Workload Scheduler

Provides an overview of the examples available of using the Java™ API for IBM Z Workload Scheduler.

The following examples help you to understand how the beans are used in the IBM Z Workload Scheduler environment:

Example 1: Adding a workstation to the database

```
// Create the connection to the server
TWSZConn connection = new TWSZConn();
// Get an instance of the interface ZConnModel
final ZConnModel model = connection.getModelBean();

// Define the workstation properties
String wksName = "CPU1";
String wksDescription = "Added by API";
String wksPrintoutRouting = "SYSOUT";
Workstation wks = new Workstation();
wks.setName(wksName);
wks.setDescription(wksDescription);
wks.setType(WorkstationType.COMPUTER);
wks.setReportingAttribute(WorkstationReportingAttribute.AUTOMATIC);
WorkstationZOSAttributes wksAttr = new WorkstationZOSAttributes();
wksAttr.setDefaultTransportTime(3600);
wksAttr.setDefaultJobDuration(60);
wksAttr.setPrintoutRouting(wksPrintoutRouting);
wksAttr.setStartedTaskSupported(true);
wks.setZosAttributes(wksAttr);

try {
 Context context = new Context();
 // Add the workstation to the database
 model.addTWSObject(wks, context);
}
catch (ConnException e) {
 // Do something to recover
}
```

Example 2: Adding a job stream (application) to the database

```
// Create the connection to the server
TWSZConn connection = new TWSZConn();
// Get an instance of the interface ZConnModel
final ZConnModel model = connection.getModelBean();

final static Date TODAY =
new Date((System.currentTimeMillis()/86400000L) * 86400000L);
```

```
// Define the job stream properties
String jsName = "APPL";
String jsOwnerName = "API";
JobStream js = new JobStream();
js.setName(jsName);
js.setOwnerName(jsOwnerName);
js.setValidFrom(TODAY);
js.setPriority(5);
// Define a JCL job to add to the job stream
String jobName = "1";
String wksName = "CPU1";
String jclName = "MYJCL";
Job jclJob = new Job();
jclJob.setName(jobName);
jclJob.setEstimatedDuration(1000);
jclJob.setPriority(-1);
ZOSJobDefinition jobDef = new ZOSJobDefinition();
jobDef.setFlowTargetKey(new FlowTargetKey(wksName));
jobDef.setTaskType(TaskTypes.ZOS_JOB_TASK);
jobDef.setJclName(jclName);
jclJob.setJobDefinition(jobDef);
// Add the JCL job to the job stream
js.getJobs().add(jclJob);
// Define a Printer job to add to the job stream
jobName = "2";
wksName = "PRNT";
Job printerJob = new Job();
printerJob.setName(jobName);
printerJob.setEstimatedDuration(1000);
printerJob.setPriority(-1);
jobDef = new ZOSJobDefinition();
jobDef.setFlowTargetKey(new FlowTargetKey(wksName));
jobDef.setTaskType(TaskTypes.ZOS_ PRINTER_TASK);
jobDef.setJclName(jclName);
jobDef.setLimitForFeedback(102);
printerJob.setJobDefinition(jobDef);
// Add to the Printer job the dependency from the JCL job
List printerJobDeps = printerJob.getInternalDependencies();
InternalDependency depFromJclJob =
new InternalDependency(null, (JobKey)jclJob.getKey());
printerJobDeps.add(depFromJclJob);
// Add the Printer job to the job stream
js.getJobs().add(printerJob);

try {
 Context context = new Context();
 // Add the job stream to the database
```

Example 3: Modifying a job stream (application) in the database

```
// Create the connection to the server
TWSZConn connection = new TWSZConn();
// Get an instance of the interface ZConnModel
final ZConnModel model = connection.getModelBean();

final static Date TODAY =
new Date((System.currentTimeMillis()/86400000L) * 86400000L);
```

```
final static Date TOMORROW =
new Date(((System.currentTimeMillis()/86400000L) * 86400000L) + 86400000L);

final static long HOUR = 3600000;

// Make the job stream key
String jsName = "APPL";
JobStreamKey jsKey = new JobStreamKey(jsName, TODAY, false, false);

try {
 Context context = new Context();
 // Get the job stream by its key
 JobStream js =
         (JobStream)model.getTWSObject(JobStream.class, jsKey, false, context);

 // Define a run cycle to add to the job stream
  String rcName = "RCRULE";
  String rcDescription = "Added by API";
  RunCycle rcRule = new RunCycle();
  rcRule.setName(rcName);
  rcRule.setDescription(rcDescription);
  rcRule.setType(RunCycleType.RULE);
  rcRule.setValidFrom(TODAY);
  rcRule.setValidTo(TOMORROW);
  rcRule.getTimeRestrictions().setStartOffset(10*HOUR);
  rcRule.getTimeRestrictions().setDeadlineOffset(12*HOUR + 24* HOUR);
  rcRule.setFreeDaysRule(FreeDaysRule.DO_NOT_SELECT);
  rcRule.setICalendar("ADRULE ONLY(001 003) LAST(001 002) DAY(DAY
MONDAY THURSDAY) MONTH(FEBRUARY APRIL JUNE SEPTEMBER NOVEMBER) YEAR  ");

  // Add the run cycle to the job stream
  js.getRunCycles().add(rcRule);
  // Modify the job stream in the database
  Identifier jsId = model.setTWSObject(js, true, true, context);
}
catch (ConnException e) {
 // Do something to recover
}
```

Example 4: Adding a special resource in the database

```
// Create the connection to the server
TWSZConn connection = new TWSZConn();
// Get an instance of the interface ZConnModel
final ZConnModel model = connection.getModelBean();

final static long HOUR = 3600000;

// Define the resource properties
String resName = "RES";
String resDescription = "Added by API";
String wksName1 = "CPU1";
String wksName2 = "CPU2";
Resource res = new Resource();
res.setName(resName);
res.setDescription(resDescription);
```

```
FlowTargetKey wksKey1 = new FlowTargetKey(wksName1);
FlowTargetKey wksKey2 = new FlowTargetKey(wksName2);
res.getConnectedWorkstationLinks().add(new WorkstationLink(wksKey1));
res.getConnectedWorkstationLinks().add(new WorkstationLink(wksKey2));
ResourceBaseConstraints resCon = new ResourceBaseConstraints();
resCon.setQuantity(30);
resCon.setUsedFor(ResourceUsage.CONTROL);
resCon.setActionOnError(ResourceActionOnError.KEEP);
resCon.setAvailable(YesNoDefaultOption.NO);
res.setDefaultConstraints(resCon);
ResourceAvailabilityInterval resInt =
new ResourceAvailabilityInterval();
resInt.setIntervalValidityDayOfWeek(Calendar.MONDAY);
resInt.setIntervalStartTime(10*HOUR);
resInt.setIntervalEndTime(21*HOUR);
resInt.setQuantity(20);
resInt.setAvailable(YesNoDefaultOption.YES);
res.getResourceAvailabilityIntervals().add(resInt);

try {
 Context context = new Context();
 // Add the resource to the database
 model.addTWSObject(res, context);
}
catch (ConnException e) {
 // Do something to recover
}
```

Example 5: Adding a job stream to the plan after modifying its contents

The program uses Java APIs to modify an existing job stream and to apply variable substitution before submitting it to the plan.

The following program adds job stream `STREAM1` to the plan. Before doing this, the program also:

1. Adds two jobs to `STREAM1` after getting their attributes from two jobs extracted from job streams `STREAM2` and `STREAM3`.
2. Sets up a number of JCL promptable variables before submitting job stream `STREAM1`.
3. Further adds/modifies specific job attributes before releasing the jobs in the plan.

After importing all the necessary classes and objects, connecting to the scheduler, and declaring the required variables, the program:

- Fetches a job from job stream STREAM2 in the data base, gets its properties and stores them in a `ZosJobInfo` container called `jZos1`, and sets other information such as the input arrival time and the associated workstation and resources.
- Repeats these actions for job with job number 10 of job stream STREAM3, storing the job properties in a `ZosJobInfo` container called `jZos2`.
- Adds the two jobs to job stream STREAM1, and modifies their properties to define their numbers (which constitute their IDs within the job stream) and names.

- Defines first general JCL variables for all four jobs in job stream STREAM1 (using the `variablesToBeSubstituted` API) and then particular variables for each job (using the `jobVariablesToBeSubstituted` API).
- Adds job stream STREAM1 to the plan.
- Makes final changes in the job stream to add more properties, such as the extended job name and a special resource, to one of the jobs and then releases the job.

```
package com.ibm;

import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.List;


import com.ibm.tws.conn.exception.ConnEngineNotMasterException;
import com.ibm.tws.conn.exception.ConnException;
import com.ibm.tws.conn.exception.ConnLockingException;
import com.ibm.tws.conn.exception.ConnNotFoundException;
import com.ibm.tws.conn.exception.ConnSecurityException;
import com.ibm.tws.conn.exception.ConnTransportException;
import com.ibm.tws.conn.exception.ConnValidationException;
import com.ibm.tws.conn.util.Context;
import com.ibm.tws.conn.util.QueryResult;
import com.ibm.tws.objects.filter.JobStreamFilters;
import com.ibm.tws.objects.filter.QueryFilter;
import com.ibm.tws.objects.filter.WorkstationFilters;
import com.ibm.tws.objects.model.Job;
import com.ibm.tws.objects.model.JobStream;
import com.ibm.tws.objects.model.JobStreamHeader;
import com.ibm.tws.objects.model.ResourceDependency;
import com.ibm.tws.objects.model.Workstation;
import com.ibm.tws.objects.model.WorkstationHeader;
import com.ibm.tws.objects.model.ZOSJobDefinition;
import com.ibm.tws.objects.plan.JobInPlan;
import com.ibm.tws.objects.plan.JobStreamInPlan;
import com.ibm.tws.objects.plan.ResourceDependencyInPlan;
import com.ibm.tws.objects.plan.ResourceInPlanKey;
import com.ibm.tws.objects.plan.WorkstationInPlanKey;
import com.ibm.tws.objects.plan.ZOSJobDefinitionInPlan;
import com.ibm.tws.objects.plan.types.DependenciesResolutionOption;
import com.ibm.tws.objects.plan.types.JobInPlanZOSAttributes;
import com.ibm.tws.objects.plan.utils.ZosJobInfo;
import com.ibm.tws.objects.types.Identifier;
import com.ibm.tws.zconn.model.ZConnModel;
import com.ibm.tws.zconn.plan.ZConnPlan;
import com.ibm.tws.zconn.plan.dao.impl.util.ResourceInPlanHelper;
import com.ibm.tws.zconn.plan.dao.impl.util.WorkstationInPlanHelper;

@SuppressWarnings("restriction")
public class MainWitRes {

 public static void main(String[] args) {
  //Create a connection to the server
  TWSZConn connection=new TWSZConn();

  //Get Model or Plan Bean
  final ZConnModel model=connection.getModelBean();
  final ZConnPlan plan=connection.getPlanBean();
  final Context context=new Context();
 com.ibm.websphere.security.auth.WSSubject.doAs
 (connection.getSubject(), new java.security.PrivilegedAction<Object>() {
  /* (non-Javadoc)
   * @see java.security.PrivilegedAction#run()
   */
  @SuppressWarnings("unchecked")
  public Object run() {
   //Variable Declaration
HashMap<String, List<Integer>> dependencyToDelete=new HashMap<String, List<Integer>>();
HashMap<String, List<Integer>> dependencyToAdd=new HashMap<String, List<Integer>>();
 HashMap<Integer, String[][]> jobVariablesToBeSubstituted=new HashMap<Integer, String[][]>();
 List<ZosJobInfo> jobsToDelete=new ArrayList<ZosJobInfo>();
 List<ZosJobInfo> jobsToAdd=new ArrayList<ZosJobInfo>();
 List<Identifier> identifierList=null;
 List<Integer> successorList=new ArrayList<>();
 List<JobStreamHeader> jobStreamHeaderList;
 List<WorkstationHeader> workstationHeaderList;
 List<ZosJobInfo> jobsToModify=new ArrayList<ZosJobInfo>();
  List<ResourceDependencyInPlan> resourceDependencyInPlanList;
   Date startTime=new Date();
   Date deadlineTime=new Date();
   Long time1,time2;
   int seconds,minutes,hours;
   int priority=5;
   int jobNum=0;
   Integer succ;
   String [][] variablesMap1=new String [3][2];
   String[][] variablesToBeSubstituted=new String [1][2];
   String [][] variablesMap2=new String [2][2];
   String resourceName;
   String authorityGroup=null;
   String JSName="STREAM1";
   String description="";
   String group=null;
   String owner=null;
   String ownerDescription=null;
   String variableTable="STREAM1";
   String jobStreamName1="STREAM2";
   String jobStreamName2="STREAM3";
   boolean holdAll=true;
   JobStream jobStream1=null;
   JobStream jobStream2=null;
   Job jobFM1=null;
   Job jobFM2=null;
   JobStreamInPlan jobStreamInPlan=null;
   ZOSJobDefinition zosJobDefinition1=null;
   ZOSJobDefinition zosJobDefinition2=null;
   ZosJobInfo zosJobInfo1=new ZosJobInfo();
   ZosJobInfo zosJobInfo2=new ZosJobInfo();
   ZosJobInfo zosJobInfo3=new ZosJobInfo();
   ZosJobInfo zosJobInfo4=new ZosJobInfo();
   QueryFilter queryFilter;
   QueryResult queryResult;
   Calendar calendar1,calendar2;
   Workstation workstation = null;
   ZOSJobDefinitionInPlan jobDefinitionInPlan;
   ResourceDependencyInPlan resourceDependencyInPlan;
   ResourceInPlanKey resourceInPlanKey;
   WorkstationInPlanKey workstationInPlanKey;
   JobInPlanZOSAttributes jobInPlanZosAtt;
        //Getting jobStream="Stream2" from the DB, first the header then the full jobStream
   queryFilter=new QueryFilter();
   queryFilter.setFilter(JobStreamFilters.JOB_STREAM_NAME, jobStreamName1);
   try {
       queryResult=model.queryTWSObject(JobStream.class, queryFilter, 1, context);
       jobStreamHeaderList=(List<JobStreamHeader>) queryResult.getList();
       if(jobStreamHeaderList.size()>0){
       jobStream1=(JobStream) model.getTWSObject(JobStream.class,jobStreamHeaderList.get(0).getKey() , false, context);
       }
         } catch (ConnTransportException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnValidationException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnSecurityException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (RemoteException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    }

    //Getting the first job from Stream2 to add to STREAM1
    if(jobStream1.getJobs().size()>0){
     jobFM1=(Job) jobStream1.getJobs().get(0);
     zosJobDefinition1=(ZOSJobDefinition) jobFM1.getJobDefinition();
    }
         //Creation of ZosJobInfo for the first job
    ZosJobInfo jZos1=new ZosJobInfo();

    //Setting the inputArrivalTime
    calendar1=Calendar.getInstance();
    time1=jobFM1.getTimeRestrictions().getStartOffset();
    if(time1!=null && time1>0){
     seconds=(int) (time1 / 1000) % 60 ;
     minutes=(int) ((time1 / (1000*60)) % 60);
     hours=(int) ((time1 / (1000*60*60)) % 24);
     calendar1.set(Calendar.HOUR,hours);
     calendar1.set(Calendar.MINUTE, minutes);
     calendar1.set(Calendar.SECOND, seconds);
     jZos1.setInputArrivalTime(calendar1);
    }
    //Setting the zosJobInfo with data in zosJobDefinition and job (&timeRestriction)
         String workstationN=zosJobDefinition1.getFlowTargetKey().getName();
     jZos1.setJobName(zosJobDefinition1.getJclName());
     jZos1.setTextDescription(jobFM1.getDescription());
     jZos1.setWorkstationName(workstationN);
      jZos1.setDuration(jobFM1.getEstimatedDuration());
     jZos1.setJobNumber(44);
     jZos1.setAutoSubmit(zosJobDefinition1.getAutoSubmit());
     jZos1.setTaskType(zosJobDefinition1.getTaskType());
     jZos1.setTimeDependent(jobFM1.getTimeRestrictions().isTimeDependent());
     jZos1.setCentralizedScript(zosJobDefinition1.getHasCentralizedScript());

     //Getting the workstation from the DB -> to get the type associated, first get the header and then the full
workstation.
     queryFilter.setFilter(WorkstationFilters.WORKSTATION_NAME, workstationN);
     try {
           queryResult=model.queryTWSObject(Workstation.class, queryFilter, 1, context);
           workstationHeaderList=(List<WorkstationHeader>) queryResult.getList();
           if(workstationHeaderList.size()>0){
             workstation=(Workstation) model.getTWSObject(Workstation.class,workstationHeaderList.get(0).getKey() ,
               false, context);
       }
         } catch (ConnTransportException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
```

```
        } catch (ConnValidationException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
        } catch (ConnSecurityException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
        } catch (ConnException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
        } catch (RemoteException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
        }

   jZos1.setWorkstationType(workstation.getType());

   //Setting the Resource value
for(ResourceDependency resourceD:(List<ResourceDependency>) jobFM1.getResourceDependencies()){
    String rName=resourceD.getResourceKey().getName();
    if(rName!=null && rName.compareToIgnoreCase("Resource1")==0){
     jZos1.setR1(resourceD.getQuantity());
    }
    if(rName!=null && rName.compareToIgnoreCase("Resource2")==0){
     jZos1.setR2(resourceD.getQuantity());
    }
    if(rName!=null && rName.compareToIgnoreCase("ParallelServers")==0){
     jZos1.setParallelServer(resourceD.getQuantity());
    }
   }

   //Getting jobStream="STREAM3" from the Db, first the header then the full jobStream
   queryFilter.setFilter(JobStreamFilters.JOB_STREAM_NAME, jobStreamName2);
        try {
   queryResult=model.queryTWSObject(JobStream.class, queryFilter, 1, context);
   jobStreamHeaderList=(List<JobStreamHeader>) queryResult.getList();
    if(jobStreamHeaderList.size()>0){
     jobStream2=(JobStream) model.getTWSObject(JobStream.class,jobStreamHeaderList.get(0).getKey() ,
           false, context);
    }
   } catch (ConnTransportException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
   } catch (ConnValidationException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
   } catch (ConnSecurityException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
   } catch (ConnException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
   } catch (RemoteException e) {
    //TODO Auto-generated catch block
    e.printStackTrace();
   }

   //Starting with second job to add to STREAM1
   //Getting the job with number=10 from STREAM3
for(Job job:(List<Job>)jobStream2.getJobs()){
    if(job.getName().compareTo("10")==0){
     jobFM2=job;
     zosJobDefinition2=(ZOSJobDefinition) jobFM2.getJobDefinition();
    }
   }
   //Creation of ZosJobInfo for the second job
```

```
    ZosJobInfo jZos2=new ZosJobInfo();

    //Setting the inputArrivalTime
    calendar2=Calendar.getInstance();
    time2=jobFM2.getTimeRestrictions().getStartOffset();
    if(time2!=null && time2>0){
     seconds=(int) (time2 / 1000) % 60 ;
     minutes=(int) ((time2 / (1000*60)) % 60);
     hours=(int) ((time2 / (1000*60*60)) % 24);
     calendar2.set(Calendar.HOUR,hours);
     calendar2.set(Calendar.MINUTE, minutes);
     calendar2.set(Calendar.SECOND, seconds);
     jZos2.setInputArrivalTime(calendar2);
    }

         //Setting the zosJobInfo for the second job
    //with data in zosJobDefinition and job (@timeRestriction)
    workstationN=zosJobDefinition2.getFlowTargetKey().getName();
    jZos2.setJobName(zosJobDefinition2.getJclName());
    jZos2.setTextDescription(jobFM2.getDescription());
    jZos2.setWorkstationName(workstationN);
    jZos2.setDuration(jobFM2.getEstimatedDuration());
    jZos2.setJobNumber(45);
    jZos2.setAutoSubmit(zosJobDefinition2.getAutoSubmit());
    jZos2.setTaskType(zosJobDefinition2.getTaskType());
    jZos2.setTimeDependent(jobFM2.getTimeRestrictions().isTimeDependent());
    jZos2.setCentralizedScript(zosJobDefinition2.getHasCentralizedScript());

    //Getting the workstation -> for the type associated. Header then full workstation.
    queryFilter.setFilter(WorkstationFilters.WORKSTATION_NAME, workstationN);
    try {
     queryResult=model.queryTWSObject(Workstation.class, queryFilter, 1, context);
     workstationHeaderList=(List<WorkstationHeader>) queryResult.getList();
     if(workstationHeaderList.size()>0){
      workstation=(Workstation) model.getTWSObject(Workstation.class,workstationHeaderList.get(0).getKey() ,
             false, context);
     }
    } catch (ConnTransportException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnValidationException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnSecurityException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (RemoteException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    }

    jZos2.setWorkstationType(workstation.getType());

         //Setting the Resource value
 for(ResourceDependency resourceD:(List<ResourceDependency>) jobFM2.getResourceDependencies()){
     String rName=resourceD.getResourceKey().getName();
     if(rName!=null && rName.compareToIgnoreCase("Resource1")==0){
      jZos2.setR1(resourceD.getQuantity());
     }
     if(rName!=null && rName.compareToIgnoreCase("Resource2")==0){
      jZos2.setR2(resourceD.getQuantity());
     }
    if(rName!=null && rName.compareToIgnoreCase("ParallelServers")==0){
     jZos2.setParallelServer(resourceD.getQuantity());
    }
   }

   //Setting predecessor and successor for the jobs.
   //Otherwise we cannot add them to STREAM1

   succ= new Integer(jZos1.getJobNumber());
   successorList.add(succ);
   succ=new Integer(jZos2.getJobNumber());
   successorList.add(succ);
   dependencyToAdd.put(String.valueOf(1),successorList);

   //Adding the zosJobInfo to STREAM1
   jobsToAdd.add(jZos1);
   jobsToAdd.add(jZos2);

   //Example of how to modify a job in STREAM1

   zosJobInfo1.setJobNumber(5);
   zosJobInfo1.setJobName("EXEC1");
   jobsToModify.add(zosJobInfo1);

   zosJobInfo2.setJobNumber(10);
   zosJobInfo2.setJobName("EXEC2");
   jobsToModify.add(zosJobInfo2);

   zosJobInfo4.setJobNumber(15);
   zosJobInfo4.setJobName("EXEC3");
   jobsToModify.add(zosJobInfo4);

   zosJobInfo3.setJobNumber(20);
   zosJobInfo3.setJobName("EXEC4");
   jobsToModify.add(zosJobInfo3);

        //Default JCL variables values (for all jobs)

   variablesToBeSubstituted [0][0] = "VAR1";
   variablesToBeSubstituted [0][1] = "ValVar1ForExec2And3";


   DependenciesResolutionOption resolutionOption = DependenciesResolutionOption.RESOLUTION_ALL;

   //Job level JCL variables values

   variablesMap1 [0][0] = "VAR1";
   variablesMap1 [0][1] = "ValVar1ForExec1";
   variablesMap1 [1][0] = "VAR2";
   variablesMap1 [1][1] = "ValVar2ForExec1";
   variablesMap1 [2][0] = "VAR3";
   variablesMap1 [2][1] = "ValVar3ForExec1";
   jobNum = 5;


   jobVariablesToBeSubstituted.put(jobNum, variablesMap1);


   variablesMap2 [0][0] = "VAR2";
   variablesMap2 [0][1] = "ValVar2ForExec2";
   variablesMap2 [1][0] = "VAR4";
   variablesMap2 [1][1] = "ValVar4ForExec2";
   jobNum = 10;
```

```
    jobVariablesToBeSubstituted.put(jobNum, variablesMap2);

    String [][] variablesMap3 = new String [2][2];
    variablesMap3 [0][0] = "VAR2";
    variablesMap3 [0][1] = "ValVar2ForExec3";
    variablesMap3 [1][0] = "VAR4";
    variablesMap3 [1][1] = "ValVar4ForExec3";
    jobNum = 15;

    jobVariablesToBeSubstituted.put(jobNum, variablesMap3);

    String [][] variablesMap4 = new String [4][2];
    variablesMap4 [0][0] = "VAR3";
    variablesMap4 [0][1] = "ValVar3ForExec4";
    variablesMap4 [1][0] = "VAR4";
    variablesMap4 [1][1] = "ValVar4ForExec4";
    variablesMap4 [2][0] = "VAR5";
    variablesMap4 [2][1] = "ValVar5ForExec4";
    variablesMap4 [3][0] = "VAR6";
    variablesMap4 [3][1] = "ValVar6ForExec4";
    jobNum = 20;

    jobVariablesToBeSubstituted.put(jobNum, variablesMap4);

         //Add STREAM1 in the plan with all the modified and new jobs (zosJobInfo)

    try {
     identifierList = plan.editAddJobStreamInstanceWithVariableSubstitution(
       JSName, startTime, deadlineTime, priority, description, group, owner,
       ownerDescription, variableTable, jobsToDelete, jobsToAdd, jobsToModify,
       dependencyToDelete, dependencyToAdd, variablesToBeSubstituted, authorityGroup,
       holdAll, resolutionOption, jobVariablesToBeSubstituted, context);
     Identifier jsid = identifierList.get(0);

     // get the jobStream (STREAM1) from the plan
     jobStreamInPlan = (JobStreamInPlan) plan.getPlanObject(JobStreamInPlan.class, jsid, context);
    } catch (ConnLockingException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnNotFoundException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnSecurityException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnTransportException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnValidationException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnEngineNotMasterException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (ConnException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    } catch (RemoteException e) {
     //TODO Auto-generated catch block
     e.printStackTrace();
    }

         //Modify the job in JobStreamInPlan to add data missing in ZosJobInfo
    //Like extendedName(job 44) and specialResource(job=45)
for(JobInPlan jobInPlan:(List<JobInPlan>) jobStreamInPlan.getJobs()){
    jobDefinitionInPlan=(ZOSJobDefinitionInPlan) jobInPlan.getJobDefinition();

    if(jobInPlan.getName().compareTo("44")==0){
     //Setting of some extended name and HORC
     jobDefinitionInPlan.setHighestOkReturnCode(0);
     jobDefinitionInPlan.setExtendedName(zosJobDefinition1.getExtendedName());
     try {
      plan.setJobInstance(jobInPlan, context);
     } catch (ConnLockingException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnNotFoundException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnSecurityException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnTransportException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnValidationException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (RemoteException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     }
    }

          if(jobInPlan.getName().compareTo("45")==0){
     jobDefinitionInPlan.setHighestOkReturnCode(0);

     jobInPlanZosAtt=jobInPlan.getZosSpecificAttributes();
     resourceDependencyInPlanList=jobInPlan.getResourceDependencies();
     //Adding a special Resource to the job in a jobStreamInPlan
for(ResourceDependency rD:(List<ResourceDependency>)jobFM2.getResourceDependencies()){
      resourceName=rD.getResourceKey().getName();
      if(resourceName!=null && !resourceName.isEmpty() && resourceName.compareToIgnoreCase("Resource1")!=0 &&
        resourceName.compareToIgnoreCase("Resource2")!=0 &&  resourceName.compareToIgnoreCase("ParallelServers")!=0){
       //Creation and setting of a ResourceDepInPlan
       resourceDependencyInPlan=new ResourceDependencyInPlan();
       resourceDependencyInPlan.setActionOnComplete(rD.getActionOnComplete());
       resourceDependencyInPlan.setAllocationType(rD.getAllocationType());
       resourceDependencyInPlan.setQuantity(rD.getQuantity());
       workstationInPlanKey=new WorkstationInPlanKey();
       workstationInPlanKey.setName(zosJobDefinition2.getFlowTargetKey().getName());
       resourceInPlanKey=new ResourceInPlanKey(resourceName,workstationInPlanKey);
       resourceDependencyInPlan.setId(ResourceInPlanHelper.keyToId(resourceInPlanKey));
       resourceDependencyInPlan.setWorkstationId(WorkstationInPlanHelper.keyToId(workstationInPlanKey));
       resourceDependencyInPlan.setKey(resourceInPlanKey);
       resourceDependencyInPlanList.add(resourceDependencyInPlan);
       jobInPlanZosAtt.setNumberOfSpecialResources(1);
       break;
              }

          }

          try {
      plan.setWholeJobInstance(jobInPlan, false, null, null, null, null, null, jobInPlan.getResourceDependencies(),
                 null, null, null, null, context);
     } catch (ConnLockingException e) {
      //TODO Auto-generated catch block
```

```
        e.printStackTrace();
       } catch (ConnNotFoundException e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
       } catch (ConnSecurityException e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
       } catch (ConnTransportException e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
       } catch (ConnValidationException e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
       } catch (ConnException e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
       } catch (RemoteException e) {
        //TODO Auto-generated catch block
        e.printStackTrace();
       }

      }
     }


     try {
      plan.holdJobStreamInstanceJobs((Identifier)identifierList.get(0), false, context);
     } catch (ConnLockingException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnNotFoundException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnSecurityException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnTransportException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnValidationException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (ConnException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     } catch (RemoteException e) {
      //TODO Auto-generated catch block
      e.printStackTrace();
     }


     return null;
    }
   }); // end doAs
   ;
  }



 }
```

Using the API to work with z/OS® JCL

Describes how to use the API to work with z/OS® JCL.

You normally define JCL jobs in IBM Z Workload Scheduler using the z/OS® Program Interface panels. This section tells you how to create a Java™ interface to maintain the JCL in the appropriate library. You can add, read, modify, and remove a JCL job, and for each such activity you need to be able to connect to the IBM Z Workload Scheduler connector which implements the API.

Defining the connection with the IBM Z Workload Scheduler connector

Describes how to define a connection with the IBM Z Workload Scheduler connector to work with z/OS® JCL.

To define the connection with the IBM Z Workload Scheduler connector use a syntax similar to the following:

```
final ZConnModel model = connection.getModelBean();
```

Add JCL Job

Describes how to add a JCL job using the API.

The parameters to add a JCL job are:

- The job itself
- The job key, which consists of the library name and the JCL job name.

The command returns the ID of the created job.

The following example code creates a JCL job and adds it to a specific job library:

```
JCL jcl = new JCL();
JCLKey jobk1 = new JCLKey("TWSSD.CWSD64.JOBLIB","MYJCL");
jcl.setKey(jobk1);
jcl.getTextLines().add("//"+name+" JOB (876903,D07),'AAAAAAA',MSGLEVEL=(1,1), 00010000");
jcl.getTextLines().add("//     MSGCLASS=A,CLASS=A,NOTIFY=CARDELL              00020000");
jcl.getTextLines().add("//STEP1    EXEC PGM=IEFBR14                           00030001");
jcl.getTextLines().add("//SYSPRINT DD SYSOUT=*                                00060000");

id = model.addTWSObject(jcl,null);
```

Read JCL Job

Describes how to read a JCL job with the API.

The parameters to read a JCL job are:

- The job key, which consists of the library name and the JCL job name
- A boolean value to determine whether to lock the job after reading it

The command returns the JCL job identified by the job key.

The following example code reads a specific JCL job in a specific job library:

```
JCLKey jobk1 = new JCLKey("TWSSD.CWSD64.JOBLIB","MYJCL1");

JCL jobJCL = (JCL)model.getTWSObject(JCL.class, jobk1, false, null);
```

Modify JCL Job

Describes how to modify a JCL job using the API.

First read the JCL job and lock it:

```
JCLKey jobk1 = new JCLKey("TWSSD.CWSD64.JOBLIB","MYJCL1");

JCL jobJCL = (JCL)model.getTWSObject(JCL.class, jobk1, true, null);
```

The parameters to modify a JCL job are:

- The job key, which consists of the library name and the JCL job name
- The modified JCL.

The command returns the ID of the modified job.

The following example code modifies a JCL job that has already been read and locked:

```
jcl.getTextLines().add("//"+name+" JOB (876903,D07),'AAAAAAA',MSGLEVEL=(1,1), 00010000");
jcl.getTextLines().add("//     MSGCLASS=A,CLASS=A,NOTIFY=PIPPO                00020000");

id = model.setTWSObject(jcl, true, true, null);
```

Remove JCL Job

Describes how to remove a JCL job with the API.

First read the JCL job and lock it:

```
JCLKey jobk1 = new JCLKey("TWSSD.CWSD64.JOBLIB","MYJCL1");

JCL jobJCL = (JCL)model.getTWSObject(JCL.class, jobk1, true, null);
```

The parameters to remove a JCL job are:

- The job key, which consists of the library name and the JCL job name

The following example code removes a JCL job that has already been read and locked:

```
model.removeTWSObject(JCL.class, jobk1, null);
```

Reference material

Describes how to access the reference material on the API.

The Integration Workbench help contains all the reference material you require.

To access this material, take the following steps:

1. From the Integration Workbench select **Help > Help Contents**
2. Expand **IBM Workload Scheduler Integration Workbench** and then **Reference**

3. Obtain reference material for any of the following:
    - What information is needed to run the wizards that create API projects, either from scratch or from an example
    - Information about the libraries of object and runtime jars
    - Description of the XML schemas
    - A link to information about the IBM Event Integration Facility
    - Full reference for every Java™ class and method

Further information

Gives links to further information about using Java™ APIs.

Redbooks®

To find out more about how to program this type of API, see the following IBM® Redbooks®:

IBM® Redbooks®: EJB 2.0 Development with WebSphere® Studio Application Developer, SG24-6819

This IBM® Redbook provides detailed information on how to effectively use WebSphere® Studio Application Developer for the development of applications based on the Enterprise JavaBeans™ (EJB) architecture, and deployment of such applications to a WebSphere® Application Server.

To access this publication, follow this link: http://www.redbooks.ibm.com/abstracts/sg246819.html.

IBM® Redbooks®: Programming J2EE APIs with WebSphere® Advanced, SG24-6124

This IBM® Redbook has examples of programming the new J2EE APIs using VisualAge® for Java™ and deployment on WebSphere® Advanced.

To access this publication, follow this link: http://www.redbooks.ibm.com/abstracts/sg246819.html.

Notices

This document provides information about copyright, trademarks, terms and conditions for product documentation.

Trademarks

Terms and conditions for product documentation

Index