IBM Tivoli Workload Automation

Developer's Guide: Driving Tivoli Workload Automation

Version 9 Release 2

SC23-9608-03

IBM Tivoli Workload Automation

Developer's Guide: Driving Tivoli Workload Automation

Version 9 Release 2

SC23-9608-03

Note
Before using this information and the product it supports, read the information in "Notices" on page 79.

This edition applies to version 9, release 2, modification level 0 of Tivoli Workload Scheduler (program number 5698-WSH) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

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®.

IBM Tivoli® Workload Automation: Developer's Guide: Driving Tivoli Workload Automation introduces you to the application programming interfaces available to drive Tivoli Workload Automation products from your own applications. It also describes how to use the Integration Workbench provided with the product to develop and implement these application programming interfaces.

For example, with the information in this publication, and the associated help in the Integration Workbench, you can have your application generate one or more jobs and a job stream, submit the job stream to the plan, monitor its progress, and delete the created objects on successful completion; all without manually running **composer**, **conman** or the Dynamic Workload Console.

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 *Tivoli Workload Automation: Overview*, section *Summary of enhancements*.

For information about the APARs that this release addresses, see the Tivoli Workload Scheduler Release Notes at http://www-01.ibm.com/support/docview.wss?rs=672&uid=swg27041032 and the Dynamic Workload Console Release Notes at http://www-01.ibm.com/support/docview.wss?rs=672&uid=swg27041033.

Who should read this publication

Describes the type of user who should read the documentation.

This publication provides information about how to create alternative interfaces to Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS than those provided with the product.

The reader of this book should be an *application programmer* expert in Java™ or web services (as appropriate), who has a reasonable understanding of the Tivoli Workload Automation infrastructure and its inter-component interactions, or the manager of such a person, who wants to better understand what you can achieve using the application programming interfaces.

The publication assumes that the application programmer is experienced at creating these types of interface. It also assumes that any product knowledge required to program the API or the web services interface is obtained from the product documentation. This publication does not attempt to explain any of the Tivoli Workload Automation concepts, procedures, and practices to which it refers.

This book also contains information useful to the *IT administrator* and the *Tivoli Workload Automation IT administrator*, for planning purposes.

Publications

The Tivoli Workload Automation product is supported by a set of publications.

For a list of publications in the Tivoli Workload Automation product library, see *Publications* under *Reference* in the product documentation.

For a list of terms used in the Tivoli Workload Automation product, see *Glossary* under *Reference* in the product documentation.

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 with respect to the Dynamic Workload Console, see the Accessibility Appendix in the *IBM Tivoli Workload Scheduler User's Guide and Reference*.

Tivoli technical training

Tivoli provides technical training.

For Tivoli technical training information, refer to the following IBM Tivoli Education website:

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 on support information in *Tivoli Workload Scheduler: Troubleshooting Guide*.

Chapter 1. Introduction to driving Tivoli Workload Automation

Provides an overview of the entire publication.

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

- Use the Java 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 Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS:
  - 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
- Use the web services interface to create your own web client application to perform a subset of Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS functions to manage jobs and job streams in the plan. Neither database actions nor other plan actions can be implemented and invoked using this interface.

You can use the Tivoli Workload Automation Software Development Kit's Integration Workbench provided with the product to develop and implement these application programming interfaces.

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

- Chapter 2, "Integration Workbench," on page 3
- Chapter 3, "Driving Tivoli Workload Automation with the Java API," on page 5
- Chapter 4, "Driving Tivoli Workload Automation with the Web services interface," on page 31

Chapter 2. Integration Workbench

Describes the Integration Workbench of the Software Development Kit.

IBM Tivoli Workload Automation: Software Development Kit comes with an Integration Workbench which you can use to work with the Java application programming interface and the web services interface to develop your own applications.

This section tells you how to install and use the Integration Workbench help. The help contains detailed information on the tasks you can perform with the Integration Workbench, and the detailed reference information on the methods and classes available:

Installing the Integration Workbench

Gives an overview of the Integration Workbench installation.

The Integration Workbench (part of the Software Development Kit - SDK) runs under Eclipse. The installation, which is fully described in the *Tivoli Workload Scheduler: Planning and Installation Guide*, *SC32-1273*, gives you the opportunity to install the Integration Workbench and a bundled, supported version of Eclipse in one action, or to install the Integration Workbench as an *Eclipse site* using an existing supported version of Eclipse available in your network.

In both cases, at the end of the installation, on the panel where you click **Finish**, there is an option to display the file `readmefirst.html`, which contains information about the workbench, and how to run it. This information is also given here, in "Using the Integration Workbench help."

For more information about Eclipse, go to http://www.eclipse.org/.

Using the Integration Workbench help

Describes how to access the Integration Workbench help facility for the Tivoli Workload Scheduler API and plug-in projects.

To use the Integration Workbench help, do the following:

1. Launch the workbench, as follows:

   Integration Workbench installed with Eclipse

   UNIX Launch the following file: `<TWS_home>/TWS/IntegrationWorkbench/eclipse/eclipse`

   Windows
   Go to **Start → Tivoli Workload Scheduler → Integration Workbench**

   Integration Workbench installed as Eclipse site
   Open your version of Eclipse, as you normally do.
2. Select the location to save your Eclipse workspace. Eclipse requests this every time you run it or the workbench within it, unless you check the option to save a particular location as the default.

3. When the Eclipse window opens, select **Help → Help Contents**
4. Expand **IBM Tivoli Workload Scheduler Integration Workbench**
5. The options displayed provide a variety of information about the Tivoli Workload Scheduler Integration Workbench. For example, to see details of all the classes and methods employed in the API, expand **Reference** and select **API reference**.

Note: The above information can also be read by opening the following document in a Web browser: <*TWS_home*>/TWS/IntegrationWorkbench/readmefirst.html

Chapter 3. Driving Tivoli Workload Automation with the Java API

This chapter describes the J2EE Application Programming Interface (API), which uses Enterprise Java Beans to drive Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS.

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 **Tivoli 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.

Tivoli 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 Tivoli 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 Tivoli Workload Scheduler instance:

"Java source tree (src)"
Separate directories for the source and class files.

"Java libraries" on page 7
A JRE System library and separate libraries are available for the Tivoli Workload Scheduler object and runtime jars.

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

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

One or more Java compilation units
A compilation unit includes a class that implements the Java interface for the connection to Tivoli 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 7
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 7
You run a wizard, supplying information about what sort of project you want to create.

"Creating a project from an API example" on page 8
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 Tivoli Workload Scheduler Java API reference to figure out 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.

Tivoli 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 Tivoli Workload Scheduler event processor runs using IBM JDK version 1.5.

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

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

This library contains all the Tivoli 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

Other project folders

Describes the other project folders.

config Use this folder to store additional configuration files (such as property files) that the Tivoli 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 Tivoli Workload Scheduler secure login.

Java build path

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"
Describes the other project folders.

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 **Tivoli Workload Scheduler Integration Workbench** and then **Tasks**
3. Expand **Tivoli Workload Scheduler API projects**
4. The steps required to create the project are listed. Read them to understand what to do.
5. 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 **Tivoli Workload Scheduler Integration Workbench** and then **Tasks**
3. Expand **Tivoli Workload Scheduler API projects**
4. Select **Creating a TWS 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®.

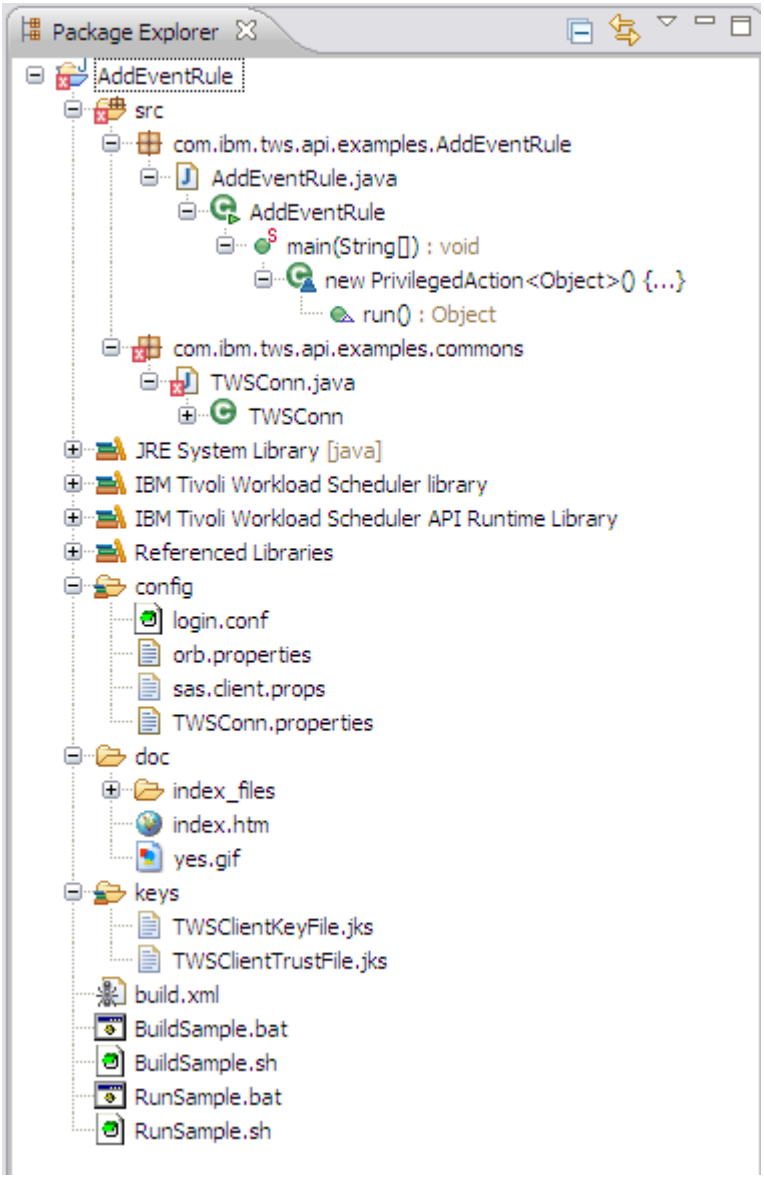
RerunJobInPlan
Submit a job stream instance into the current plan rerun.

SubmitJobInPlan
Submit a job stream instance into the current plan.

Example Tivoli Workload Scheduler API project

Provides an example of a Tivoli Workload Scheduler API project.

The following figure shows the typical structure of a new Tivoli Workload Scheduler java API project:



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).

The Java Build Path of the plug-in project is described in Java build path.

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

The ANT `build.xml` file is described in build.xml.

- **Java source tree (src)**
- **Java build path**
- **Other project folders**
- **build.xml**

Connecting to the products

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

To connect to the products, you need to set the Connection Parameters to connect either to the Tivoli Workload Scheduler master domain manager (where there is a Connector installed) or to the Tivoli Workload Scheduler for z/OS 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 Tivoli 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 Tivoli 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 Tivoli Workload Scheduler for z/OS 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 Tivoli Workload Scheduler

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

The following examples help you to understand how the beans are used. The examples are annotated with explanatory comments. In the javadoc reference (see "API detailed specification" on page 5), 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.ibm.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...
}
```

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 Tivoli Workload Scheduler for z/OS

Provides an overview of the examples available of using the Java API for Tivoli Workload Scheduler for z/OS.

The following examples help you to understand how the beans are used in the Tivoli Workload Scheduler for z/OS 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 get 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 Tivoli Workload Scheduler for z/OS 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 Tivoli Workload Scheduler for z/OS connector which implements the API.

Defining the connection with the Tivoli Workload Scheduler for z/OS connector

Describes how to define a connection with the Tivoli Workload Scheduler for z/OS connector to work with z/OS JCL.

To define the connection with the Tivoli Workload Scheduler for z/OS 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 **Tivoli 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 Tivoli 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.

Chapter 4. Driving Tivoli Workload Automation with the Web services interface

Use the Web Services interface to perform a subset of Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS functions to manage jobs and job streams in the plan, from your own web client application. Neither database actions nor other plan actions can be implemented and invoked using this interface.

Whenever you install a Tivoli Workload Scheduler component that includes the WebSphere Application Server, the following WSDL files are automatically installed:

SchedulingFactory.wsdl
Use these services to submit jobs or job streams to the plan and identify jobs and job streams in the plan that match defined criteria.

JobService.wsdl
Use these services to view the properties of jobs in the plan, view the job output, set selected properties, release all dependencies, cancel, and kill jobs in the plan.

JobStreamService.wsdl
Use these services to view the properties of job streams in the plan, view the job stream output, set selected properties, release all dependencies, and cancel job streams in the plan.

They are installed in the following path:

`<WAS_profile_path>/installedApps/DefaultNode/<component_path>`

where the default value of <*WAS_profile_path*> is `<TWA_home>/WAS/TWSprofile` and <*component_path*> depends on which components are installed:

z/OS connector
`ZConnector.ear/PlanServicesWeb.war/WEB-INF/wsdl`

Other Tivoli Workload Scheduler components
`TWSEngineModel.ear/PlanServicesWeb.war/WEB-INF/wsdl`

If you have both the z/OS Connector and a Tivoli Workload Scheduler component installed, the files will be present twice (but have the same content).

Open these WSDL files with a Web Services development tool. They provide you with:

- The server part interfacing the master domain manager to perform the supported subset of scheduling operations against jobs and job streams in production.
- A means of creating your own client interface from where service requesters can request to perform a subset of operations from any system in your environment

The same set of services is delivered for both Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS environments. However, some can be used in only one environment, and many of them have different parameters in the different environments, so they are documented separately for the two environments.

The Web Services interface is described in the following topics:

Web services for Tivoli Workload Scheduler

Provides an overview, and links to detailed descriptions of, the web services provided for Tivoli Workload Scheduler.

Full details are as follows (click the links in these tables to go to the description of the service):

SchedulingFactory.wsdl

Table 1. Available services in the SchedulingFactory Web services interface for Tivoli Workload Scheduler

Service name	Actions performed
submitJob	Submits jobs defined in the database into the production plan (distributed environment only).
submitAdHocJob	Submits ad-hoc jobs into the production plan (distributed environment only).
queryJobs	Returns all job instances matching the filtering criteria.
submitJobStream	Submits job streams into the production plan.
queryJobStreams	Returns all job stream instances matching the filtering criteria.

JobService.wsdl

Table 2. Available services in the JobService Web services interface for Tivoli Workload Scheduler

Service name	Actions performed
getProperties	Displays job properties.
setProperties	Sets specified properties for a job instance.
getOutput	Gets a job instance output.
kill	Kills a job instance.
cancel	Cancels a job instance.
releaseAllDependencies	Releases all dependencies for a job instance.

JobstreamService.wsdl

Table 3. Available services in the JobStreamService Web services interface for Tivoli Workload Scheduler

Service name	Actions performed
getProperties	Displays job stream properties.
setProperties	Sets specified properties for a job stream instance.
getJobsList	Lists all jobs contained in a job stream.
cancel	Cancels a job stream instance.
releaseAllDependencies	Releases all dependencies for a job stream instance.

Web services for Tivoli Workload Scheduler for z/OS

Provides an overview, and links to detailed descriptions of, the web services provided for Tivoli Workload Scheduler for z/OS.

Full details are as follows (click the links in these tables to go to the description of the service):

SchedulingFactory.wsdl

Table 4. Available services in the SchedulingFactory Web services interface for Tivoli Workload Scheduler for z/OS

Service name	Actions performed
queryJobs (z/OS)	Returns all job instances matching the filtering criteria.
submitJobStream (z/OS)	Submits job streams into the production plan.
submitJobStreamWithVarSub (z/OS)	Submits job streams with variable substitution into the production plan.
editSubmitJobStreamWithVarSub (z/OS)	Edits and submits job streams with variable substitution into the production plan.
queryJobStreams (z/OS)	Returns all job stream instances matching the filtering criteria.

JobService.wsdl

Table 5. Available services in the JobService Web services interface for Tivoli Workload Scheduler for z/OS

Service name	Actions performed
getProperties (z/OS)	Displays job properties.
setProperties (z/OS)	Sets specified properties for a job instance.
getOutput (z/OS)	Gets a job instance output.
cancel (z/OS)	Cancels a job instance.

JobstreamService.wsdl

Table 6. Available services in the JobStreamService Web services interface for Tivoli Workload Scheduler for z/OS

Service name	Actions performed
getProperties (z/OS)	Displays job stream properties.
setProperties (z/OS)	Sets specified properties for a job stream instance.
getJobsList (z/OS)	Lists all jobs contained in a job stream.
cancel (z/OS)	Cancels a job stream instance.

Web services management

Describes how to manage your Web service environment.

This section describes how to manage the web services from your application. It has the following topics:

• "Accessing the services" on page 34

• "Managing errors" on page 35

Accessing the services

Access the web services by invoking a proxy.

The web services are accessed by invoking a proxy which contains the path that your application will use to access the web services wsdl files at the following location:

Distributed (not z/OS) environment

`http(s)://<localhost>:port_number/PlanServicesWeb/services/<service_name>`

z/OS environment

`http(s)://<localhost>:port_number/zPlanServicesWeb/services/<service_name>`

where:

localhost
The hostname of the master domain manager

port_number
The port numbers for http or https are defined in your WebSphere Application Server installation in:

Tivoli Workload Scheduler
`<WAS_profile_path>/config/cells/ TWSNodeCell/nodes/TWSNode/servers/server1/server.xml`

where the default value of <*WAS_profile_path*> is `<TWA_home>/WAS/TWSprofile`

Dynamic Workload Console
`<JazzSM_profile_dir>/config/cells/JazzSMNode01Cell/nodes/JazzSMNode01/servers/server1/`

where, the default value of < *JazzSM_profile_dir*> is:

On Windows operating systems
C:\Program Files\IBM\JazzSM\profile

On UNIX operating systems
/opt/IBM/JazzSM/profile

under:

- `WC_defaulthost` for http
- `WC_defaulthost_secure` for https

service_name
The name of the service you invoke: `SchedulingFactory`, `JobService`, or `JobStreamService`.

An example of accessing the web services in the z/OS environment is as follows:

```
SchedulingFactoryProxy proxy = new SchedulingFactoryProxy("http://111.222.333.444:
                               31126/zPlanServicesWeb/services/SchedulingFactory");

String[] jstreams = proxy.submitJobStreamWithVarSub(
                    ENGINE_NAME,
                    JOB_STREAM_KEY,
                    schedTime,
                    deadlineTime,
                    null,
```

```
null,
null,
null,
null,
null,
null,
variablesToBeSubstituted);
```

Identifying the correct master domain manager

Describes how to identify the correct master domain manager when invoking Web services.

To ensure that your web service is performed on the correct master domain manager, do the following:

Distributed (not z/OS) environment

1. When you create the access proxy to the wsdl where the service you require is located, access the wsdl *on the same master domain manager* (engine) where you want it to be performed.
2. When invoking the service, set the `engineName` element in the service method to null.

The service is performed on the master domain manager which is local to the wsdl you access.

z/OS environment

1. When you create the access proxy to the wsdl where the service you require is located, access the wsdl *on the system where the z/OS connector is installed*
2. When invoking the service, set the `engineName` element in the service method to the z/OS engine name where you want to perform the service.

The service is performed on the master domain manager identified by the z/OS engine name.

Managing errors

Describes how to manage errors when using Web services.

This section indicates the errors which could occur, depending on the individual service being used. In normal circumstances, each error is accompanied by another error message explaining the problem in more detail. The errors are as follows:

InvalidArguments
The syntax of the service request is not correct. Correct the syntax and retry the operation.

Locking
An error occurred when the service tried to lock an object before using it. This is normally caused because another user has an object involved in the operation locked for editing. It is normally sufficient to wait and retry the action. A repeated failure of this type might indicate a more serious problem that should be brought to the attention of the Tivoli Workload Scheduler administrator.

ObjectNotFound
The object identified in the request could not be found and thus cannot be submitted. Before submitting a request which modifies or deletes an existing object you might decide to query the object first and only perform

the modification or deletion on the object or objects returned by the query. Note that it is possible that an object is deleted by another user between retrieving a list of objects with a query and implementing a modification or deletion.

EngineNotMaster
This message is only returned by the z/OS Connector and indicates that the engine identified in the request is not the master domain manager. This might represent an error in the supply of the engine name.

Transport
A communications error occurred. See the accompanying message to determine how to resolve the problem.

OperationFailed
The operation failed. See the accompanying message to determine the reason and how to resolve the problem.

Security
The operation could not be completed because the user running the request does not have the appropriate security access to the one or more of the objects which were the subject of the request. Either change the security access rights of the user in the Security file or submit the operations as a user with the appropriate rights

SchedulingFactory Web services

Describes the Web services available in the SchedulingFactory.wsdl file.

This section describes the web services you can use from the `SchedulingFactory.wsdl` file. Examples are given of the use of some of the web services in the list.

Web services are described separately for Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS.

SchedulingFactory web services for Tivoli Workload Scheduler

Describes the Web services available in SchedulingFactory.wsdl for Tivoli Workload Scheduler.

The following Web services can be used to interface with the plan in Tivoli Workload Scheduler.

submitJob

Describes the submitJob Web service for Tivoli Workload Scheduler.

Description
Use this service to submit a job in the Tivoli Workload Scheduler plan.

Input parameters

engineName
Not used; set to null.

jobKey
The key identifying the job in the Tivoli Workload Scheduler database: *workstationName#jobName*.

alias The alias name of the job. Also accepts `null` value.

Output

The response `submitJobResponse` returns a string `submitJobReturn` containing the identifier of the job that was actually submitted in the plan.

submitAdHocJob

Describes the submitAdHocJob Web service for Tivoli Workload Scheduler.

Description

Use this service to submit a job in the Tivoli Workload Scheduler plan. The job is not defined in the database.

Input parameters

engineName

Not used; set to null.

job A `jobToSubmit` statement which describes the job. The `jobToSubmit` statement is defined and documented in the `TWS-Types.xsd` file, and in "Defining the ad hoc job."

Output

The response `submitAdHocJobResponse` returns a string `submitAdHocJobReturn` containing the identifier of the job that was actually submitted in the plan.

Defining the ad hoc job:

To submit an ad hoc job, you must supply a `jobToSubmit` statement that describes the job. The statement has the following format:

jobName

The name of the job. Wildcard characters are permitted, in which case, all qualifying jobs are submitted.

workstationName

The name of the workstation where the job is to run. If you leave blank, the workstation where the command is run becomes the default.

userLogin

The login of the user who runs the job.

taskType

The task type. It can be one of the following:

- UNIX
- Windows
- Broker
- Other

taskString

A string containing the parameters that control the execution of the job.

priority

The execution priority for the job. Must be one of the following:

- A number ranging from 0 to 99
- hi
- go

command

Specifies if the job is a command. Can be `yes` or `no`.

monitored

Specifies if the job is to be monitored by event rules. Can be `yes` or `no`.

requiredConfirmation
Specifies if the completion (failed or successful) of this job requires user confirmation. Can be `yes` or `no`.

recoveryJobName
The name of a recovery job that is to run if this job ends abnormally.

recoveryJobWorkstationName
The name of the workstation where the recovery job is to run. The default is `workstationName`.

recoveryOption
Recovery options for the job. Values can be:
- Stop
- Continue
- Rerun

The default is `stop` with no recovery job and no recovery prompt.

recoveryPromptText
Specifies the text of a recovery prompt, enclosed in double quotes, to be displayed if the job ends abnormally. The rules are:
- The text can contain up to 64 characters
- If the text begins with a colon (:), the prompt is displayed, but no reply is required to continue processing
- If the text begins with an exclamation mark (!), the prompt is displayed, but it is not recorded in the log file

returnCodeMapping
An expression which defines a job final status (successful or failed) based on a condition on the return code of the execution of the program or script of the job.

The return code can be provided also to the recovery job that is associated with it in the job definition. This causes the recovery job to perform different processing based on the return code.

estimatedDuration
The estimated duration of a job. This value is based on the statistics collected from previous runs of the job. If the job has never run before, a default value of one minute is used.

This parameter is relevant if the job is a predecessor of a critical job.

startTime
The date and time at which the job must start.

latestStartTime
The latest date and time at which the job can start.

latestStartAction
The action to take on the job if the latest start time has elapsed. Values can be:
- Cancel
- Continue
- Suppress

deadlineTime
The date and time within which the job must have completed. After this time the job is considered late.

repeatInterval
The repetition rate for the job in hours and minutes, in the *hhmm* format. The job is launched again every time the end of the interval is reached. The interval can be longer than 24 hours.

For more information see the description of the `submit job (sbj)` command in the Tivoli Workload Scheduler *Tivoli Workload Scheduler: User's Guide and Reference*.

queryJobs

Describes the queryJobs Web service for Tivoli Workload Scheduler.

Description
Use this service to run queries on Tivoli Workload Scheduler job instances.

Input parameters

engineName
Not used; set to null.

filter A list of filtering criteria represented by an array of `filterCriteria` statements. The `filterCriteria` statements are defined and documented in the `TWS-Types.xsd` file and in "Defining the filtering criteria to query jobs."

Output
The response `queryJobsResponse` returns an array `queryJobsReturn` of the identifiers of the jobs that matched the query.

Defining the filtering criteria to query jobs:
Each `filterCriteria` statement has the following form:

details
Not used; set to null.

value An array of values for the dataType.

minimum
A single value representing the minimum of a range.

maximum
A single value representing the maximum of a range.

dataType
A string identifying the field for which you have supplied a value, several values, or a range. The fields are defined and documented in the `SchedulingFactory.wsdl` file and also as follows:

JOB_ID
The job identifier.

JOB_NAME
The name of the job.

JOB_STREAM_NAME
The name of the job stream in which the job ran.

WORKSTATION_NAME
The name of the workstation that ran the job.

STATUS_LIST
Can be one of the following:
- BLOCKED
- CANCELLED
- COMPLETED

- ERROR
- HELD
- READY
- STARTED
- WAITING
- UNDECIDED

INTERNAL_STATUS_LIST
Can be one of the following:
- ABEND
- ABEND_P
- ADDING
- CANCEL
- CANT_STREAM
- CNPEND
- END_P
- ERROR_STAT
- EXEC
- EXEC_BM
- EXTRN
- FENCE
- HOLD
- MPE_INTRO
- MPE_INTRO_BM
- MPE_SCHED
- MPE_SUSP
- MPE_WAIT
- MPE_WAITD
- READY
- RESTART_JOB
- SUCC
- SUCC_P
- UNKNOWN
- USER_HELD
- USER_STREAM

PRIORITY
The execution priority with which the job ran. Can be one (or a range of values) of the following:
- A number ranging from 0 to 99
- hi
- go

PRIORITY_RANGE
Used if you want to query jobs within a range of priority values. In this case, `minimum` and `maximum` must also be supplied.

CONFIRMED
Determines if only confirmed jobs are to be selected. Can be `true` or `false`.

RERUN
Determines if only rerun jobs are to be selected. Can be `true` or `false`.

START_TIME
The start time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

START_TIME_RANGE
Used if you want to query jobs within a range of start times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

UNTIL_TIME
The until time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

UNTIL_TIME_RANGE
Used if you want to query jobs within a range of until times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

FINISH_TIME
The finish time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

FINISH_TIME_RANGE
Used if you want to query jobs within a range of finish times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

RECOVERY_OPTION_LIST
Can be one of the following:
- STOP
- CONTINUE
- RERUN

MONITORED_JOB
Not used; set to null.

TASK The name of the task.

USER_LOGIN
The login of the user who ran the job.

ERROR_CODE
Not used; set to null.

submitJobStream

Describes the submitJobStream Web service for Tivoli Workload Scheduler.

Description
Use this service to submit a job stream (application) in the Tivoli Workload Scheduler plan.

Input parameters

engineName
Not used; set to null.

jsKey The key identifying the job stream in the scheduler database: *workstationName#jobStreamName*

schedTime
The scheduled submission time for the job stream.

alias The alias name of the job stream.

Output
The response `submitJobStreamResponse` returns an array `submitJobStreamReturn` of the identifiers of the job streams that were actually submitted in the plan.

queryJobStreams

Describes the queryJobStreams Web service for Tivoli Workload Scheduler.

Description

Use this service to run queries on Tivoli Workload Scheduler job stream instances.

Input parameters

engineName
Not used; set to null.

filter A list of filtering criteria represented by an array of `filterCriteria` statements. The `filterCriteria` statements are defined and documented in the `TWS-Types.xsd` file and in "Defining the filtering criteria to query job streams."

Output

The response `queryJobStreamsResponse` returns an array `queryJobStreamsReturn` of the identifiers of the job streams that matched the query.

Defining the filtering criteria to query job streams:

Each `filterCriteria` statement has the following form:

details
Not used; set to null.

value An array of values for the dataType.

minimum
A single value representing the minimum of a range.

maximum
A single value representing the maximum of a range.

dataType
A string identifying the field for which you have supplied a value, several values, or a range. The fields are defined and documented in the `SchedulingFactory.wsdl` file and also as follows:

JOB_STREAM_ID
The job stream identifier.

JOB_STREAM_NAME
The name of the job stream.

WORKSTATION_NAME
The name of the workstation that ran the job stream.

STATUS_LIST
Can be one of the following:
- BLOCKED
- CANCELLED
- COMPLETED
- ERROR
- HELD
- READY
- STARTED
- WAITING
- UNDECIDED

INTERNAL_STATUS_LIST

Can be one of the following:

- ABEND
- ADDING
- CANCEL
- CNPEND
- EXEC
- HOLD
- READY
- SUCC
- SUSP
- USER_HELD

LIMIT

The limit value with which the job stream ran. Can be one of the following:

- A number ranging from 0 to 1024
- hi
- go

LIMIT_RANGE

Used if you want to query job streams within a range of limit values. In this case, `minimum` and `maximum` must also be supplied.

PRIORITY

The execution priority with which the job stream ran. Can be one of the following:

- A number ranging from 0 to 99
- hi
- go

PRIORITY_RANGE

Used if you want to query job streams within a range of priority values. In this case, `minimum` and `maximum` must also be supplied.

CARRIED_FORWARD

Determines if only carried-forward job streams are to be selected. Can be `true` or `false`.

CARRIED_FORWARD

Determines if only carry-forward job streams are to be selected. Can be `true` or `false`.

START_TIME

The start time defined for the job stream in `YYYYMMDD HHMM` format or milliseconds.

START_TIME_RANGE

Used if you want to query job streams within a range of start times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

UNTIL_TIME

The until time defined for the job stream in `YYYYMMDD HHMM` format or milliseconds.

UNTIL_TIME_RANGE

Used if you want to query job streams within a range of until times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

DEADLINE_TIME
The deadline time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

DEADLINE_TIME_RANGE
Used if you want to query job streams within a range of deadline times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

SchedulingFactory web services for Tivoli Workload Scheduler for z/OS

Describes the Web services available in SchedulingFactory.wsdl for Tivoli Workload Scheduler for z/OS.

The following Web services can be used to interface with the plan in Tivoli Workload Scheduler for z/OS.

queryJobs (z/OS)

Describes the queryJobs Web service for Tivoli Workload Scheduler for z/OS.

Description
Use this service to run queries on Tivoli Workload Scheduler for z/OS job instances.

Input parameters

engineName
The Tivoli Workload Scheduler for z/OS engine name.

filter A list of filtering criteria represented by an array of `filterCriteria` statements. The `filterCriteria` statements are defined and documented in the `TWS-Types.xsd` file and in "Defining the filtering criteria to query z/OS jobs" on page 45.

Output
The response `queryJobsResponse` returns an array `queryJobsReturn` of the identifiers of the jobs that matched the query. The fields in the array are:

occurrenceToken
The identifier of the occurrence that includes the operation.

extendedStatus
The extended status code assigned to the operation.

errorCode
The error code that terminated the operation.

operationCommand
Can be one of the following: BD = Bind shadow; job EX = Execute operation; KJ = Kill operation; KR = Kill recovery job; MH = Hold operation; MR = Release operation; NP = NOP operation; PN = Prompt reply no; PY = Prompt reply yes; UN = Un-NOP operation.

authorityGroup
The name of the authority group that the operation belongs to.

cleanUpStatus
Can be one of the following: Blank=none C=Completed E=Ended in error I=Initiated O=Avail opinfo R=Request opinfo S=Started W=Waiting opinfo

latestOutPassed
The operation exceeded its latest starting time and is late. Can be `true` or `false`.

latestOut
The latest possible time, calculated at plan-creation-time, that the operation can start to meet the deadline time (HHMM).

actualArrival
The actual run time of the operation (HHMM).

actualEnd
The time the operation is reported as complete or ended-in-error (HHMM).

Defining the filtering criteria to query z/OS jobs:
Each `filterCriteria` statement has the following form:

details
Not used. Set to null.

value An array of values for the dataType.

minimum
A single value representing the minimum of a range.

maximum
A single value representing the maximum of a range.

dataType
A string identifying the field for which you have supplied a value, several values, or a range. The fields are defined and documented in the `SchedulingFactory.wsdl` file and also as follows:

JOB_ID
Not used; set to null.

JOB_NAME
The name of the job. Must be numeric.

JOB_STREAM_NAME
The name of the job stream in which the job ran.

WORKSTATION_NAME
The name of the workstation that ran the job.

STATUS_LIST
Can be one of the following:
- BLOCKED
- CANCELLED
- COMPLETED
- ERROR
- HELD
- READY
- STARTED
- WAITING
- UNDECIDED

INTERNAL_STATUS_LIST
Can be one of the following:
- COMPLETE
- DELETED
- ERROR

- WAITING
- STARTED
- UNDECIDED
- READY
- PENDINGPRED
- WAITINGFORINPUT
- INTERRUPTED
- NOREPORTINGPRED

PRIORITY
The execution priority with which the job ran. Can be one (or a range of values) of the following:
- A number ranging from 0 to 99
- hi
- go

PRIORITY_RANGE
Used if you want to query jobs within a range of priority values. In this case, `minimum` and `maximum` must also be supplied.

CONFIRMED
Not used; set to null.

RERUN
Not used; set to null.

START_TIME
The start time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

START_TIME_RANGE
Used if you want to query jobs within a range of start times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

UNTIL_TIME
Not used; set to null.

UNTIL_TIME_RANGE
Not used; set to null.

FINISH_TIME
The finish time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

FINISH_TIME_RANGE
Used if you want to query jobs within a range of finish times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

RECOVERY_OPTION_LIST
Not used; set to null.

MONITORED_JOB
Determines if only monitored jobs are to be selected. Can be `true` or `false`.

TASK The name of the task.

USER_LOGIN
Not used; set to null.

ERROR_CODE
Job error code.

submitJobStream (z/OS)

Describes the submitJobStream Web service for Tivoli Workload Scheduler for z/OS.

Description

Use this service to submit a job stream (application) in the Tivoli Workload Scheduler for z/OS plan.

Input parameters

engineName
The Tivoli Workload Scheduler for z/OS engine name.

jsKey The key identifying the job stream in the scheduler database: *jobStreamName*

schedTime
Not used; set to null.

alias Not used; set to null.

Output

The response `submitJobStreamResponse` returns an array `submitJobStreamReturn` of the identifiers of the job streams that were actually submitted in the plan. Note that the plan object identifiers returned contain the `'\0'` character. This is a not valid character and it must be replaced by a blank.

submitJobStreamWithVarSub

Describes the submitJobStreamWithVarSub Web service for Tivoli Workload Scheduler for z/OS.

Description

Use this service to submit a job stream (application) in the Tivoli Workload Scheduler for z/OS plan and to assign values to any variables present in the included jobs. The variable substitution is performed only on promptable variables that are substituted at job set up phase. The other variables are substituted at submission time.

Note: The job stream must include no setup jobs. If it does, the following message is displayed:

```
EQQM229E
   JCL BROWSE/EDIT CAN ONLY BE SELECTED FOR PROCESSOR WORKSTATIONS.
```

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS engine.

jsKey The key identifying the job stream in the Tivoli Workload Scheduler for z/OS database. The valid format is *jobStreamName*.

schedTime
The input arrival time for the job stream.

deadlineTime
The completion dead line time for the job stream.

priority
The job stream priority value.

description
The job stream description. Maximum 24 characters.

groupName
The job stream group name. Maximum 16 characters.

ownerName
The job stream owner name. Maximum 16 characters.

ownerDescription
The job stream owner description. Maximum 24 characters.

authorityGroup
The job stream authority group. Maximum 8 characters.

dependenciesResolution
Specifies which type of dependencies are to be resolved. The value can be:

Y Resolve both predecessor and successor dependencies.

N Ignore all dependencies.

P Resolve only predecessor dependencies.

S Resolve only successor dependencies.

If no value is entered, the default is **N**.

variableTable
The name of the variable table associated with the job stream. Maximum 16 characters.

variableToBeSubstituted
An array of `Property` statements specifying the variables that will be substituted in the job stream. The `Property` statement is defined and documented in the `TWS-Types.xsd` file, and has the following form:

value An array of values for the dataType.

dataType
A string identifying the variable for which you are supplying the value or values, and which can be found in the defined variable table.

Output
The service returns an array of the identifiers of the job streams that were actually submitted in the plan. Note that the plan object identifiers returned contain the `'\0'` character. This is not a valid character and must be replaced by a blank.

Example:
The following is a coding example of `submitJobStreamWithVarSub`, where the variables to be substituted are called SURNAME and NAME and are in the default variable table for the job stream:

```
long now = System.currentTimeMillis();

 /**
  * inputArrivalTime
  */
 Calendar schedTime = new GregorianCalendar();
 Date schedDate = new Date(now);
 schedTime.setTime(schedDate);
```

```
/**
 * deadlineTime
 */
Calendar deadlineTime = new GregorianCalendar();
Date deadlineDate = new Date(now + HOUR_8);
deadlineTime.setTime(deadlineDate);

/**
 * variablesToBeSubstituted
 */
Property prop1 = new Property();
prop1.setDataType("SURNAME");
String[] valueSurname = {"Robinson"};
prop1.setValue(valueSurname);

Property prop2 = new Property();
prop2.setDataType("NAME");
String[] valueName = {"John"};
prop2.setValue(valueName);

Property[] variablesToBeSubstituted = new Property[2];
variablesToBeSubstituted[0] = prop1;
variablesToBeSubstituted[1] = prop2;

SchedulingFactoryProxy proxy = new SchedulingFactoryProxy("http://111.222.333.444:
                               55555/zPlanServicesWeb/services/SchedulingFactory");

String[] jstreams = proxy.submitJobStreamWithVarSub(
                    ENGINE_NAME,
                    JOB_STREAM_KEY,
                    schedTime,
                    deadlineTime,
                    null,
                    null,
                    null,
                    null,
                    null,
                    null,
                    null,
                    variablesToBeSubstituted);
```

editSubmitJobStreamWithVarSub

Describes the editSubmitJobStreamWithVarSub Web service for Tivoli Workload Scheduler for z/OS.

Description

Use this service to submit a job stream (application) in the Tivoli Workload Scheduler for z/OS plan and to perform some or all of the following actions:

- Assign values to any variables present in the jobs included in the job stream
- Add, modify, or delete jobs, and their internal dependencies, in the job stream

The variable substitution is performed only on promptable variables that are substituted at job set up phase. The other variables are substituted at submission time.

Note: The job stream must include no setup jobs. If it does, the following message is displayed:

```
EQQM229E
   JCL BROWSE/EDIT CAN ONLY BE SELECTED FOR PROCESSOR WORKSTATIONS.
```

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS engine.

jsKey The key identifying the job stream in the Tivoli Workload Scheduler for z/OS database. The valid format is *jobStreamName*.

schedTime
The input arrival time for the job stream.

deadlineTime
The completion dead line time for the job stream.

priority
The job stream priority value.

description
The job stream description. Maximum 24 characters.

groupName
The job stream group name. Maximum 16 characters.

ownerName
The job stream owner name. Maximum 16 characters.

ownerDescription
The job stream owner description. Maximum 24 characters.

authorityGroup
The job stream authority group. Maximum 8 characters.

dependenciesResolution
Specifies which type of dependencies are to be resolved. The value can be:

Y Resolve both predecessor and successor dependencies.

N Ignore all dependencies.

P Resolve only predecessor dependencies.

S Resolve only successor dependencies.

If no value is entered, the default is **N**.

variableTable
The name of the variable table associated with the job stream. Maximum 16 characters.

jobsList
An array of `ZOSJob` statements specifying the jobs that will be added, edited, and deleted in the job stream. The `ZOSJob` type is defined and documented in the `TWS-Types.xsd` file and in "Add, modify, or delete jobs in the job stream you are submitting" on page 56

dependencyList
An array of `Dependency` statements specifying the dependencies that will be added and deleted in the job stream. The `Dependency` type is defined and documented in the `TWS-Types.xsd` file and in "Add or delete internal dependencies in the plan" on page 58

variablesToBeSubstituted
An array of `Property` statements specifying the variables that will

be substituted in the job stream. The Property statement is defined and documented in the TWS-Types.xsd file, and has the following form:

value An array of values for the dataType.

dataType
A string identifying the variable for which you are supplying the value or values, and which can be found in the defined variable table.

Output

The service returns an array of the identifiers of the job streams that were actually submitted in the plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

Example:
The following is a coding example of editSubmitJobStreamWithVarSub:

```
long now = System.currentTimeMillis();

 /**
  * inputArrivalTime
  */
 Calendar schedTime = new GregorianCalendar();
 Date schedDate = new Date(now);
 schedTime.setTime(schedDate);

 /**
  * deadlineTime
  */
 Calendar deadlineTime = new GregorianCalendar();
 Date deadlineDate = new Date(now + HOUR_8);
 deadlineTime.setTime(deadlineDate);

 /**
  * variablesToBeSubstituted
  */
 Property prop1 = new Property();
 prop1.setDataType("SURNAME");
 String[] valueSurname = {"Robinson"};
 prop1.setValue(valueSurname);

 Property prop2 = new Property();
 prop2.setDataType("NAME");
 String[] valueName = {"John"};
 prop2.setValue(valueName);

 Property[] variablesToBeSubstituted = new Property[2];
 variablesToBeSubstituted[0] = prop1;
 variablesToBeSubstituted[1] = prop2;

 ZOSJob[] zj = new ZOSJob[1];
 zj[0] = new ZOSJob();
 zj[1] = new ZOSJob();
 zj[2] = new ZOSJob();

 zj[0].setAction("ADD");
 zj[0].setJobNumber(50);
 zj[0].setWorkstationName("VALL");
 zj[0].setDuration(10000);
 zj[0].setParallelServer(1);
 zj[0].setAutoSubmit("true");
 zj[0].setJobName("VPCEN1");
 zj[0].setInternalStatus("W");
```

```
zj[1].setAction("MODIFY");
zj[1].setJobNumber(10);
zj[1].setWorkstationName("VAL1");
zj[1].setJobName("VPCEN2");
zj[1].setInternalStatus("R");

zj[2].setAction("DELETE");
zj[2].setJobNumber(50);

Dependency[] dj = new Dependency[2];

dj[0] = new Dependency();
dj[0].setAction("ADD");
dj[0].setType("PREDECESSOR");
dj[0].setJobNumber(50);
dj[0].setDependencyNumber(40);

dj[0] = new Dependency();
dj[0].setAction("DELETE");
dj[0].setType("PREDECESSOR");
dj[0].setJobNumber(40);
dj[0].setDependencyNumber(20);

String[] jstreams = proxy.editSubmitJobStreamWithVarSub(ENGINE_NAME,
            JOB_STREAM_KEY,
              schedTime,
              deadlineTime,
              null,
              null,
                    null, null, null, null, null, zj, dj, variablesToBeSubstituted);
```

editSubmitJobStreamWithJobVarSub

Describes the editSubmitJobStreamWithJobVarSub Web service for Tivoli Workload Scheduler for z/OS.

Description

Use this service to submit a job stream (application) in the Tivoli Workload Scheduler for z/OS plan and to perform some or all of the following actions:

- Assign values to any variables present in the jobs included in the job stream with the option of specifying a different set of variables for the individual jobs.
- Add, modify, or delete jobs, and their internal dependencies, in the job stream.

The variable substitution is performed only on promptable variables that are substituted at job set up phase. The other variables are substituted at submission time.

Note: The job stream must include no setup jobs. If it does, the following message is displayed:

```
EQQM229E
   JCL BROWSE/EDIT CAN ONLY BE SELECTED FOR PROCESSOR WORKSTATIONS.
```

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS engine.

jsKey The key identifying the job stream in the Tivoli Workload Scheduler for z/OS database. The valid format is *jobStreamName*.

schedTime
The input arrival time for the job stream.

deadlineTime
The completion dead line time for the job stream.

priority
The job stream priority value.

description
The job stream description. Maximum 24 characters.

groupName
The job stream group name. Maximum 16 characters.

ownerName
The job stream owner name. Maximum 16 characters.

ownerDescription
The job stream owner description. Maximum 24 characters.

authorityGroup
The job stream authority group. Maximum 8 characters.

dependenciesResolution
Specifies which type of dependencies are to be resolved. The value can be:

Y Resolve both predecessor and successor dependencies.

N Ignore all dependencies.

P Resolve only predecessor dependencies.

S Resolve only successor dependencies.

If no value is entered, the default is **N**.

variableTable
The name of the variable table associated with the job stream. Maximum 16 characters.

jobsList
An array of `ZOSJob` statements specifying the jobs that will be added, edited, and deleted in the job stream. The `ZOSJob` type is defined and documented in the `TWS-Types.xsd` file and in "Add, modify, or delete jobs in the job stream you are submitting" on page 56

dependencyList
An array of `Dependency` statements specifying the dependencies that will be added and deleted in the job stream. The `Dependency` type is defined and documented in the `TWS-Types.xsd` file and in "Add or delete internal dependencies in the plan" on page 58

variablesToBeSubstituted
An array of `Property` statements specifying the variables that will be substituted in the job stream. The `Property` statement is defined and documented in the `TWS-Types.xsd` file, and has the following form:

value An array of values for the dataType.

dataType
A string identifying the variable for which you are supplying the value or values, and which can be found in the defined variable table.

jobVariablesToBeSubstituted
An iteration of the following items:

An array of `Property` statements specifying the variables that will be substituted in the job stream. The `Property` statement is defined and documented in the `TWS-Types.xsd` file, and has the following form:

value An array of values for the dataType.

dataType
A string identifying the variable for which you are supplying the value or values, and which can be found in the defined variable table.

jobNumber
The integer (from 1 to 255) that identifies the job that will use the variables specified below and that must have been previously defined with the `.setJobNumber(`*`number`*`)`

For example:

```
String [][] variablesMap1 = new String [3][2];
variablesMap1 [0][0] = "VAR1";
variablesMap1 [0][1] = "ValVar1ForExec1";
variablesMap1 [1][0] = "VAR2";
variablesMap1 [1][1] = "ValVar2ForExec1";
variablesMap1 [2][0] = "VAR3";
variablesMap1 [2][1] = "ValVar3ForExec1";
int jobNum = 5;

jobVariablesToBeSubstituted.put(jobNum, variablesMap1);

String [][] variablesMap2 = new String [2][2];
variablesMap2 [0][0] = "VAR2";
variablesMap2 [0][1] = "ValVar2ForExec2";
variablesMap2 [1][0] = "VAR4";
variablesMap2 [1][1] = "ValVar4ForExec2";
jobNum = 10;

jobVariablesToBeSubstituted.put(jobNum, variablesMap2);

...

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);
```

The sequence repeats for all the jobs that use their own particular variables or variable values rather than those specified for the whole job stream. Any jobs that are not in the iteration use either

the variables or variable values specified with the `variablesToBeSubstituted` parameter for the job stream or no variables at all.

Output

The service returns an array of the identifiers of the job streams that were actually submitted in the plan. Note that the plan object identifiers returned contain the `'\0'` character. This is not a valid character and must be replaced by a blank.

Example:

The following is a coding example of `editSubmitJobStreamWithJobVarSub`:

```
String description = "";
 String group = null;
 String owner = null;
 String ownerDescription = null;
 String variableTable = "STREAM1";

 List<ZosJobInfo>jobsToDelete = new ArrayList<ZosJobInfo>();
     List<ZosJobInfo>jobsToAdd = new ArrayList<ZosJobInfo>();

     List<ZosJobInfo>jobsToModify = new ArrayList<ZosJobInfo>();
     ZosJobInfo job1 = new ZosJobInfo();
     job1.setJobNumber(5);
     job1.setJobName("EXEC1");
     jobsToModify.add(job1);
     ZosJobInfo job2 = new ZosJobInfo();
     job2.setJobNumber(10);
     job2.setJobName("EXEC2");
     jobsToModify.add(job2);
     ZosJobInfo job21 = new ZosJobInfo();
     job21.setJobNumber(15);
     job21.setJobName("EXEC3");
     jobsToModify.add(job21);
     ZosJobInfo job3 = new ZosJobInfo();
     job3.setJobNumber(20);
     job3.setJobName("EXEC4");
     jobsToModify.add(job3);

     HashMap<String, List<Integer>> dependencyToDelete = new HashMap<String, List<Integer>>();
     HashMap<String, List<Integer>> dependencyToAdd = new HashMap<String, List<Integer>>();

     // default (for all jobs) JCL variables values
  String[][] variablesToBeSubstituted = new String [1][2];
 variablesToBeSubstituted [0][0] = "VAR1";
 variablesToBeSubstituted [0][1] = "ValVar1ForExec2And3";

 String authorityGroup = null;
 boolean holdAll = true;
 DependenciesResolutionOption resolutionOption = DependenciesResolutionOption.RESOLUTION_ALL;

 // job level JCL variables values
 HashMap<Integer, String[][]> jobVariablesToBeSubstituted = new HashMap<Integer, String[][]>();
  String [][] variablesMap1 = new String [3][2];
  variablesMap1 [0][0] = "VAR1";
  variablesMap1 [0][1] = "ValVar1ForExec1";
  variablesMap1 [1][0] = "VAR2";
  variablesMap1 [1][1] = "ValVar2ForExec1";
  variablesMap1 [2][0] = "VAR3";
  variablesMap1 [2][1] = "ValVar3ForExec1";
  int jobNum = 5;

  jobVariablesToBeSubstituted.put(jobNum, variablesMap1);

 String [][] variablesMap2 = new String [2][2];
 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);

Context context = null;


@SuppressWarnings("unchecked")
     List<Identifier> list = plan.editAddJobStreamInstanceWithVariableSubstitution(
                             JSName, startTime, deadlineTime, priority, description,
                             group, owner, ownerDescription, variableTable, jobsToDelete,
                             jobsToAdd, jobsToModify, dependencyToDelete, dependencyToAdd,
                             variablesToBeSubstituted, authorityGroup, holdAll,
                             resolutionOption, jobVariablesToBeSubstituted, context);
```

Add, modify, or delete jobs in the job stream you are submitting

To carry out any of these actions, you must write an array of `ZOSJob` statements containing the specifications for these actions. If the `jobsList` array is empty, or if it contains null values (except where allowed), no action is taken.

Table 7 describes the parameters you need to specify in each `ZOSJob` statement to specify a job you want to add, modify, or delete in the job stream you are submitting to the plan.

Table 7. Properties to set for added, modified, and deleted jobs in the `ZOSJob` elements.

Element	Action to perform on job in job stream		
	Add	**Modify**	**Delete**
action	Set to `ADD`. If left void or with an unauthorized value, an error message is returned.	Set to `MODIFY`. If left void or with an unauthorized value, an error message is returned.	Set to `DELETE`. If left void or with an unauthorized value, an error message is returned.
jobNumber	Set to a value between 1 and 255. If left void, an error message is returned.	Set to a value between 1 and 255 corresponding to the job number in the plan. If left void, an error message is returned.	Set to a value between 1 and 255 corresponding to the job number in the plan. If left void, an error message is returned.
workstationName	Set to the ID of the workstation as defined in the current plan. Maximum 4 characters. If left void, an error message is returned.	Set to the ID of the workstation as defined in the current plan. Maximum 4 characters. If left void, the last-saved value is used.	Enter a null value.

Table 7. Properties to set for added, modified, and deleted jobs in the `ZOSJob` elements. (continued)

Element	Action to perform on job in job stream		
	Add	**Modify**	**Delete**
textDescription	Optional. Maximum 24 characters.	Maximum 24 characters. If left void, the last-saved description is used.	Enter a null value.
jobName	Set to the name of the job. Maximum 8 characters. If left void, an error message is returned.	Maximum 8 characters. If left void, the last-saved value is used.	Enter a null value.
parallelServer	The number of parallel servers required to run the job. If left void, it takes 0 as the default value. Depending on the workstation in some cases may return an error message.	Set to -1 to leave the last-saved value. The default is 0.	Enter a null value.
duration	Set in milliseconds. If left void, an error message is returned.	Set to -1 to leave the last-saved value. The default is 0.	Enter a null value.
autoSubmit	If auto submit is required, set to `true` otherwise set to `false`. If left void, takes`false`.	If auto submit is required, set to `true` otherwise set to `false`. If left void, takes the last-saved value.	Enter a null value.
timeDependent	If the job is time-dependent, set to `true` otherwise set to `false`. If left void, takes `false`.	If the job is time-dependent, set to `true` otherwise set to `false`. If left void, takes the last-saved value.	Enter a null value.
centralizedScript	If the job has a centralized script, set to `true` otherwise set to `false`. If left void, takes `true`. This option is only available on jobs scheduled to run on a fault-tolerant agent.	Enter a null value (cannot be modified from the original value).	Enter a null value.
inputArrivalTime	The input arrival time in HH.MM format.	The input arrival time in HH.MM format. If left void, the last-saved value is used.	Enter a null value.
R1	The number of instances of job resource 1 required. If left void, it takes 0 as value.	The number of instances of job resource 1 required. Set to -1 to leave the last-saved value. The default is 0.	Enter a null value.

Table 7. Properties to set for added, modified, and deleted jobs in the `ZOSJob` elements. (continued)

Element	Action to perform on job in job stream		
	Add	**Modify**	**Delete**
R2	The number of instances of job resource 2 required. If left void, it takes 0 as value.	The number of instances of job resource 2 required. Set to -1 to leave the last-saved value. The default is 0.	Enter a null value.
internalStatus	Can be one of the following: • COMPLETE • DELETED • ERROR • WAITING • STARTED • UNDECIDED • READY • PENDINGPRED • WAITINGFORINPUT • INTERRUPTED • NOREPORTINGPRED If left void, takes `UNDECIDED` as value.	If left void, the last-saved value is used.	Enter a null value.

Remember to define the dependencies (if there are any) of the new jobs you add.

Add or delete internal dependencies in the plan

To perform either of these actions, you must write an array of `Dependency` statements containing the specifications for these actions. If the `dependecyLst` array is empty, or if it contains null values, no action is taken.

The following describes the parameters you need to specify in each `Dependency` statement to specify a dependency you want to add or delete in a job stream in the plan. If all elements are not defined, an error is returned:

action The action to be taken on the dependency. Can be either `ADD` or `DELETE`.

type The type of dependency. Can be either `PREDECESSOR` or `SUCCESSOR`.

jobNumber
The job number of the job impacted.

dependencyNumber
The job number of the job that you are defining as a predecessor or successor of *jobNumber*.

queryJobStreams (z/OS)

Describes the queryJobStreams Web service for Tivoli Workload Scheduler for z/OS.

Description
Use this service to run queries on Tivoli Workload Scheduler for z/OS job stream instances.

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS engine.

filter A list of filtering criteria represented by an array of `filterCriteria` statements. The `filterCriteria` statements are defined and documented in the `TWS-Types.xsd` file and in "Defining the filtering criteria to query z/OS job streams."

Output
The response `queryJobStreamsResponse` returns an array `queryJobStreamsReturn` of the identifiers of the job streams that matched the query. The array includes the following fields:

occurrenceToken
The occurrence identifier.

owner The owner ID defined for the application.

authorityGroup
The name of the authority group that the application belongs to.

containingMonitoredJob
The application includes at least one monitored operation. Can be `true` or `false`.

Defining the filtering criteria to query z/OS job streams:
Each `filterCriteria` statement has the following form:

details
Not used; set to null.

value An array of values for the dataType.

minimum
A single value representing the minimum of a range.

maximum
A single value representing the maximum of a range.

dataType
A string identifying the field for which you have supplied a value, several values, or a range. The fields are defined and documented in the `SchedulingFactory.wsdl` file and are as follows:

JOB_STREAM_NAME
The name of the job stream.

WORKSTATION_NAME
The name of the workstation that ran the job stream.

STATUS_LIST
Can be one of the following:
- BLOCKED
- CANCELLED
- COMPLETED
- ERROR
- HELD
- READY
- STARTED
- WAITING
- UNDECIDED

INTERNAL_STATUS_LIST
Can be one of the following:

- COMPLETE
- DELETED
- ERROR
- WAITING
- STARTED
- UNDECIDED

PRIORITY
The execution priority with which the job stream ran. Can be one of the following:
- A number ranging from 0 to 99
- hi
- go

PRIORITY_RANGE
Used if you want to query job streams within a range of priority values. In this case, `minimum` and `maximum` must also be supplied.

START_TIME
The start time defined for the job stream in `YYYYMMDD HHMM` format or milliseconds.

START_TIME_RANGE
Used if you want to query job streams within a range of start times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

DEADLINE_TIME
The deadline time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

DEADLINE_TIME_RANGE
Used if you want to query job streams within a range of deadline times in `YYYYMMDD HHMM` format or milliseconds. In this case, `minimum` and `maximum` must also be supplied.

OCCURRENCE_TOKEN
The occurrence token of the job stream instance in the plan (in hexadecimal).

OWNER
The owner of the job stream.

AUTH_GROUP
The authority group assigned to the job stream.

MONITORED_JOB
Determines if only monitored jobs are to be selected. Can be `true` or `false`.

JobService details

Describes the Web services available in the JobService.wsdl file.

This section describes the web services you can use from the `JobService.wsdl` file.

Web services are described separately for Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS.

JobService web services for Tivoli Workload Scheduler

Describes the web services that are available in JobService.wsdl for Tivoli Workload Scheduler.

The following web services can be used to interface with job instances in Tivoli Workload Scheduler.

getProperties

Describes the getProperties Web service for Tivoli Workload Scheduler jobs.

Description

Use this service to display information about a job instance.

You must have `list` access to the job in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobId The job identifier in the plan: *workstationName#jobStreamName.jobName*.

Output

The response `getPropertiesResponse` returns an array `getPropertiesReturn` containing the following information fields:

jobId The job identifier.

jobName
The name of the job.

jobStreamName
The name of the job stream including the job.

workstationName
The name of the workstation that ran the job.

jobStreamWorkstationName
The name of the workstation associated with the job stream.

jobNumber
The number assigned to the job at runtime.

priority
The execution priority with which the job ran. Can be one (or a range of values) of the following:
- A number ranging from 0 to 99
- hi
- go

status Can be one of the following:
- ERROR
- HELD
- READY
- RUNNING
- SUCCESSFULL
- UNDECIDED
- WAITING

internalStatus
Can be one of the following:
- ABEND

- ABENDP
- BOUND
- CANCP
- DONE
- ERROR
- EXEC
- EXTRN
- FAIL
- FENCE
- HOLD
- INTRO
- PEND
- READY
- RJOB
- SCHED
- SUCC
- SUCCP
- SUSP
- USER
- WAIT
- WAITD

requiredConfirmation
Can be `true` or `false`.

aliased
Can be `true` or `false`.

canceled
Can be `true` or `false`.

every The job instance was run with the `every` option. Can be `true` or `false`.

everyRerun
The job instance is a rerun of a job defined with the `every` option. Can be `true` or `false`.

external
The job is a predecessor to another job stream or one of its jobs. Can be `true` or `false`.

jobLate
The job passed its completion deadline. Can be `true` or `false`.

pendingCancellation
The job stream is pending cancellation. Cancellation is deferred until all of the dependencies, including an `at` time, are resolved. Can be `true` or `false`.

recoveryRerunJob
The job is defined with the recovery action RERUN. This enables the recovery job to take some corrective action, before the parent job attempts to run again. Can be `true` or `false`.

released
The job was released from its dependencies. Can be `true` or `false`.

rerunJob
The job was rerun. Can be `true` or `false`.

running
The job is still running. Can be `true` or `false`.

startTime
The start time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

lateststartTime
The value of the `until` restriction of the job instance in `YYYYMMDD HHMM` format or milliseconds.

lateststartAction
The action taken after the `until` restriction time elapsed. Can be CANCEL, CONTINUE, or SUPPRESS.

deadlineTime
The value of the deadline restriction of the job instance in `YYYYMMDD HHMM` format or milliseconds.

repeatRange
The time interval between `every` reruns of the job in milliseconds.

If the service did not run successfully, `getPropertiesResponse` returns one of the errors described in "Managing errors" on page 35.

setProperties

Describes the setProperties Web service for Tivoli Workload Scheduler jobs.

Description

Use this service to define additional properties for a job in the plan.

You must have `submit` access to the job in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobId The job identifier in the plan: *workstationName#jobStreamName.jobName*.

properties
An array containing the properties you want to set. The properties can be:

priorityIsMonitored
The job priority is monitored if it is a key job. The value is either `true` or `false`.

requiresConfirmation
Operator confirmation is required after the job completes to mark it as successful or failed. The value is either `true` or `false`.

startTime
The time when the job must start in `YYYYMMDD HHMM` format or milliseconds.

lateststartTime
The latest time when the job must start in `YYYYMMDD HHMM` format or milliseconds.

lateststartAction
The action that will be taken if the job exceeds its latest start time. Can be CANCEL, CONTINUE, or SUPPRESS.

deadlineTime
The time within which the job must complete in `YYYYMMDD HHMM` format or milliseconds.

repeatRange
The time interval between `every` reruns of the job in milliseconds.

Output
The response `setPropertiesResponse` is void if the service ran succesfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

getOutput

Describes the getOutput Web service for Tivoli Workload Scheduler jobs.

Description
Use this service to get the execution log of a job in the plan.

You must be logged on as the TWS_user to have proper authorization to run this service.

Input parameters

engineName
Not used; set to null.

jobId The job identifier in the plan: *workstationName#jobStreamName.jobName*.

Output
The response `getOutputResponse` returns a string containing the execution log of the specified job.

If the service did not run successfully, `getOutputResponse` returns one of the errors described in "Managing errors" on page 35.

kill

Describes the kill Web service for Tivoli Workload Scheduler jobs.

Description
Use this service to stop a job that is running.

You must have `kill` access to the job in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobId The job identifier in the plan: *workstationName#jobStreamName.jobName*.

Output
The response `killResponse` is void if the service ran successfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

cancel

Describes the cancel Web service for Tivoli Workload Scheduler jobs.

Description

Use this service to cancel a job.

If you cancel the job before it is started, it does not start. If you cancel it after it was started, it continues to run. If you cancel a job that is running and it completes in the ABEND state, no automatic job recovery steps are attempted.

You must have `cancel` access to the job in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobId The job identifier in the plan: *workstationName#jobStreamName.jobName*.

isPending
Value is `true` or `false`. `True` cancels the job only after its dependencies are resolved. `False` cancels the job immediately (and any jobs and job streams that are dependent on the cancelled job are released immediately from the dependency).

Output

The response `cancelResponse` is void if the service ran successfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

releaseAllDependencies

Describes the releaseAllDependencies Web service for Tivoli Workload Scheduler jobs.

Description

Use this service to release a job from all its defined dependencies.

You must have `release` access to the job in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobId The job identifier in the plan: *workstationName#jobStreamName.jobName*.

Output

The response `releaseAllDependenciesResponse` is void if the service ran successfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

JobService web services for Tivoli Workload Scheduler for z/OS

Describes the web services that are available in JobService.wsdl for Tivoli Workload Scheduler for z/OS.

The following web services can be used to interface with operations in Tivoli Workload Scheduler for z/OS.

getProperties (z/OS)

Describes the getProperties Web service for Tivoli Workload Scheduler for z/OS operations.

Description

Use this service to display information about an operation.

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS controller.

jobId The operation identifier in the current plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

Output

The response `getPropertiesResponse` returns an array `getPropertiesReturn` containing the following information fields:

occurrenceToken
The identifier of the occurrence that includes the operation.

extendedStatus
The extended status code assigned to the operation.

errorCode
The error code that terminated the operation.

operationCommand
Can be one of the following: BD = Bind shadow; job EX = Execute operation; KJ = Kill operation; KR = Kill recovery job; MH = Hold operation; MR = Release operation; NP = NOP operation; PN = Prompt reply no; PY = Prompt reply yes; UN = Un-NOP operation.

authorityGroup
The name of the authority group that the operation belongs to.

cleanUpStatus
Can be one of the following: Blank=none C=Completed E=Ended in error I=Initiated O=Avail opinfo R=Request opinfo S=Started W=Waiting opinfo

latestOutPassed
The operation exceeded its latest starting time and is late. Can be `true` or `false`.

latestOut
The latest possible time, calculated at plan-creation-time, that the operation can start to meet the deadline time (HHMM).

actualArrival
The actual run time of the operation (HHMM).

actualEnd
The time the operation is reported as complete or ended-in-error (HHMM).

If the service did not run successfully, `getPropertiesResponse` returns one of the errors described in "Managing errors" on page 35.

setProperties (z/OS)

Describes the setProperties Web service for Tivoli Workload Scheduler for z/OS operations.

Description

Use this service to define additional properties for an operation.

Input parameters

engineName

The name of the Tivoli Workload Scheduler for z/OS controller.

jobId The operation identifier in the current plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

properties

An array containing the properties you want to set. The properties can be:

priorityIsMonitored

The priority is monitored if it is a key operation. The value is either `true` or `false`.

startTime

The time when the operation must start in `YYYYMMDD HHMM` format or milliseconds.

lateststartTime

The latest time when the operation must start in `YYYYMMDD HHMM` format or milliseconds.

lateststartAction

The action that will be taken if the operation exceeds its latest start time. Can be CANCEL, CONTINUE, or SUPPRESS.

deadlineTime

The time within which the operation must complete in `YYYYMMDD HHMM` format or milliseconds.

Output

The response `setPropertiesResponse` is void if the service ran successfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

getOutput (z/OS)

Describes the getOutput Web service for Tivoli Workload Scheduler for z/OS operations.

Description

Use this service to get the job log of an operation.

Input parameters

engineName

The name of the Tivoli Workload Scheduler for z/OS controller.

jobId The operation identifier in the current plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

Output

The response `getOutputResponse` returns a string containing the job log of the specified operation.

If the service did not run successfully, `getOutputResponse` returns one of the errors described in "Managing errors" on page 35.

cancel (z/OS)

Describes the cancel Web service for Tivoli Workload Scheduler for z/OS operations.

Description

Use this service to cancel an operation.

Note that this service works only if the operation has not started. If the operation has already started, it continues to run.

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS controller.

jobId The operation identifier in the current plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

isPending
The operation is in pending status. The value can be `true` or `false`.

Output

The response `cancelResponse` is void if the service ran successfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

JobStreamService details

Describes the Web services available in the JobStreamService.wsdl file.

This section describes the web services you can use from the `JobStreamService.wsdl` file.

Web services are described separately for Tivoli Workload Scheduler and Tivoli Workload Scheduler for z/OS.

JobStreamService web services for Tivoli Workload Scheduler

Describes the web services that are available in JobStreamService.wsdl for Tivoli Workload Scheduler.

The following web services can be used to interface with job stream instances in Tivoli Workload Scheduler.

getProperties

Describes the getProperties Web service for Tivoli Workload Scheduler job streams.

Description

Use this service to display information about a job stream instance.

You must have `list` access to the job stream in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobStreamId
The job stream identifier in the plan: either

workstationName#jobStreamName(hhmm[date]) or *workstation#jobstream_id*. See the *Tivoli Workload Scheduler: User's Guide and Reference* for details.

Output

The response `getPropertiesResponse` returns an array `getPropertiesReturn` containing the following information fields:

jobStreamId
The job stream instance identifier.

jobStreamName
The name of the job stream.

aliasJobStreamName
The job stream alias name.

originalJobStreamName
The name of the job stream as defined in the database.

workstationName
The name of the workstation associated with the job stream.

status Can be one of the following:
- BLOCKED
- CANCELLED
- ERROR
- HELD
- READY
- RUNNING
- SUCCESSFUL
- UNDECIDED
- WAITING

internalStatus
Can be one of the following:
- ABEND
- ABENDP
- CANCP
- DONE
- ERROR
- EXEC
- EXTRN
- FAIL
- FENCE
- HOLD
- INTRO
- PEND
- READY
- RJOB
- SCHED
- SUCC
- SUCCP
- SUSP
- USER
- WAIT
- WAITD

limit The job limit of the job stream.

priority
The execution priority defined for the job stream. Can be one (or a range of values) of the following:
- A number ranging from 0 to 99
- hi
- go

numberOfJobs
The number of jobs included in the job stream.

canceled
The job stream instance is in the `cancelled` status. Can be `true` or `false`.

carriedForward
The job stream instance was carried forward. Can be `true` or `false`.

carryForward
The job stream was scheduled with the `carryforward` option. Can be `true` or `false`.

external
The job stream is a predecessor to another job stream or to one of its jobs located in another Tivoli Workload Scheduler network. Can be `true` or `false`.

lateJobStream
The job stream instance passed its completion deadline. Can be `true` or `false`.

pendingCancellation
The job stream is pending cancellation. Cancellation is deferred until all of the dependencies, including an `at` time, are resolved. Can be `true` or `false`.

released
The job stream was released from its dependencies. Can be `true` or `false`.

startTime
The start time defined for the job stream in `YYYYMMDD HHMM` format or milliseconds.

lateststartTime
The value of the `until` restriction of the job stream in `YYYYMMDD HHMM` format or milliseconds.

lateststartAction
The action taken after the `until` restriction time elapsed. Can be CANCEL, CONTINUE, or SUPPRESS.

deadlineTime
The value of the deadline restriction of the job stream instance in `YYYYMMDD HHMM` format or milliseconds.

If the service did not run successfully, `getPropertiesResponse` returns one of the errors described in "Managing errors" on page 35.

setProperties

Describes the setProperties Web service for Tivoli Workload Scheduler job streams.

Description
Use this service to define additional properties for a job stream in the plan.

You must have `submit` access to the job stream in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobStreamId
The job stream identifier in the plan: either *workstationName#jobStreamName(hhmm[date])* or *workstation#jobstream_id*. See the *Tivoli Workload Scheduler: User's Guide and Reference* for details.

properties
An array containing the properties you want to set. The properties can be:

limit The number of jobs in the job stream that can run simultaneously on the same workstation.

priority
The execution priority. Can be one (or a range of values) of the following:
- A number ranging from 0 to 99
- hi
- go

isCarryForward
The value is either `true` or `false`.

isMonitored
The job stream is monitored. The value is either `true` or `false`.

startTime
The time when the job stream must start in `YYYYMMDD HHMM` format or milliseconds.

lateststartTime
The latest time when the job stream must start in `YYYYMMDD HHMM` format or milliseconds.

lateststartAction
The action that will be taken if the job stream exceeds its latest start time. Can be CANCEL, CONTINUE, or SUPPRESS.

deadlineTime
The time within which the job stream must complete in `YYYYMMDD HHMM` format or milliseconds.

Output
If the service ran successfully, the response `setPropertiesResponse` returns the identifier of the modified job stream; otherwise, it returns one of the errors described in "Managing errors" on page 35.

getJobsList

Describes the getJobsList Web service for Tivoli Workload Scheduler job streams.

Description
Use this service to get a list of the jobs - and their properties - of a job stream in the plan.

You must have `list` access to the job stream in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobStreamId
The job stream identifier in the plan: either *workstationName#jobStreamName(hhmm[date])* or *workstation#jobstream_id*. See the *Tivoli Workload Scheduler: User's Guide and Reference* for details.

Output

The response `getJobsListResponse` returns an array `getJobsListReturn` containing the list of the jobs included in the job stream. For each job instance the following details are also listed:

jobId The job identifier.

jobName
The name of the job.

jobStreamName
The name of the job stream including the job.

workstationName
The name of the workstation that ran the job.

jobStreamWorkstationName
The name of the workstation associated with the job stream.

jobNumber
The number assigned to the job at runtime.

priority
The execution priority with which the job ran. Can be one (or a range of values) of the following:
- A number ranging from 0 to 99
- hi
- go

status Can be one of the following:
- ERROR
- HELD
- READY
- RUNNING
- SUCCESSFULL
- UNDECIDED
- WAITING

internalStatus
Can be one of the following:
- ABEND
- ABENDP
- BOUND
- CANCP
- DONE
- ERROR
- EXEC
- EXTRN

- FAIL
- FENCE
- HOLD
- INTRO
- PEND
- READY
- RJOB
- SCHED
- SUCC
- SUCCP
- SUSP
- USER
- WAIT
- WAITD

requiredConfirmation
Can be `true` or `false`.

aliased
Can be `true` or `false`.

canceled
Can be `true` or `false`.

every The job instance was run with the `every` option. Can be `true` or `false`.

everyRerun
The job instance is a rerun of a job defined with the `every` option. Can be `true` or `false`.

external
The job is a predecessor to another job stream or one of its jobs. Can be `true` or `false`.

jobLate
The job passed its completion deadline. Can be `true` or `false`.

pendingCancellation
The job stream is pending cancellation. Cancellation is deferred until all of the dependencies, including an `at` time, are resolved. Can be `true` or `false`.

recoveryRerunJob
The job is defined with the recovery action RERUN. This enables the recovery job to take some corrective action, before the parent job attempts to run again. Can be `true` or `false`.

released
The job was released from its dependencies. Can be `true` or `false`.

rerunJob
The job was rerun. Can be `true` or `false`.

running
The job is still running. Can be `true` or `false`.

startTime
The start time defined for the job in `YYYYMMDD HHMM` format or milliseconds.

lateststartTime
The value of the `until` restriction of the job instance in `YYYYMMDD HHMM` format or milliseconds.

lateststartAction
The action taken after the `until` restriction time elapsed. Can be CANCEL, CONTINUE, or SUPPRESS.

deadlineTime
The value of the deadline restriction of the job instance in `YYYYMMDD HHMM` format or milliseconds.

repeatRange
The time interval between `every` reruns of the job in milliseconds.

If the service did not complete successfully, `getJobsListResponse` returns one of the errors described in "Managing errors" on page 35.

cancel

Describes the cancel Web service for Tivoli Workload Scheduler job streams.

Description

Use this service to cancel a job stream.

If you cancel the job stream before it is started, it does not start. If you cancel it after it was started, the jobs that have started complete, but no other jobs are launched.

You must have `cancel` access to the job stream in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobStreamId
The job stream identifier in the plan: either *workstationName#jobStreamName(hhmm[date])* or *workstation#jobstream_id*. See the *Tivoli Workload Scheduler: User's Guide and Reference* for details.

isPending
The value can be `true` or `false`. `True` cancels the job stream only after its dependencies are resolved. `False` cancels the job stream immediately (and any jobs and job streams that are dependent on the cancelled job stream are released immediately from the dependency).

Output

The response `cancelResponse` is void if the service ran successfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

releaseAllDependencies

Describes the releaseAllDependencies Web service for Tivoli Workload Scheduler job streams.

Description

Use this service to release a job stream from all its defined dependencies.

You must have `release` access to the job stream in the security file to run this service.

Input parameters

engineName
Not used; set to null.

jobStreamId
The job stream identifier in the plan: either *workstationName#jobStreamName(hhmm[date])* or *workstation#jobstream_id*. See the *Tivoli Workload Scheduler: User's Guide and Reference* for details.

Output
The response `releaseAllDependenciesResponse` is void if the service ran successfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

JobStreamService web services for Tivoli Workload Scheduler for z/OS

Describes the Web services available in JobStreamService.wsdl for Tivoli Workload Scheduler for z/OS.

The following Web services can be used to interface with application occurrences in Tivoli Workload Scheduler for z/OS.

getProperties (z/OS)

Describes the getProperties Web service for Tivoli Workload Scheduler for z/OS application occurrences.

Description
Use this service to display information about an application occurrence.

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS controller.

jobStreamId
The occurrence identifier in the current plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

Output
The response `getPropertiesResponse` returns an array `getPropertiesReturn` containing the following information fields:

occurrenceToken
The occurrence identifier.

owner The owner ID defined for the application.

authorityGroup
The name of the authority group that the application belongs to.

containingMonitoredJob
The application includes at least one monitored operation. Can be `true` or `false`.

If the service did not run successfully, `getPropertiesResponse` returns one of the errors described in "Managing errors" on page 35.

setProperties (z/OS)

Describes the setProperties Web service for Tivoli Workload Scheduler for z/OS application occurrences.

Description

Use this service to define additional properties for an application occurrence.

Input parameters

engineName

The name of the Tivoli Workload Scheduler for z/OS controller.

jobStreamId

The occurrence identifier in the current plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

properties

An array containing the properties you want to set. The properties can be:

priority

The execution priority. The value can be from 1 (lowest) to 9 (urgent).

isMonitored

The occurrence includes monitored operations. The value is either `true` or `false`.

startTime

The time when the occurrence must start in `YYYYMMDD HHMM` format or milliseconds.

lateststartTime

The latest time when the occurrence must start in `YYYYMMDD HHMM` format or milliseconds.

lateststartAction

The action that will be taken if the occurrence exceeds its latest start time. Can be CANCEL, CONTINUE, or SUPPRESS.

deadlineTime

The time within which the occurrence must complete in `YYYYMMDD HHMM` format or milliseconds.

Output

If the service ran successfully, the response `setPropertiesResponse` returns the identifier of the modified job stream; otherwise, it returns one of the errors described in “Managing errors” on page 35.

getJobsList (z/OS)

Describes the getJobsList Web service for Tivoli Workload Scheduler for z/OS application occurrences.

Description

Use this service to get a list of the operations - and their properties - that make up an application occurrence.

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS controller.

jobStreamId
The occurrence identifier in the current plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

Output

The response `getJobsListResponse` returns an array `getJobsListReturn` containing the list of the operations included in the occurrence. For each operation the following details are also listed:

occurrenceToken
The identifier of the occurrence that includes the operation.

extendedStatus
The extended status code assigned to the operation.

errorCode
The error code that terminated the operation.

operationCommand
Can be one of the following: BD = Bind shadow; job EX = Execute operation; KJ = Kill operation; KR = Kill recovery job; MH = Hold operation; MR = Release operation; NP = NOP operation; PN = Prompt reply no; PY = Prompt reply yes; UN = Un-NOP operation.

authorityGroup
The name of the authority group that the operation belongs to.

cleanUpStatus
Can be one of the following: Blank=none C=Completed E=Ended in error I=Initiated O=Avail opinfo R=Request opinfo S=Started W=Waiting opinfo

latestOutPassed
The operation exceeded its latest starting time and is late. Can be `true` or `false`.

latestOut
The latest possible time, calculated at plan-creation-time, that the operation can start to meet the deadline time (HHMM).

actualArrival
The actual run time of the operation (HHMM).

actualEnd
The time the operation is reported as complete or ended-in-error (HHMM).

If the service did not complete successfully, `getJobsListResponse` returns one of the errors described in "Managing errors" on page 35.

cancel (z/OS)

Describes the cancel Web service for Tivoli Workload Scheduler for z/OS application occurrences.

Description
Use this service to delete an application occurrence from the current plan.

Input parameters

engineName
The name of the Tivoli Workload Scheduler for z/OS controller.

jobStreamId
The occurrence identifier in the current plan. Note that the plan object identifiers returned contain the '\0' character. This is not a valid character and must be replaced by a blank.

isPending
The occurrence is in pending status. The value can be `true` or `false`.

Output
The response `cancelResponse` is void if the service ran successfully; otherwise, it returns one of the errors described in "Managing errors" on page 35.

Further information

Describes how to obtain further information about using Web services.

For more information about how to manage `WSDL` files to create your own Web Services-based user interface refer to the IBM redbook *IBM Redbooks: WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook*, *SG24-6891*, which has the following description:

- This IBM Redbook describes the new concept of Web services from various perspectives. It presents the major building blocks Web services rely on. Here, well-defined standards and new concepts are presented and discussed.
- Whereas these concepts are described vendor-independent, this book also presents IBM's view and illustrates with suitable demonstration applications how Web services can be implemented using IBM's product portfolio, especially WebSphere Application Server Version 5.1 and WebSphere Studio Application Developer Version 5.1.1.
- This book is a major update to the IBM Redbook Web Services Wizardry with WebSphere Studio Application Developer, SG24-6292, and to WebSphere Version 5 Web Services Handbook, SG24-6891-00.

To access this publication, follow this link: http://www.redbooks.ibm.com/abstracts/sg246891.html.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

P

Q

R

S

T

U

W

Product Number: 5698-T08, 5698-WSH, and 5698-WSE

Printed in USA

SC23-9608-03