IBM Tivoli Workload Scheduler

Scheduling Workload Dynamically

Version 8 Release 6

SC23-9856-03

IBM Tivoli Workload Scheduler

Scheduling Workload Dynamically

Version 8 Release 6

SC23-9856-03

Note
Before using this information and the product it supports, read the information in "Notices" on page 61.

This edition applies to version 9, release 1, 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

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.

This guide explains how to dynamically allocate resources to run your workload using the services of the dynamic workload broker component of Tivoli Workload Scheduler.

Dynamic workload broker is an on-demand scheduling infrastructure which provides dynamic management of your environment.

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=swg27038323 and the Dynamic Workload Console Release Notes at http://www-01.ibm.com/support/docview.wss?rs=672&uid=swg27038328.

Who should read this publication

Describes the type of user who should read the documentation.

This guide is intended for administrators responsible for defining user roles and performing high-level tasks and for operators responsible for creating and submitting jobs.

Readers should be familiar with the following topics:

- Working knowledge of IBM Tivoli Workload Scheduler
- PC and UNIX operating systems
- Graphical and command line interfaces

Publications

Full details of Tivoli Workload Scheduler publications can be found in *Tivoli Workload Automation: Publications*. This document also contains information about the conventions used in the publications.

A glossary of terms used in the product can be found in *Tivoli Workload Automation: Glossary*.

Both of these are in the Information Center as separate publications.

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

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

http://www.ibm.com/software/tivoli/education

Support information

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. Scheduling jobs

This chapter explains how to define and submit jobs.

This guide applies to customers which have been using Tivoli Workload Scheduler version 8.5.1 and have now moved to version 8.6.

Creating Tivoli Workload Scheduler jobs managed by dynamic workload broker

This section explains how to create Tivoli Workload Scheduler jobs to be managed by dynamic workload broker.

To create a Tivoli Workload Scheduler job and have resources allocated dynamically, perform the following steps:

1. Create a JSDL job definition in dynamic workload broker using the Job Brokering Definition Console. This job contains the instructions or program to be run.
2. Create a job to be submitted in Tivoli Workload Scheduler. This job contains the reference to the job in dynamic workload broker defined in step 1. Define the Tivoli Workload Scheduler job as follows:
   a. In the Dynamic Workload Console, from the navigation bar, click **Administration** > **Workload Design** > **Manage Workload Definitions**.
   b. Select **New > Job Definition > Cloud > Workload Broker**.
   c. In the **General** tab, in the **Workstation** field specify the workload broker workstation.
   d. In the **Task** tab, in the **Workload Broker job name** field specify the name of the JSDL job definition you created in step 1.

When you submit the Tivoli Workload Scheduler job, this job causes the referenced job to be submitted on dynamic workload broker. This method is known as submission by reference because you only need to reference the job you want to submit, without having to write or import the whole job in Tivoli Workload Scheduler.

Using variables in dynamic workload broker jobs

This section explains how to add variables to jobs you plan to run with dynamic workload broker.

When importing jobs from Tivoli Workload Scheduler, you can add variables to obtain higher flexibility for your job.

The variables are assigned a value when you submit the job in Tivoli Workload Scheduler. The supported Tivoli Workload Scheduler variables are as follows:

Table 1. Supported Tivoli Workload Scheduler variables in JSDL definitions.

Variables that can be inserted in the dynamic workload broker job definition	Description
tws.host.workstation	Name of the host workstation
tws.job.date	Date of the submitted job.

Table 1. Supported Tivoli Workload Scheduler variables in JSDL definitions. (continued)

Variables that can be inserted in the dynamic workload broker job definition	Description
tws.job.fqname	Fully qualified name of the job (UNISON_JOB)
tws.job.ia	Input arrival time of the job
tws.job.interactive	Job is interactive. Values can be `true` or `false`. Applies only to backward-compatible jobs.
tws.job.logon	Credentials of the user who runs the job (LOGIN). Applies only to backward-compatible jobs.
tws.job.name	Name of the submitted job
tws.job.num	UNISON_JOBNUM
tws.job.priority	Priority of the submitted job
tws.job.promoted	Job is promoted. Values can be `YES` or `No`. For more information about promotion for dynamic jobs, see the section about promoting jobs scheduled on dynamic pools in *Tivoli Workload Scheduler: User's Guide and Reference*.
tws.job.recnum	Record number of the job.
tws.job.resourcesForPromoted	Quantity of the required logical resources assigned on a dynamic pool to a promoted job. Values can be `1` if the job is promoted or `10` if the job is not promoted. For more information about promotion for dynamic jobs, see the section about promoting jobs scheduled on dynamic pools in *Tivoli Workload Scheduler: User's Guide and Reference*.
tws.job.taskstring	Task string of the submitted job. Applies only to backward-compatible jobs.
tws.job.workstation	Name of the workstation on which the job is defined
tws.jobstream.id	ID of the job stream that includes the job (UNISON_SCHED_ID)
tws.jobstream.name	Name of the job stream that includes the job (UNISON_SCHED)
tws.jobstream.workstation	Name of the workstation on which the job stream that includes the job is defined
tws.master.workstation	Name of the master domain manager (UNISON_MASTER)
tws.plan.date	Start date of the production plan (UNISON_SCHED_DATE)
tws.plan.date.epoch	Start date of the production plan, in epoch format (UNISON_SCHED_EPOCH)
tws.plan.runnumber	Run number of the production plan (UNISON_RUN)

If you want to create a dynamic workload broker job to be submitted from Tivoli Workload Scheduler, you can add one or more of the variables listed in Table 1 on page 1 in the **Variables** field of the **Overview** pane as well as in the **Script** field of the **Application** pane in the Job Brokering Definition Console.

If you plan to use the variables in a script, you also define the variables as environment variables in the **Environment Variables** field in the **Application** pane. Specify the Tivoli Workload Scheduler name of the variable as the variable value. You can find the Tivoli Workload Scheduler name of the variable in the **Variables inserted in the dynamic workload broker job definition** column.

You then create a Tivoli Workload Scheduler job which contains the name of the job definition, as explained in "Creating Tivoli Workload Scheduler jobs managed by dynamic workload broker" on page 1.

The following example illustrates a JSDL file with several of the supported Tivoli Workload Scheduler variables defined:

```
...<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod
/scheduling/1.0/jsdl" xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/
jsdle"
description="This jobs prints UNISON Variables received
from TWS in standard OutPut "
name="sampleUNISON_Variables">
  <jsdl:annotation>This jobs prints UNISON Variables
received from TWS in
standard OutPut </jsdl:annotation>
  <jsdl:variables>
    <jsdl:stringVariable name="tws.jobstream.name">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.job.fqname">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.master.workstation">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.plan.runnumber">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.plan.date">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.plan.date.epoch">none</jsdl:stringVariable>
    <jsdl:stringVariable name="tws.job.logon">none</jsdl:stringVariable>
  </jsdl:variables>
  <jsdl:application name="executable">
    <jsdle:executable output="${tws.plan.runnumber}">
      <jsdle:environment>
        <jsdle:variable name="UNISON_SCHED">${tws.jobstream.name}
</jsdle:variable>
        <jsdle:variable name="UNISON_JOB">${tws.job.fqname}
</jsdle:variable>
        <jsdle:variable name="UNISON_MASTER">${tws.master.workstation}
</jsdle:variable>
        <jsdle:variable name="UNISON_RUN">${tws.plan.runnumber}
</jsdle:variable>
        <jsdle:variable name="UNISON_SCHED_DATE">${tws.plan.date}
</jsdle:variable>
        <jsdle:variable name="UNISON_SCHED_EPOCH">${tws.plan.date.epoch}
</jsdle:variable>
        <jsdle:variable name="LOGIN">${tws.job.logon}
</jsdle:variable>
      </jsdle:environment>
...
```

Using variables in Workload Broker jobs

This section explains how to define and use variables in jobs for additional flexibility.

Dynamic workload broker supports the use of variables in jobs for additional flexibility. You can assign values to the variables or leave them blank, so that you can define the value when the job is submitted.

When you define jobs that will be processed through dynamic scheduling, you can include variables that can be used at run time to valorize or override the variables defined in the JSDL job definition.

You define the variables in the Task String section of the Tivoli Workload Scheduler job, as described in the following example:

```
jobName -var var1Name=var1Value,...,varNName=varNValue
```

To define variables in the Tivoli Workload Scheduler job, perform the following steps:

1. Create a JSDL job definition using the Job Brokering Definition Console.
2. Define the variables for the job. For example, you can define the **memory** variable to specify the amount of memory required for the job to run.
3. Move to the **Resources** tab, **Hardware Requirements** section and type the name of the variable in the **Exact** value field in the **Physical Memory** section. When the job is submitted, the value assigned to the **memory** variable defines the amount of physical memory.
4. Save the job definition in the **Job Repository** database.
5. Create a job to be submitted in Tivoli Workload Scheduler. This job contains the reference to the job in dynamic workload broker created in step 1. Define the Tivoli Workload Scheduler job as follows:
   a. In the Dynamic Workload Console, from the navigation bar, click **Administration** > **Workload Design** > **Manage Workload Definitions**.
   b. Select **New > Job Definition > Cloud > Workload Broker**.
   c. In the **General** tab, in the **Workstation** field specify the workload broker workstation.
   d. In the **Task** tab, in the **Workload Broker job name** field specify the name of the JSDL job definition you created in step 1.
6. Add the Tivoli Workload Scheduler job to a job stream.
7. Submit or schedule the Tivoli Workload Scheduler job using either the Dynamic Workload Console or **conman**.
8. After any existing dependencies are resolved, the master domain manager submits the Tivoli Workload Scheduler job to dynamic workload broker via the workload broker workstation.
9. The workload broker workstation identifies the job definition to be submitted based on the information on the **Task String** section of the Tivoli Workload Scheduler job. It also creates an alias which contains the association with the job.
10. The job definition is submitted to dynamic workload broker with the value specified for the **memory** variable.
11. Dynamic workload broker manages and monitors the whole job lifecycle.
12. Dynamic workload broker returns status information on the job to the workload broker workstation, which communicates it to the Master Domain Manager. The job status is mapped to the Tivoli Workload Scheduler status as described in Table 2 on page 6.

Defining affinity relationships

Affinity relationships cause jobs to run on the same resource. The resource on which the first job runs is chosen dynamically by dynamic workload broker, and the affine job or jobs run on the same resource.

In dynamic workload broker, you can define affinity relationships between two or more jobs when you want them to run on the same resource. When submitting the job from the Tivoli Workload Scheduler environment, you can define affinity that will be resolved by dynamic workload broker by adding an affinity definition to the **Task String** section of the Tivoli Workload Scheduler job in one of the following ways:

- Identifying affine job with the dynamic workload broker job ID
- Identifying affine job with the dynamic workload broker job alias
- Identifying affine job with the Tivoli Workload Scheduler job name

Identifying affine job with the dynamic workload broker job ID

jobName [-var *varName=varValue*,...,]-affinity jobid=*jobid*

Identifying affine job with the dynamic workload broker job alias

jobName [-var *varName=varValue*,...,]-affinity alias=*alias*

where

jobid Is the ID dynamic workload broker assigns when the job is submitted.

alias Is one of the following:

- The alias defined by the user at submission time for dynamic workload broker jobs.
- The alias automatically generated by the workload broker workstation when the job is submitted from Tivoli Workload Scheduler.

Identifying affine job with the Tivoli Workload Scheduler job name

The jobs must belong to the same job stream

jobName [-var *varName=varValue*,...,]-twsaffinity jobname=*twsJobName*

where

twsJobName
Is the name of the instance of the Tivoli Workload Scheduler job with which you want to establish an affinity relationship.

Alias definition in Tivoli Workload Scheduler

When the Tivoli Workload Scheduler job is submitted to the workload broker workstation, an alias is automatically generated. This alias uniquely identifies the job in the Tivoli Workload Scheduler environment and is displayed in the Dynamic Workload Console.

The syntax of the alias is as follows:

cpuschedname#schedname.jobname.**JNUM**-*nnnn*

where:

cpuschedname
Is the job stream workstation.

schedname
Is the job stream name

jobname
Is the job name.

JNUM Is the job number created by Tivoli Workload Scheduler before submitting the job. Identifies a specific instance of a job within a given production date.

Monitoring and canceling jobs

This section explains how you can monitor and cancel jobs using the Dynamic Workload Console or the **conman** command.

You can use the Dynamic Workload Console or the **conman** command line to monitor the status of submitted jobs, retrieve the job output, and cancel jobs if necessary, as you normally do in Tivoli Workload Scheduler. You can also use the Dynamic Workload Console to view their status since it provides more detail on jobs processed through dynamic workload broker.

Job statuses in dynamic workload broker correspond to the following statuses in Tivoli Workload Scheduler:

Table 2. Status mapping between dynamic workload broker and Tivoli Workload Scheduler

Dynamic workload broker job status	Tivoli Workload Scheduler job status
1. Run failed 2. Unable to start 3. Resource allocation failed 4. Unknown	1. ABEND 2. FAILED 3. FAILED 4. ABEND
1. Submitted 2. Submitted to Agent 3. Resource Allocation Received 4. Waiting for Reallocation 5. Waiting for Resources	1. INTRO 2. WAIT 3. WAIT 4. WAIT 5. WAIT
1. Running	1. EXEC
1. Completed Successfully	1. SUCC
1. Canceled 2. Cancel Pending 3. Cancel Allocation	1. ABEND 2. The status is updated when the job reaches the Canceled state in dynamic workload broker 3. The status is updated when the job reaches the Canceled state in dynamic workload broker

Note: The + flag written beside the INTRO and EXEC statuses means that the job is managed by the local **batchman** process.

You can view the job output by using both the Dynamic Workload Console or the **conman** command-line.

Consider the following Job Submission Description Language (JSDL) example, called BROKER_COMMAND_JOB:

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
                                name="BROKER_COMMAND_JOB">
 <jsdl:variables>
  <jsdl:stringVariable name="command">dir<jsdl:stringVariable>
  <jsdl:stringVariable name="params">d</jsdl:stringVariable>
 </jsdl:variables>
 <jsdl:application name="executable">
  <jsdle:executable interactive="false">
   <jsdle:script>${command} ${params}</jsdle:script>
  </jsdle:executable>
 </jsdl:application>
</jsdl:jobDefinition>
```

and the associated Tivoli Workload Scheduler job definition, called TWS_COMMAND_JOB:

```
 TTANCRED_BRK#TWS_COMMAND_JOB
 SCRIPTNAME "BROKER_COMMAND_JOB -var command=ping,
             params=ttancred.romelab.it.ibm.com"
 STREAMLOGON tws86master
 TASKTYPE BROKER
 RECOVERY STOP
```

The following example displays the output of the previous job, submitted from Tivoli Workload Scheduler to the workload broker workstation.

```
%sj TTANCRED_DWB#JOBS.TWS_COMMAND_JOB;std

================================================================================
= JOB       : TTANCRED_DWB#JOBS[(0000 10/28/12),(JOBS)].TWS_COMMAND_JOB
= USER      : twa86master
= TASK      : <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
                        name="BROKER_COMMAND_JOB">
 <jsdl:variables>
  <jsdl:stringVariable name="command">dir<jsdl:stringVariable>
  <jsdl:stringVariable name="params">d</jsdl:stringVariable>
 </jsdl:variables>
 <jsdl:application name="executable">
  <jsdle:executable interactive="false">
   <jsdle:script>${command} ${params}</jsdle:script>
  </jsdle:executable>
 </jsdl:application>
</jsdl:jobDefinition>
= AGENT     : TTAGENT
= Job Number: 318858480
= Wed Oct 24 15:58:24 CEST 2012
================================================================================
Pinging ttancred.romelab.it.ibm.com [9.168.101.100] with 32 bytes of data:

Reply from 9.168.101.100: bytes=32 time<1ms TTL=128
Reply from 9.168.101.100: bytes=32 time<1ms TTL=128
Reply from 9.168.101.100: bytes=32 time<1ms TTL=128
Reply from 9.168.101.100: bytes=32 time<1ms TTL=128

Ping statistics for 9.168.101.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

================================================================================
= Exit Status           : 0
= Elapsed Time (Minutes) : 1
= Job CPU usage (ms) : 1406
```

```
= Job Memory usage (kb) : 3940
= Wed Oct 24 15:58:27 CEST 2012
================================================================================
```

The keywords in the job output are as follows:

JOB Is the host name of the Tivoli Workload Scheduler agent to which the job has been submitted and the job name.

USER Is the Tivoli Workload Scheduler user who submitted the job to the workload broker workstation. When a scheduled job is submitted, the user name is retrieved from the **STREAMLOGON** keyword specified in the job definition. When an ad-hoc job is submitted from conman and the logon is not specified, the user name corresponds to the user who submitted the job.

TASK Is the full JSDL job submitted with all the variables substituted.

AGENT
Is the agent that run the job.

Job Number
Is the job number.

Exit Status
Is the status of the job when completed.

System Time
Is the time the kernel system spent for the job.

User Time
Is the time the system user spent for the job.

Note: The same job output format is used for JSDL template job types.

You can also kill the job after submitting it. Killing a job in Tivoli Workload Scheduler performs the same operations as issuing the **cancel** command in dynamic workload broker.

Chapter 2. Identifying the resources for jobs

To schedule jobs, dynamic workload broker first scans the computers in the environment to retrieve hardware and operating system information from the agent computers. You can also optionally create logical resources that represent characteristics of the computers that are not gathered by the scan, such as software licenses or installed applications, to further identify resources available in the environment.

After the Tivoli Workload Scheduler agent is installed, an automatic scan is performed on the discovered computers where the agent is installed. The scan returns hardware and operating system information which is stored in the **Resource Repository**.

The hardware and operating system information returned by the scan is considered a physical resource. Physical resources collected from the agent computers include the following:

Types of physical resources	Examples
Computer system	Computer system name, model, number of processors, CPU speed
Operating system	Operating system type and version, virtual memory, physical memory, swap space
Network system	IP address, network card, host name
File system	File system storage capacity

The automatic discovery process of gathering physical resource information is capable of identifying available computers with the resources required for jobs to run on. The scan is scheduled and configurable. You can configure the scan from the `ResourceAdvisorAgentConfig.properties` file on the master domain manager and from the `JobManager.ini` file on the agents. Ensure the scan runs at regular times to update any changes to resources. Refer to the *Tivoli Workload Scheduler: Administration Guide*, *SC23-9113* for more information about this configuration file.

When the physical resources gathered by the scan do not supply enough information to accurately address specific job requirements, you can define logical resources or resource groups using the Dynamic Workload Console. The Dynamic Workload Console gives you the capability to set up additional logical resources and link them to computers where the resources are available. Logical resources help identify resources required by jobs to make allocation more accurate. Logical resources can also be used when expressing a consumable quantity of a resource. For example, you might use a logical resource to identify specific applications, and you might also use them to define a limited number of software licenses available. When a job is submitted, the job definition includes the physical and logical resource requirements, and with this information dynamic workload broker finds the most suitable computer.

You can also use the Dynamic Workload Console to define a resource group. A resource group is a combination of physical and logical resources defined to accurately match and allocate resources to job requirements when the job is submitted. After creating logical resources and resource groups, you can

subsequently edit them using the Dynamic Workload Console. If a computer, logical resource, or resource group becomes unavailable or you need to make it unavailable to perform maintenance tasks, for example, you can set the status to offline. You can subsequently set the status online using the Dynamic Workload Console.

The following are tasks for configuring resources:

- "Creating logical resources" on page 12
- "Creating resource groups" on page 14

Checking physical resources on computers

You can browse and display the physical resources discovered by the agent scan on the computers in your environment.

The automatically scheduled scan that runs on computers when the Tivoli Workload Scheduler agent is installed returns hardware and operating system information from the agent computers to the Resource Repository. You can use the Dynamic Workload Console to view the information collected by the agent, in addition to the other information about the computers. The following information can be accessed:

- Operating system information
- Computer availability
- Processor information
- Machine information
- Free memory, free virtual memory, and free swap space
- System resources allocated to jobs
- A history of job instances that ran on the computers and are currently running

To view information about the physical resources available on the computers in your environment, do the following:

1. In the Tivoli Dynamic Workload Broker navigation tree, expand **System Configuration** and click **Set Broker Server Connections**
2. Define viable connections, test, and save them .
3. Expand **System Status and Health** and click **Monitor Computers on Broker**.
4. Specify the search criteria for the computers you want to find.
5. Click **Search**. The computers that meet the search criteria are displayed in the **Monitor Computers on Broker** page.
6. To view the physical resources for a computer, select the display name link for the computer. The **Computer details** page displays the physical resources on the computer.

Figure 1 on page 11 shows the **Monitor Computers on Broker** page. From this view you can perform the following tasks:

- Set computers offline so that they cannot be allocated to run jobs.
- Set offline computers back online so that they can be allocated to run jobs.
- Delete computers so that they are no longer visible when you search for computers. When you delete a computer, it is temporarily removed from the database for a period of time defined in the ResourceAdvisorAgentConfig.properties file. After the deletion, the Tivoli Workload Scheduler agent remains installed and running. Any jobs currently

allocated and running on the computer complete. To permanently delete a computer, you must uninstall the Tivoli Workload Scheduler agent.

- Refresh the view of the computer search results to see updated information about computers.
- View the number of jobs currently allocated on a given computer in the **Active Jobs** column. For each computer this column shows the number of jobs that have selected the computer as a target system, as well as the number of jobs that are currently allocating the computer as a related resource. In the specific case you defined a computer system as a type for a related resource required to run a job (in the JSDL definition of the job), when the job is allocated it is displayed twice in **Active Jobs** as follows:
  - If the same computer is selected both as a target system and as a related resource, the column shows 2 jobs for that computer, even though there is only one job running.
  - If different computers are selected for the target system and the related resource, the column shows the same job twice (once for each computer).
- View additional information about a computer.

To perform these tasks, do the following:

1. Select a computer in the **Monitor Computers on Broker** table.
2. Select one of the following operations from the **Actions** menu:
   - **Set as online**
   - **Set as offline**
   - **Delete**
   - **Refresh**
3. Click **Go** to perform the operation.

You can display details on computers by clicking the computer name link in the **Monitor Computers on Broker** table.

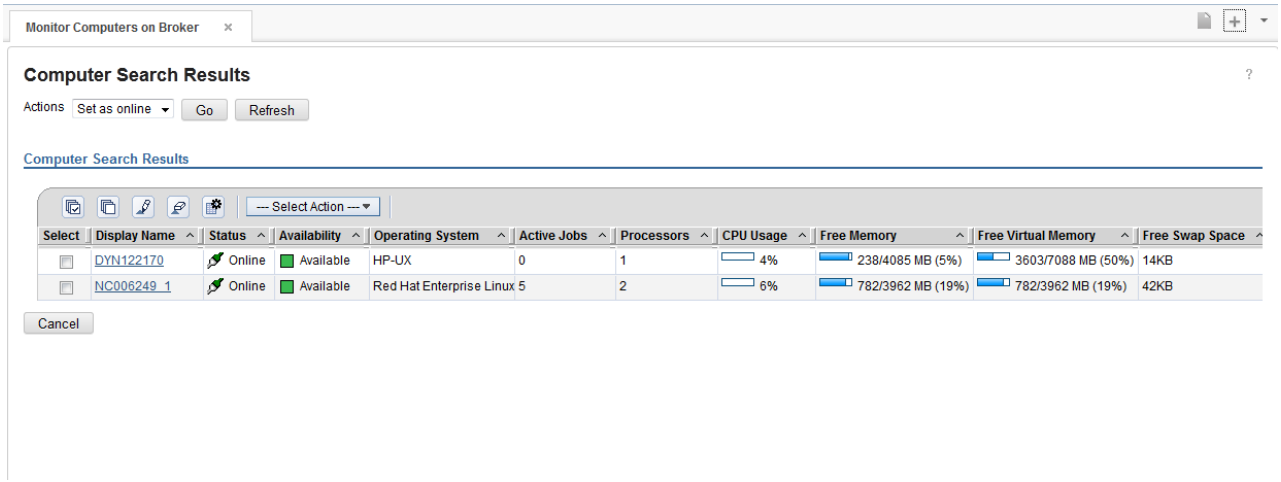

Figure 1. Computer Search Results page

In general, you can click links for job names, job instance names, and computers from the Dynamic Workload Console to display more details about them.

Creating logical resources

Using the Dynamic Workload Console, you can define logical resources and resource groups for workstations with properties that are not discovered with a system scan. You create a logical resource specifying the characteristics of the resource that are required to run jobs.

To create a logical resource, perform the following steps:

1. In the Tivoli Dynamic Workload Broker navigation tree, expand **Administration** and click **Create Logical Resources on Broker**. The wizard helps you create a new resource and add it to computers in your environment.
2. In the **General Properties** page, define the general properties for the logical resource:
   a. In the **Name** field, type the name to be assigned to the logical resource. This field is required. The name must start with an alphabetical character, and can contain underscore symbols (_), minus symbols (-), and periods (.). Spaces, special characters, accented characters are not supported.
   b. In the **Type** field, type a meaningful name for the logical resource type. For example, if the logical resource describes DB2® applications, you might call the resource DB2. The name must start with an alphabetical character, and can contain underscore symbols (_), minus symbols (-), and periods (.). Spaces, special characters, accented characters are not supported.
   c. In the **Quantity** field, specify a value that represents the availability of the logical resource. For example, if the resource consists in a license server, you can specify the number of licenses available on the server for a product. This field is optional.
   d. Select the **Set as Offline** check box to mark the resource as not available. You can subsequently change the resource status by expanding **Scheduling Environment** in the left pane and selecting **Logical Resources**. You can then search for a resource and modify the status.
   e. Click the **Next** button to proceed.
3. In the **Computer Search Criteria** page, specify the criteria for searching the computers to be added to the logical resource. In this page, you can either perform a search on all computers available in the dynamic workload broker environment or you can search for specific computers. To search on all available computers, leave all fields blank. As an alternative, you can specify one or more of the search criteria listed below. The search criteria are cumulative; each additional information you specify further refines the search:
   - In the **Host Name** field, specify the host name of a computer. Wildcards are supported. The search is performed on computers with the specified host name or section of the host name. For example, if you enter test* in the **Host name** field, the test1 and testing host names are returned.
   - In the **Logical resources already on the computer** area, specify the name and type of logical resources already present on the computer, if any. In the **Name** field, specify the logical resource name and in the **Type** field specify the logical resource type. Wildcards are supported.
   - In the **Availability** area, specify whether the computer is:

     Available
     The computer is available and can be assigned jobs.

     Unavailable
     The computer is not available. The network might be down or the computer might be switched off.

- In the **Status** area, define the status of the specified computer.

  Online
  The computer is online.

  Offline
  The computer is offline. The administrator might have set the computer offline for maintenance purposes.
- In the **Hardware Characteristics** area, specify the number of processors available on the computer:

  Single processor
  The computer contains one processor.

  Double processor
  The computer contains two processors.

  Multi processor
  The computer contains three or more processors.
- In the **Operating System** area, specify the operating system installed on the computers for which you want to search. The search is performed only for computers with the selected operating systems. Available operating systems are:
  - **Windows**
  - **Linux**
  - **AIX®**
  - **Oracle Solaris**
  - **HP-UX**
  - **zOS**
  - **IBM i**

  The results of the search are displayed in the **Computer Search Results** pages.

4. In the **Computer Search Results** page, you specify to which computers you want to add the logical resource you are creating. Your selections are displayed in the **Summary** page.
5. In the **Summary** page, you can optionally remove the selected computer from the logical resource you are defining. Click **Finish** to save the logical resource.

The logical resource has been created and can be accessed using the **System Status and Health** > **Monitor Logical Resources on Broker** task. You can perform the following operations using this task:

- Set the online or offline status of the logical resource.
- Delete the logical resource.
- Edit the logical resource specifications, including the computers where the logical resource is located, the online or offline status of the computers, and the logical resource name, unless the logical resource was imported from the Change and Configuration Management Database. A Change and Configuration Management Database logical resource can be identified in the table of logical resources by the value `CCMDB` in the **Owner** column.

Creating resource groups

Using the Dynamic Workload Console, you can create resource groups to group computers, logical resources, or both. A resource group represents a logical association between computers, logical resources, or both with similar hardware or software characteristics. It is a combination of physical and logical resources to accurately match and allocate resources to job requirements when the job is submitted.

To create a resource group, perform the following steps:

1. In the console navigation tree, expand **Administration** and click **Create Resource Groups on Broker**. The wizard helps you create a new resource group.
2. In the **Group Type Selection** page, specify a name, the status, and a type for the resource group:
   a. In the **Name** field, type the name to be assigned to the resource group. The name must start with an alphabetical character, and can contain underscore symbols (_), minus symbols (-), and periods (.). Spaces, special characters, and accented characters are not supported. This field is required.
   b. Select the **Set as Offline** check box to mark the resource group as not available. You can subsequently change the resource group status by expanding **Scheduling Environment** in the left pane and selecting **Resource Groups**. You can then search for a resource group and modify the status.
   c. In the **Select items to be grouped** area, select the elements the group consists of. Supported values are computers, logical resources or both. This field is required.
   d. Click the **Next** button to proceed.
3. In the **Computer Search Criteria** page, specify the criteria for searching available computers to be added to the resource group. In this page, you can either perform a search on all computers available in the dynamic workload broker environment or you can search for specific computers. To search on all available computers, leave all fields blank. As an alternative, you can specify one or more of the search criteria listed below. The search criteria are cumulative; each additional information you specify further refines the search:
   - In the **Host Name** field, specify the host name of a computer. Wildcards are supported. The search is performed on computers with the specified host name or section of the host name. For example, if you enter test* in the **Host name** field, the test1 and testing host names are returned.
   - In the **Logical resources already on the computer** area, specify the name and type of logical resources already present on the computer, if any. In the **Name** field, specify the logical resource name and in the **Type** field specify the logical resource type. The name and type must start with an alphabetical character, and can contain underscore symbols (_), minus symbols (-), or periods (.). Spaces, special characters, and accented characters are not supported.
   - In the **Availability** area, specify whether the computer is:

     Available
     The computer is available to be allocated.

     Unavailable
     The computer is not available. The network might be down or the computer might be switched off.
   - In the **Status** area, define the status of the specified computer.

Online
The computer is online.

Offline
The computer is offline. The administrator might have set the computer offline for maintenance purposes.

- In the **Hardware Characteristics** area, specify the number of processors available on the computer:

  Single processor
  The computer contains one processor.

  Double processor
  The computer contains two processors.

  Multi processor
  The computer contains three or more processors.

- In the **Operating System** area, specify the operating system installed on the computers for which you want to search. The search is performed only for computers with the selected operating systems. Available operating systems are:
  - **Windows**
  - **Linux**
  - **AIX**
  - **Oracle Solaris**
  - **HP-UX**
  - **zOS**
  - **IBM i**

Click **Next** to perform the search based on the criteria specified. The results of the search are displayed in the **Computer Search Result** page.

4. In the **Computer Search Result** page, you specify to which computers you want to add the group you are creating. Click **Next**.
5. If the resource group you are defining includes a logical resource, then the Logical Resource Search Criteria page prompts you to specify the following search criteria:
   a. The name of the logical resource.
   b. The type of logical resource.

   Click **Next** to display the **Logical Resources Search Results** page.
6. Select the logical resource to add to the resource group you are defining and click **Next** to display the **Summary** page.
7. In the **Summary** page, you can optionally remove any computer or logical resource that you included in the resource group. Click **Finish** to save the resource group.

The resource group has been created and can be accessed using the **System Status and Health** > **Monitor Resource Groups on Broker** task. You can perform the following operations using this task:

- Set the online or offline status of the resource group.
- Delete the resource group.
- Edit the resource group specifications, including adding and removing computers and logical resources from the group, changing their online or offline status, and changing the resource group name.

Chapter 3. Writing JSDL definitions with the Job Brokering Definition Console

The Job Brokering Definition Console provides an easy-to-use graphical interface that helps you create and edit JSDL job definitions for use with dynamic workload broker.

The Job Brokering Definition Console graphical interface helps you create and edit job definitions based on the Job Submission Definition Language (JSDL) schema. Each text field in the Job Brokering Definition Console corresponds to an element or attribute in the JSDL file and vice versa. You can use the Job Brokering Definition Console to create existing job types. If you need to create job types with advanced options, use the Dynamic Workload Console or **composer** command.

The Job Brokering Definition Console simplifies the task of creating a JSDL file by hiding the complexity of the file itself and validating the file structure against the JSDL schema. Information defined in the Job Brokering Definition Console is automatically converted to the corresponding element or attribute in the JSDL file.

You can save a JSDL file locally or upload it as a job definition in the **Job Repository** where it becomes available for submission. When you save the file in the Job Brokering Definition Console, the JSDL file is checked against an .xsd file provided with the product installation which contains the syntax rules. A message is displayed if a syntax error is encountered in the JSDL file, allowing you to correct the error.

The Job Submission Description Language (JSDL) is an XML-based language used for specifying the characteristics and requirements of a job, as well as instructions on how to manage and run it. These include the following:

- Job identification information
- Program running information
- Resource requirements
- Scheduling and running requirements
- Resource quantity to be allocated or required
- Logical allocation of the resource quantity

Selecting target types

When creating a JSDL file, you can choose between the following resource types as targets for your job:

Resources
A resource is a computer system. You can use this resource type to define a basic requirement for your job.

Related resources
A related resource is a set of resource types. You can use this resource type to define a basic requirement for your job. A related resource includes the following resource types:

- A set of hardware and software properties of a computer such as operating system, file system and network system.

- Logical resources and logical entities that can be associated to one or more computers to represent applications, groups, licenses, servers and so on.

Related resources have two main functions:

- You can specify related resources as an additional requirement adding to the resource requirement. In this case, you must create a relationship between the resource and the related resource.
- You can use the related resource to indicate that the presence of a certain resource in your environment is a co-requisite for running the job. In this case, you must not create a relationship between the resource and the related resource. A related resource having no relation to a resource is a global resource. For example, if you want to move a file from resource A to resource B, resource B is a co-requisite for running the job which moves the file. Computers can only be defined as global resources.

Selecting resource types

Dynamic workload broker manages the resource types listed in Table 3. For each resource type, you can specify requirements on the properties listed in the **Available properties** column. Table 3 also lists consumable properties and properties that can be optimized. Consumable properties can be allocated exclusively to the job while it runs using the allocation mechanism. Properties that can be optimized can be used to provide a more effective load balancing on the resource property.

Table 3. Resource types and properties

Resource Type	Available properties	Is consumable	Can be optimized	Supports wildcards
ComputerSystem	CPUUtilization	No	Yes	No
	HostName	No	No	Yes
	Manufacturer	No	No	Yes
	Model	No	No	Yes
	NumOfProcessors	Yes	Yes	No
	ProcessingSpeed	No	Yes	No
	ProcessorType	No	No	No
LogicalResource	DisplayName	No	No	Yes
	SubType	No	No	Yes
	Quantity	Yes	Yes	No
OperatingSystem	DisplayName	No	No	Yes
	FreePhysicalMemory	No	Yes	No
	FreeSwapSpace	No	Yes	No
	FreeVirtualMemory	No	Yes	No
	OperatingSystemType	No	No	No
	OperatingSystem Version	No	No	No
	TotalPhysicalMemory	Yes	Yes	No
	TotalSwapSpace	Yes	Yes	No
	TotalVirtualMemory	Yes	Yes	No

Table 3. Resource types and properties (continued)

Resource Type	Available properties	Is consumable	Can be optimized	Supports wildcards
FileSystem	DisplayName	No	No	Yes
	FileSystemRoot	No	No	Yes
	FileSystemType	No	No	No
	FreeStorageCapacity	No	Yes	No
	TotalStorageCapacity	Yes	Yes	No
NetworkSystem	NetworkAddress	No	No	No
	NetworkSystem HostName	No	No	Yes

When you define the requirements for a job definition, you can define the amount of a consumable property which will be allocated to the job. When a resource property is allocated to a job, the amount you specify is logically reserved for the job. If another job is submitted which allocates a value greater than the remaining capacity of the same consumable property, this job cannot run on the same resource as the previous job because the required property is already reserved. If no property allocation is specified in the job definition, the job can run on the same resource as the previous job because the allocation mechanism applies only if both jobs allocate the same property.

You can use the allocation mechanism to limit concurrent use of the same quantity by several jobs and improve system performance.

To allocate a property for a job, use the **allocation** element in the JSDL file or the Software Requirements and Hardware Requirements tabs in the Job Brokering Definition Console.

This allocation type applies to computer systems. To allocate a property for a resource other than a computer system, you define the resource whose property you want to allocate in the Related resource pane and define the allocation setting for one or more of its properties. You then define a relationship between the resource and the related resource you created. In this way you define the related resource and the allocated property as a requirement for the job to run.

Job definitions

This topic provides an overview of the possible content of job definitions and describes how the different types of job definition content are added using the Job Brokering Definition Console.

A job definition contains all the information required to determine the computer system or systems on which a job could run, any scheduling and job balancing rules that are to be applied when allocating resources, as well as the information required to identify and run the application. It is defined using the Job Submission Description Language (JSDL).

JSDL is an XML-based language used for specifying the characteristics and requirements of a job, as well as instructions on how to manage and run the jobs. A JSDL file can include the following types of information

Basic job information
Includes the job name, any job categories to which you want to assign it, and any variables that are used in the job.

Variables can be used in several ways in a job definition. For example:

- A set of variables can describe a command and its arguments. These can be added to program running information and within the script of the job.
- Variables can also be used to identify resources, for example, a target host.

The default value assigned to the variable in the job definition is used when the job is run, unless it is overridden at submission time. See "Using variables in job definitions" on page 27.

Program running information
Identifies the script or executable to be run and, if necessary, standard input, output, and error files and the working directory. If the job needs to provide credentials these can also be specified.

You can define the required credentials to run a job if the credentials are different from those under which the Tivoli Workload Scheduler agent runs.

On Windows targets, jobs with no specified credentials, run under the user account specified during the Tivoli Workload Scheduler agent installation, unless the agent runs under the Local System account. In this case, any job submitted to the agent runs under the default administrator's account.

On UNIX targets, jobs with no specified credentials, run under `root`.

Required resource specifications
Enables dynamic workload broker to identify the computer systems on which the job can run based on hardware and software requirements.

Related requirements
Allow you to specify required relationships between resources and co-requisite resources for a job.

Allocation
Resource quantity to be allocated or required.

Optimization and load-balancing policies.

The following load balancing policies are available:

Balance load between resources by number of jobs running
Jobs are assigned to targets based on the number of jobs currently running on each target. The objective is to ensure that each resource runs the same number of jobs during the same time interval. This policy is the default. It is suitable for situations where many similar jobs, which consume similar quantities of system resources, are to be run on a set of resources

Balance load between resources by optimization objective
You define an objective by selecting a resource type and related resource property and specifying the objective to maximize or minimize the property. For example, you could balance load with the aim of keeping the amount of free physical memory available on operating system resources as high as possible. The objective would be set to maximize free physical memory and when jobs with this objective are submitted, they are allocated to available

resources so that more jobs go to the resources that currently have the greatest amount of free physical memory.

Select best resource by optimization objective
You define the optimization objective in exactly the same way as described for the **Balance load between resources by optimization objective**. However, when a job with this policy is submitted, it would always be assigned to the resource that best matched the objective. For example, if the objective is to maximize free physical memory, the job would run on the resource that had the highest amount of free physical memory at submission time.

Enterprise Workload Manager
If you have Enterprise Workload Manager installed, you can define jobs with an optimization policy to use the load-balancing capabilities of this product.

In the JSDL schema, the **Optimization** page corresponds to the **optimization** element.

Scheduling and running requirements.
Allows you to define a priority, the time a job can wait for resources before failing, and recovery actions in the event of a failure.

The maximum priority is 100 and priority settings between 90 and 100 should only be used for critical jobs. Jobs with these priorities are always allocated resources ahead of other waiting jobs regardless of how long the other jobs have been waiting. At lower priorities than 90, jobs are allocated resources based on the priority setting and the age of the job. As time passes, jobs with a low priority setting increase their priority so that they eventually are allocated resources even if jobs with higher initial priorities are waiting.

The Job Brokering Definition Console graphical interface allows you to create and edit job definitions based on the JSDL schema. Fields in the Job Brokering Definition Console correspond to elements in the JSDL schema. When creating a job definition using the Job Brokering Definition Console, you can view the job definition structure in the **Outline** pane.

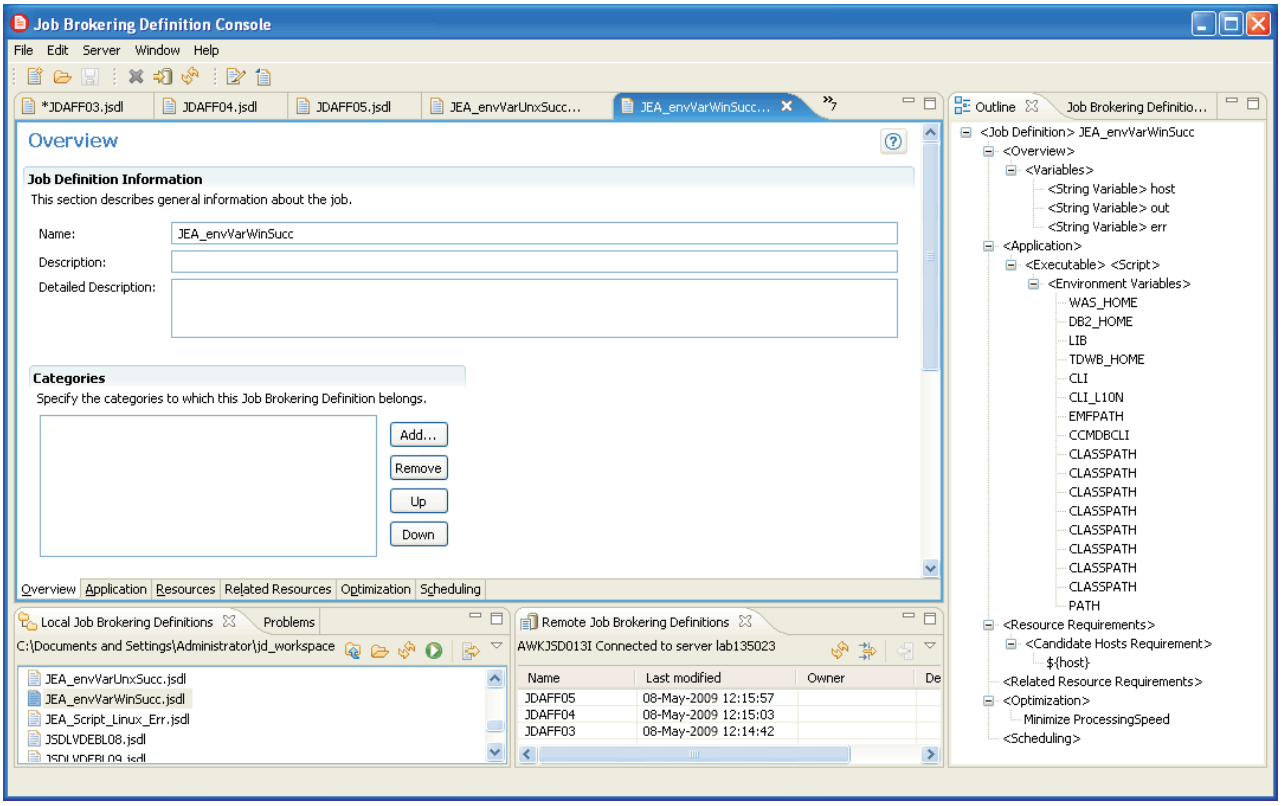

Figure 2. Job Brokering Definition Console main page

The JSDL schema offers great flexibility in defining jobs and their requirements. A job can have a very open definition, with few defined requirements, allowing it to run on a wide range of resources and to follow default rules for load balancing. Other jobs could have a much more detailed set of hardware and software requirements, as well as specific resource allocations and a load balancing policy. Using the graphical interface simplifies the task of creating JSDL files and eliminates many of the risks of error that occur when the files are edited manually. The different elements that make up a job definition are available, in many cases with a set of fixed values from which you can choose. Information defined in the Job Brokering Definition Console is validated, ensuring that any values you have entered are correct and consistent with each other.

In addition, the Job Brokering Definition Console also includes content assistance that provides server-side values for several fields on the interface, for example, candidate host names and logical resources, to name a few. Fields with content assistance are identified by a light-bulb icon next to the field. Position your mouse over the light-bulb and press **Ctrl + Space** to display a list of possible values. Server-side values are populated using the server cache for the currently active server connection. Server data is cached automatically when the initial connection to a server is made or each time the server connection is changed. You can refresh the cache at any time, for example, if you have defined a new resource requirement on the server, by selecting **Server** > **Refresh Server Data Cache**.

When you save the file in the Job Brokering Definition Console, the JSDL file is checked against an .xsd file provided with the product installation which contains the syntax rules. A message is displayed if a syntax error is encountered in the

JSDL file, allowing you to correct the error. You can save the JSDL files locally or upload them as job definitions in the **Job Repository** where they become available for submission.

Resources in the job definition

This topic provides an overview of how resources and their properties are used in the job definition to identify possible targets, to reserve allocations of consumable resources, and to optimize load balancing between available resources.

An understanding of physical and logical resources and their properties is the key to creating a job definition that accurately targets suitable resources for running the job, determines the resource allocation requirement, and contributes to balancing the load between available resources. Each resource has one or more properties associated with it. Properties can have the following characteristics:

Is consumable
Properties of resources that are consumable have finite amount associated with them which can be consumed by the jobs that are allocated to the resource. For example, a computer system has a finite number of processors.

Can be optimized
Some properties can be used to define optimization objectives, which determine how load is to be balanced when jobs are allocated to a group of resources. For example, you could choose to allocate a job to the matching resource that has the lowest CPU usage.

Supports wildcards
Some properties can be specified in the job definition using wildcards. For example, a requirement for a particular series of computer models could be defined by specifying the model using wildcards.

Table 4 shows the different resource types that can be included in a job definition and their available properties.

Table 4. Resource types and properties

Resource Type	Available properties	Is consumable	Can be optimized	Supports wildcards
ComputerSystem	CPUUtilization	No	Yes	No
	HostName	No	No	Yes
	Manufacturer	No	No	Yes
	Model	No	No	Yes
	NumOfProcessors	Yes	Yes	No
	ProcessingSpeed	No	Yes	No
	ProcessorType	No	No	No
LogicalResource	DisplayName	No	No	Yes
	SubType	No	No	Yes
	Quantity	Yes	Yes	No

Table 4. Resource types and properties (continued)

Resource Type	Available properties	Is consumable	Can be optimized	Supports wildcards
OperatingSystem	DisplayName	No	No	Yes
	FreePhysicalMemory	No	Yes	No
	FreeSwapSpace	No	Yes	No
	FreeVirtualMemory	No	Yes	No
	OperatingSystemType	No	No	No
	OperatingSystem Version	No	No	No
	TotalPhysicalMemory	Yes	Yes	No
	TotalSwapSpace	Yes	Yes	No
	TotalVirtualMemory	Yes	Yes	No
FileSystem	DisplayName	No	No	Yes
	FileSystemRoot	No	No	Yes
	FileSystemType	No	No	No
	FreeStorageCapacity	No	Yes	No
	TotalStorageCapacity	Yes	Yes	No
NetworkSystem	NetworkAddress	No	No	No
	NetworkSystem HostName	No	No	Yes

Resource properties can be used in the job definition in the following ways:

Identifying targets for the job

On the **Resources** page of the Job Brokering Definition Console, you can supply information about the resources required for the job. Using this information, dynamic workload broker can identify the computer systems on which the job could run. In addition to the basic hardware and software requirements, you can use the **Advanced Requirements** tab to include requirements for specific resource properties. For example, you can add a requirement for a specific processor type or specify a required range of processor speeds. In the JSDL schema, the **Resources** page corresponds to the **resources** element.

When you define a resource requirement, the underlying relationship between the required resource and the computer system which contains the resource is automatically created by the Job Brokering Definition Console to facilitate the usage of the product.

Resource property requirements to be used when identifying targets for job can also be specified on the **Related Resources** page. A related resource includes the following resource types:

- A set of hardware and software properties of a computer such as operating system, file system, and network system.
- Logical resources, which are a flexible means of providing information about your environment in addition to the information collected by the hardware scan. For example, you could create logical resources to represent applications, groups, licenses, or database access. A logical resource can be linked to one or more specified computers or it can be a freestanding global resource, available to all computers.

Related resources have two main functions:

To specify additional requirements, making the matching criteria for possible targets more precise

Targets can only match if they either contain or are associated with the specified resource. In addition to defining the related resource in the job definition, you must also define its relationship to the target resource and specify the relationship type as **contains** or **associates with**. Related resource that define hardware and software properties always have a **contains** relationship while logical resources often have an **associates with** relationship. For example, if a related requirement for a logical resource that represents a node-locked license is included, the target system must be one of those that is associated with this resource, and therefore a target where the license is available.

To specify global resources that must be available for the job to run

These related resources are not related to the target resource and have no role in finding matching resources for the job to run on. The resource must be available to the job at submission time. For example, if a license required to run the software used by the job is of a type that is not assigned to any computer, a logical resource could be created to identify it and to track the number of licenses that exist and that are in use. No computers are associated with this logical resource and so it is referred to as a global resource, available to all computers. The job definition includes a related resource identifying the floating license logical resource and the number of licenses required. Before the job can run, it must be possible to meet this requirement.

In the JSDL schema, the **Related Resources** page corresponds to the **relatedResources** element.

When the resource requirements for the job are defined, logical rules are applied to determine whether the requirements are alternatives to each other (OR) or whether they are inclusive (AND). In general, the different types of requirements have an AND relationship, for example, if you specify an operating system type, CPU architecture, and a value for minimum physical memory, the target resource for the job must meet all of these requirements.

Within the following requirement types, you can specify alternatives that have an OR relationship:

- Candidate hosts
- Candidate CPU architectures
- Candidate operating systems

If several entries are added for any of these requirement types, they are considered as alternatives. For example, if Linux, AIX, and HP-UX are specified as candidate operating systems, the target resource for the job must have one of these operating system types.

Within the following requirement types, all requirements specified must be met by the target resource for the job.

- Logical resources
- File systems

For example, if you add Local Disk and CD ROM to the File system requirements, the target resource for the job must have both a local disk and a CD ROM.

Reserving resources

When defining the requirements for a job definition, you can define the amount of a consumable property which will be allocated to the job. When a resource property is allocated to a job, the amount you specify is logically reserved for the job. If another job is submitted which allocates a value greater than the remaining capacity of the same consumable property, this job cannot run on the same resource as the previous job because the required property is already reserved. If no property allocation is specified in the job definition, the job can run on the same resource as the previous job because the allocation mechanism applies only if both jobs allocate the same property.

You can use the allocation mechanism to limit concurrent use of the same quantity by several jobs and improve system performance.

On the Job Brokering Definition Console, you can allocate a specified quantity of a consumable property. You can use the allocation pane from the **Advanced Requirements** tab of the **Resources** page or you can define a required resource and property in the **Related resource** page and specify the amount of the property to be allocated. From the **Advanced Requirements** tab on the **Resources** page, you can only allocate consumable properties of computer system resources.

Defining load-balancing policies

You can use the **Optimization** page in the Job Brokering Definition Console to define custom rules for load-balancing to be applied when the job is submitted. The default method of load-balancing is to aim to equalize the number of jobs running on each resource.

Dynamic workload broker provides two types of optimization policy types that use rules based on resource properties:

- Balance load between resources by optimization objective
- Select best resource by optimization objective

For both policies, you define an objective to distribute jobs minimizing or maximizing the property of a computer system, a file system, a logical resource, or an operating system. For example, you could balance loads with the aim of keeping the free physical memory available on operating system resources as high as possible.

When the **Balance load between resources by optimization objective policy** is used, jobs are distributed between matching resources based on a statistical probability that the resource currently has highest amount of free physical memory of all matching resources. When the **Select best resource by optimization objective policy** is used, the job is allocated to the resource that has the highest amount of free physical memory.

When defining an objective, you must select a resource that is included in the job definition as part of the identification of targets for the job. For example, if you want to define the objective to minimize free physical memory, at least one operating system requirement must be included in the job definition. This could be candidate operating systems, a physical or virtual memory requirement, or a related requirement involving operating system properties. Computer system properties are the exception to this

rule. Optimization objectives using computer system properties can be always defined even if the job definition includes no explicit computer system requirements.

For information about all available load balancing policies, see "Job definitions" on page 19.

Using variables in job definitions

This section explains how to use variables to add flexibility to the job definitions.

There are two types of variables in a job definition:

Job variables

There are three types of job variables: String, Double, Integer. You can define job variables in your job definition that are resolved or overwritten at job submission time. This enables the job definition to be used for different situations by specifying different values for the variable at submission time. You define variables in the variable element, but you can refer to the variable from several other elements.

You define variables and assign them values from the **Overview** page on the Job Brokering Definition Console. Job variables are referenced in the job definition in the format ${*variable_name*}. For example, to use a variable to set the minimum amount of physical memory required for a job to 512 MB, do the following:

1. In the **Variables** pane of the **Overview** page, add the string variable `memory` and assign it a value of `512`.
2. On the **Hardware Requirements** tab of the **Resources** page, select **Range value** for Physical memory and set the Minimum value to `${memory}`.

When jobs are submitted, using Dynamic Workload Console, Tivoli Workload Scheduler Task field, or the dynamic workload broker CLI, default values for variables defined in the job definition can be overridden and new variables can be added.

Environment variables

Environment variables are set in the run time environment for the dynamic workload broker job definition. Environment variables can be used to change the run time environment for the job on the assigned resource. This enables you to change only the values of the environment variables when you change the resources for the job definition. Environment variables are referenced in the job definition in the format $*variable_name* where *variable_name* is the name of the environment variable.

Environment variable values cannot be set or overwritten when the job is submitted.

Using JSDL job definition templates

Use job definition templates to be able to run multiple jobs based on a single JSDL document, or to turn a traditional job into a *dynamic* job without the need to create a specific JSDL definition for it.

You have two options for writing the JSDL job definitions for the workload you want to submit with dynamic workload broker:

- Writing a separate definition for each job

- Writing a generalized definition that you can use as a template to run more jobs

Writing and using templates is an option that lets you reuse the same JSDL document on multiple jobs when they use the same resources and candidate hosts and share similar scheduling and optimization preferences. This requires that you also define an extended agent workstation for each template you implement, so that at runtime the JSDL template can be properly identified by selecting the extended agent on which the job you want to run is defined. In this way, you can make up *classes* of jobs where all the jobs that belong to the same class are defined to run on the same extended agent and therefore select, by means of the workload broker workstation, the same JSDL document to submit to the broker.

Traditional Tivoli Workload Scheduler jobs can be routed to dynamic workload broker by simply changing their CPU to an appropriate extended agent, without changing the job definition and without requiring a different JSDL definition for each job. This is the recommended way for changing static workload into dynamic workload in Tivoli Workload Scheduler.

Writing a JSDL job definition template

Specific, prepackaged JSDL templates that you can fill in do not exist. Rather, you work a number of steps so that you can write in the Job Brokering Definition Console a JSDL file that can be referenced by more Tivoli Workload Scheduler job definitions.

To write a template you use the following:

- The `composer` command line or the Dynamic Workload Console to define extended agents (with their access method) and to create or modify job definitions in Tivoli Workload Scheduler.
- The Job Brokering Definition Console to write the JSDL file that you then use as a template.

The steps are:

1. In the Job Brokering Definition Console, you create a JSDL document, give it a name, and save it in the Job Repository of dynamic workload broker. Like for regular job definitions, fill in the data throughout the pages of the Job Brokering Definition Console, specifying the required resources, and optimization and scheduling details. Unlike you do in regular job definitions, in the Application page, after setting the `Type` to `Executable` (or to `Extended Job`), specify the following variable name in the `Script` (or `Task string`) field:

   ```
   ${tws.job.taskstring}
   ```

2. With `composer` or the Dynamic Workload Console define a workstation of type `extended agent` hosted by the workload broker workstation.

   If you need background information about extended agents, see the *Tivoli Workload Scheduler: User's Guide and Reference*, *SC32-1274*. For the purpose of creating the template, however, you only need to know the following facts about an extended agent:

   - It is a logical definition that must be hosted by a physical workstation. In this case the physical workstation must always be the workload broker workstation. This workstation can host as many extended agents as you need.
   - It requires an access method. An access method can be a complex program, but in this case it is only a statement that references the name of the JSDL

file that will be your template. The access method statement is included in the definition of the extended agent and must have the following syntax:

```
ACCESS "/jsdl/filename_of_the_ JSDL_template -var name=value,name=value,..."
```

where `-var name=value` is optional and represents one or more variables passed by the workload broker workstation to dynamic workload broker at job submission time.

3. Add the extended agent to the plan as you do with any other workstation. The workload broker workstation has the task of managing the lifecycle of the extended agent, notifying the master domain manager that it is up and running.

When jobs are run on the extended agent, they are routed to the workload broker workstation, which handles them differently from other jobs. Instead of searching for the name of the JSDL definition in the task string of the job, the workload broker workstation:

1. Gets the name of the target JSDL from the access method, and passes the task string as a value for variable `${tws.job.taskstring}`.
2. The task string value is replaced in the script element of the target JSDL, and is used as a command string to run on the target agent that is dynamically selected by the dynamic workload broker.

   Thus, the JSDL definition invoked by the workload broker workstation works as a sort of template that you can use to run different task strings defined in different Tivoli Workload Scheduler jobs: the same JSDL document is reused for multiple jobs.

Example

You want to exploit dynamic workload broker to run a job named `SUBMIT_JOBXA` and you want to use a JSDL template. The following definitions accomplish this:

1. The definition of the workload broker workstation. It is named `DGCENTER_DWB` and it is of type `BROKER`. There can be only one workload broker workstation running at a time in a Tivoli Workload Scheduler network (this applies also to the dynamic workload broker server).

```
CPUNAME DGCENTER_DWB
  OS OTHER
  NODE DGCENTER TCPADDR 41111
  ENGINEADDR 31111
  DOMAIN MASTERDM
  FOR MAESTRO
    TYPE BROKER
    AUTOLINK ON
    BEHINDFIREWALL OFF
    FULLSTATUS OFF
END
```

2. The definition of extended agent `DGCENTER_DWBXA`. The extended agent must:
   - Be hosted by the workload broker workstation (`DGCENTER_DWB` in this example).
   - Include the access method. While normally the `ACCESS` keyword is followed by the name of the program that implements the specific access method, in the case of JSDL templates it needs only to define the name of the JSDL document you use as template - that must be stored in the dynamic workload broker Job Repository in the Tivoli Workload Scheduler database and available in a local folder in the workstation where you run the Job

Brokering Definition Console- and whatever other parameters you want to use. These items must be enclosed between double quotes.

This requires that you created the JSDL document you will be using as a template (named SJT in this example), defining the required resources, candidate hosts, and scheduling and optimization preferences, and specifying ${tws.job.taskstring} in the Script field of the executable.

```
CPUNAME DGCENTER_DWBXA
  OS OTHER
  NODE DGCENTER TCPADDR 41111
  FOR MAESTRO HOST DGCENTER_DWB ACCESS "/jsdl/SJT -var
target=D:\vmware,RES=RES1"
    TYPE X-AGENT
    AUTOLINK OFF
    BEHINDFIREWALL OFF
    FULLSTATUS OFF
END
```

3. The definition of job SUBMIT_JOBXA in Tivoli Workload Scheduler:

```
DGCENTER_DWBXA#SUBMIT_JOBXA
 SCRIPTNAME "C:\TWS\Utils\Jobs\javacount_on.bat"
 STREAMLOGON Administrator
 DESCRIPTION "Added by composer."
 TASKTYPE WINDOWS
 RECOVERY STOP
```

The fact that the job is defined to run on extended agent DGCENTER_DWBXA, hosted by the workload broker workstation and matched with the SJT JSDL definition, drives the process that:

a. Submits the job via dynamic workload broker
b. Uses the specifications of the SJT JSDL definition
c. Replaces variable ${tws.job.taskstring} in SJT with the task string of SUBMIT_JOBXA, that is:

   ```
   C:\TWS\Utils\Jobs\javacount_on.bat
   ```

Scenarios for creating job definitions

These scenarios provide examples of creating job definitions with different types of requirements.

JSDL and the Job Brokering Definition Console provide very flexible tools for defining jobs. The following scenarios provide examples of how to set up a job definition to achieve your objectives for identification of targets, resource allocation, and load balancing:

- "Scenario: Creating a job definition using a computer resource group" on page 31.

  This scenario demonstrates the use of a resource group to specify candidate target systems.
- "Scenario: Creating a job definition using a logical resource group" on page 31.

  This scenario demonstrates the use of a resource group to specify logical resources required for the job..
- "Scenario: Creating a job definition for a job to run on x86 processors" on page 32

  This scenario demonstrates the use of advanced requirements for resources and the use of resource properties for defining load-balancing rules.
- "Scenario: Creating a job definition for a script to run on a specific operating system" on page 34

This scenario demonstrates the creation of relationships between an operating system type software resource and an additional resource requirement.

- "Scenario: Alternative operating system requirements" on page 35

  This scenario demonstrates the definition of two resource requirements related to specific operating system types and a minimum free physical memory requirement.

Scenario: Creating a job definition using a computer resource group

In this scenario, a job is created to run the inventory update program, selecting the target system from the **invadmin** resource group set up to include the computers that are suitable for running the script.

To create a job definition that does this, perform the following steps:

1. In the Job Brokering Definition Console select **File** > **New** > **Job brokering definition** and create a new job definition named `compgroupjob`. The job definition opens at the Overview page with the job name assigned.
2. Open the Application page and identify and attach the script, as follows:
   a. In the **Type** menu, select **Executable**.
   b. In the **Executable** pane, select the **Executable File** radio button.
   c. Click **Browse** and locate the executable file.
   d. Click **OK**.
3. Open the Resources page and specify the resource group, as follows:
   a. Select the **Advanced Requirements** tab.
   b. In the Resource Group pane, click **Add**. The Resource Group Details dialog box is displayed.
   c. In the Group Name field, type `invadmin` (the resource group name, as defined in the Dynamic Workload Console).
4. Select **File** > **Save** to save the job definition file.

The JSDL file created for this scenario has the following syntax:

```
<jsdl:jobDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl JSDL.xsd
http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle JSDL-Native.xsd"
description="Run inventory update script on a computer from the
invadmin resource group.
" name="compgroupjob">
  <jsdl:application name="executable">
    <jsdle:executable path="/opt/invupdate">
</jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:group name="invadmin"/>
  </jsdl:resources>
</jsdl:jobDefinition>
```

Scenario: Creating a job definition using a logical resource group

In this scenario, the target for the job is determined by several requirements defined as logical resources. A resource group has been created to include all the logical resources required for the job.

To create a job definition that does this, perform the following steps:

1. In the Job Brokering Definition Console select**File** > **New** > **Job brokering definition** and create a new job definition named `loggroupjob`. The job definition opens at the Overview page with the job name assigned.
2. Open the Application page and define the required information for the application that the job is to run.
3. Open the Related Resources page and create a requirement for a logical resource, as follows:
   a. In the Resource Requirements pane, click **Add**. The Resource Requirement Details dialog box is displayed.
   b. In the **ID** field, specify a meaningful ID, in this example, `loggroup`.
4. Open the Resources page and create a relationship to the resource requirement, as follows:
   a. Select the **Advanced Requirements** tab.
   b. In the Relationships pane, click **Add**. The Relationship Details dialog box is displayed.
   c. In the **Type** menu, select **Associates with**.
   d. In the **Target** menu, select the resource requirement that you created and click **OK**.
5. Switch back to the Related Resources page and add the logical resource group as follows:
   a. In the Resource Group pane, click **Add**. The Resource Group Details dialog box is displayed.
   b. In the Group Name field, type the resource group name, as defined in the Dynamic Workload Console.
6. Select **File** > **Save** to save the job definition file.

The JSDL file created for this scenario has the following syntax:

```
<jsdl:jobDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl JSDL.xsd"
description="A job whose requirements are defined by a number of logical
resources. " name="loggroupjob">
  <jsdl:application name="executable">
    <jsdle:executable path="/opt/myExecutable">
    </jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:relationship target="loggroup" type="AssociatesWith"/>
  </jsdl:resources>
  <jsdl:relatedResources id="loggroup" type="LogicalResource">
    <jsdl:group name="logresgroup"/>
  </jsdl:relatedResources>
</jsdl:jobDefinition>
```

Scenario: Creating a job definition for a job to run on x86 processors

In this scenario, a job is created to run the application, appx86. The application must run on a workstation with an x86 processor where the CPU usage between 3 and 30%. Load balancing is to be defined by an objective to keep CPU use on matching resources to a minimum.

To create the job definition, perform the following steps:

1. In the Job Brokering Definition Console select **File** > **New** > **Job brokering definition** and create a new job definition named `x86job`. The job definition opens at the Overview page with the job name assigned.
2. Open the Application page and define the required information for the appx86 application that the job is to run.
3. Open the Resources page and specify the processor and CPU usage requirements as follows:
   a. Select the Advanced Requirements tab.
   b. Click **Add Requirement**. The Resource Property Details dialog box is displayed.
   c. In the **Property Name** menu, select **CPU Utilization**.
   d. In the **Property Value** section, select the **Range Value** radio button and assign values of 3 to **Minimum** and 30 to **Maximum**.
   e. Click **Add Requirement**. The Resource Property Details dialog box is displayed.
   f. In the **Property Name** menu, select **Processor type**.
   g. In the **Property Value** section, select the **Exact Value** radio button and assign a values of x86.
4. Open the Optimization page and specify the load balancing requirement as follows:
   a. In the **Type** menu, select **Balance load between resources by optimization objective**.
   b. In the **Resource Type** menu, select **Computer System**.
   c. In the **Resource Property** menu, select **CPU Utilization**.
   d. In the **Optimization Objective** menu, select the **Minimize**.
5. Select **File** > **Save** to save the job definition file.

The JSDL file created for this scenario has the following syntax:

```
<jsdl:jobDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl JSDL.xsd"
description="Job to run on X86 processors" name="x86job">
 <jsdl:application name="executable">
    <jsdle:executable path="/opt/appx86">
    </jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:properties>
      <jsdl:requirement propertyName="CPUUtilization">
        <jsdl:range>
          <jsdl:minimum>3</jsdl:minimum>
          <jsdl:maximum>30</jsdl:maximum>
        </jsdl:range>
      </jsdl:requirement>
      <jsdl:requirement propertyName="ProcessorType">
        <jsdl:exact>x86</jsdl:exact>
      </jsdl:requirement>
    </jsdl:properties>
  </jsdl:resources>
  <jsdl:optimization name="JPT_JSDLOptimizationPolicyType">
    <jsdl:objective propertyObjective="minimize"
          resourcePropertyName="CPUUtilization"
          resourceType="ComputerSystem"/>
  </jsdl:optimization>
</jsdl:optimization>
</jsdl:jobDefinition>
```

Scenario: Creating a job definition for a script to run on a specific operating system

In this scenario, a job is created to run a script on a Red Hat Enterprise Linux system.

By specifying candidate operating systems, you can define the type of operating system on which a job is to run, in this case Linux. To direct the job to a specific flavor of Linux, you must define a related resource and link it to the job resources by creating a relationship. To create a job definition that does this, perform the following steps:

1. In the Job Brokering Definition Console select **File** > **New** > **Job brokering definition** and create a new job definition named `rhjob`. The job definition opens at the Overview page with the job name assigned.
2. Open the Application page and define the required information for the application that the job is to run.
3. Open the Resources page and specify the operating system type requirement, as follows:
   a. Select the **Software Requirements** tab.
   b. In the **Candidate Operating Systems** pane, click **Add**. The Operating System Details dialog box is displayed.
   c. In the **Type** menu, select **LINUX** and click **OK**.
4. Open the Related Resources page and create a resource requirement for the Red Hat flavor of Linux, as follows:
   a. In the Resource Requirements pane, click **Add**. The Resource Requirement Details dialog box is displayed.
   b. In the **ID** field, specify a meaningful ID, in this example, `redhat`.
   c. In the **Type** menu, select **Operating System**.
   d. In the Resource Properties pane, click **Add Requirement.** The Resource Property details dialog box is displayed.
   e. In the **Property Name** menu, select **Display Name**.
   f. In the **Property Value** , type `Red*`.
5.
6. Switch back to the Resources page to link the resource requirement to the operating system resource.
   a. Select the Advanced Requirements tab.
   b. In the Relationships pane, click **Add**. The Relationship Details dialog box is displayed.
   c. In the **Type** menu, select **Contains**.
   d. In the **Target** menu, select the Red Hat resource requirement that you created and click **OK**.
7. Select **File** > **Save** to save the job definition file.

The JSDL file created for this scenario has the following syntax:

```
<jsdl:jobDefinition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl
JSDL.xsd" description="Job to run on Red Hat Linux" name="rhjob">
  <jsdl:application name="executable">
    <jsdle:executable path="/opt/myExecutable">
    </jsdle:executable>
  </jsdl:application>
```

```
  <jsdl:resources>  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="LINUX"/>
    </jsdl:candidateOperatingSystems>
    <jsdl:relationship target="redhat" type="Contains"/>
  </jsdl:resources>
  <jsdl:relatedResources id="redhat" type="OperatingSystem">
    <jsdl:properties>
      <jsdl:requirement propertyName="DisplayName">
        <jsdl:exact>red*</jsdl:exact>
      </jsdl:requirement>
    </jsdl:properties>
  </jsdl:relatedResources>
</jsdl:jobDefinition>
```

Scenario: Alternative operating system requirements

In this scenario, a definition is created for a job that can run on either a Linux or an AIX computer.

The job can run on Linux operating systems with a minimum of 512 MB of RAM or on AIX operating systems with a minimum of 1024 MB of RAM. The job definition must include a resource requirement that specifies the two alternative requirements.

To create job definition for this job, perform the following steps:

1. In the Job Brokering Definition Console select **File** > **New** > **Job brokering definition** and create a new job definition named `jobWithRequirementsByOS`. The job definition opens at the Overview page with the job name assigned.
2. Open the Application page and define the required information for the application that the job is to run.
3. Open the Related Resources page.
4. In the Resource Requirements pane, click **Add** then specify a meaningful value for the **ID** field. In this example it is `OperatingSystemType`.
5. In the Resource Properties pane, define the logic that describes the two alternative operating system requirements, as follows:
   a. Click **Add OR Operand** to indicate that you are defining alternatives.
   b. Highlight the OR operand and click **Add AND Operand** to indicate that the alternative includes more than requirement.
   c. Highlight the AND operand and click **Add Requirement**.
   d. In the Resource Property Details dialog, select **Operating System Type** from the **Property Name** menu and type `LINUX` in the **Property value** field.
   e. Highlight the AND operand again and click **Add Requirement**.
   f. In the Resource Property Details dialog, select **Total Physical Memory** from the **Property Name** menu and type `512` in the **Property value** field.
   g. Highlight the OR operand again and click **Add AND Operand** to add the requirements for the second alternative.
   h. Highlight the new AND operand and click **Add Requirement**.
   i. In the Resource Property Details dialog, select **Operating System Type** from the **Property Name** menu and type `AIX` in the **Property value** field.
   j. Highlight the AND operand again and click **Add Requirement**.
   k. In the Resource Property Details dialog, select **Total Physical Memory** from the **Property Name** menu and type `1024` in the **Property value** field.

6. Open the Resources page and create a relationship to the resource requirement, as follows:
   a. Select the **Advanced Requirements** tab.
   b. In the Relationships pane, click **Add**. The Relationship Details dialog box is displayed.
   c. In the **Type** menu, select **Contains**.
   d. In the **Target** menu, select the resource requirement `OperatingSystemType` and click **OK**.
7. Select **File** > **Save** to save the job definition file.

The JSDL file created for this scenario has the following syntax:

```
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/
    1.0/jsdl" xmlns:jsdle="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle"
    xmlns:xmi="http://www.omg.org/XMI" xmi:version="2.0" description="This job
    has different requirements for memory based on the operating system it will
    run on  " name="jobWithRequirementsByOS">
  <jsdl:application name="executable">
    <jsdle:executable path="/opt/myExecutable">
    </jsdle:executable>
  </jsdl:application>
  <jsdl:resources>
    <jsdl:relationship target="OperatingSystemType" type="Contains"/>
  </jsdl:resources>
  <jsdl:relatedResources id="OperatingSystemType" type="OperatingSystem">
    <jsdl:properties>
      <jsdl:or>
        <jsdl:and>
          <jsdl:requirement propertyName="OperatingSystemType">
            <jsdl:exact>LINUX</jsdl:exact>
          </jsdl:requirement>
          <jsdl:requirement propertyName="TotalPhysicalMemory">
            <jsdl:range>
              <jsdl:minimum>512</jsdl:minimum>
            </jsdl:range>
          </jsdl:requirement>
        </jsdl:and>
        <jsdl:and>
          <jsdl:requirement propertyName="OperatingSystemType">
            <jsdl:exact>AIX</jsdl:exact>
          </jsdl:requirement>
          <jsdl:requirement propertyName="TotalPhysicalMemory">
            <jsdl:range>
              <jsdl:minimum>1024</jsdl:minimum>
            </jsdl:range>
          </jsdl:requirement>
        </jsdl:and>
      </jsdl:or>
    </jsdl:properties>
  </jsdl:relatedResources>
</jsdl:jobDefinition>
```

Chapter 4. Submitting and tracking jobs

Although you should normally use the standard Tivoli Workload Scheduler means to schedule and submit workload, there is also an additional way to submit jobs directly to dynamic scheduling using either the Dynamic Workload Console or the dynamic workload broker command line, described in the next chapter. This implies that you only need to write a JSDL job definition. You do not need to write a job definition in Tivoli Workload Scheduler and the workload broker workstation does not come into play. Choosing to do so, however, results in not exploiting the scheduling and choreography services of Tivoli Workload Scheduler. This chapter explains how to submit and track jobs using the Dynamic Workload Console.

Job definitions are stored in the **Job Repository** and you search for them and select them when the job needs to be submitted. At submission time, you can also specify that a job run on the same resource as a job that has previously run. The job definition provides all necessary parameters for a job to run, however, you can add to and change the parameters defined in the job definition for the job instance you are submitting. This does not change the job definition stored in the Job Repository.

The lifecycle of a job involves the following sequence of phases:

Using the Dynamic Workload Console, you can manage the whole lifecycle of a job by performing the following task:

- "Monitoring submitted jobs" on page 39

Submitting jobs with affinity relationships

An affinity relationship is established between two or more jobs when you want them to run on the same resource.

An affinity relationship is useful when the results of one job are required for the next job to run. You can define an affinity relationship between two or more jobs in the following ways:

- "Submitting a job with affinity from the Dynamic Workload Console"
- "Submitting a job with affinity from the command line" on page 38
- "Defining affinity relationships" on page 5 in Tivoli Workload Scheduler.

Submitting a job with affinity from the Dynamic Workload Console

To submit a job to run on the same resources as a previous job, do the following:

1. In the console navigation tree, expand **Definitions** and click **Jobs**.
2. Search for a job definition stored in the **Job Repository** database.
3. Select the job definition that you want to submit.
4. Click **Submit..**.
5. Click **Go**. The **Job Definitions** wizard starts.
6. You can optionally define an alias for the job by selecting **Provide an alias to the job for this submission**. Subsequently, you can use the alias name to submit a new job using the **jobsubmit** command to define an affinity relationship with the job you are submitting now. You can also use the alias as

a user-friendly alternative to the job ID when performing queries on jobs from the command line or tracking job instances.

7. In the **Target Resource** area, select the option **Specify that this job runs on the same resources as a previous job**.
8. Click **Next**. If you selected to specify an alias, then you are prompted to provide the alias name. Click **Next**.
9. Search and select the job with which you want to establish an affinity relationship. The current job will then run on the same resource as the job you select. Click **Next**.
10. On the summary page, review your selections and click **Finish** to submit the job.

The job is submitted on the same resources as the job you selected. You can now check the job status by selecting **Tracking** > **Job Instances** from the navigation. See "Monitoring submitted jobs" on page 39 for more information.

Submitting a job with affinity from the command line

The **jobsubmit** command requires a job ID or an alias name for the affine job.

Submit the following command to make the job defined in job definition WinJob2.jsdl, with the alias WJ220070606, run on the same resource as the previously run job, WinJob1, which was submitted with the alias, WJ120070606:

```
jobsubmit -jsdl WinJob2.jsdl -alias WJ220070606 -affinity alias=WJ120070606
```

Submitting jobs with variables

When you submit a job, you can define or change a variable to be used by the job.

During job submission, you can define variables that are to be used by the job itself or to assign the job to a resource. You can add new variables or override the default values for variables that are included in the job definition. For more information about including variables in the job definitions see "Using variables in job definitions" on page 27.

Submitting a job with variables from the command line

The **jobsubmit** command submits jobs from the command line interface. You can include arguments to change the value of predefined variables and add new ones. For example, the job definition for Job1 includes the variable `memory` with the value `512` which is used to set the free physical memory requirement. To increase the requirement to `1024` when submitting the job, issue the following command:

```
jobsubmit -jdname Job1 -var memory=1024
```

Job statuses

This section describes all supported statuses for a job as returned both by the command line interface and by the Dynamic Workload Console. It also lists the operations a user can perform depending on the status the job is in.

Table 5. Job statuses and supported operations

Dynamic Workload Console status	Icon	Command line status	You can cancel the job	You can browse the job output	You can define affinity
Run failed	RED	FAILED_ EXECUTION		√	√
Resource allocation failed	RED	RESOURCE_ ALLOCATION_ FAILED			
Unable to start	RED	NOT_EXECUTED			√
Unknown	YELLOW	UNKNOWN		√	√
Submitted	WAITING	SUBMITTED	√		
Waiting for resources	WAITING	WAITING_FOR_ RESOURCES	√		
Resource allocation received	WAITING	RESOURCE_ ALLOCATION_ RECEIVED	√		
Submitted to agent	WAITING	SUBMITTED_ TO_ENDPOINT	√		√
Waiting for reallocation	WAITING	RESOURCE_ REALLOCATE	√		
Cancel pending	ABORT	PENDING_ CANCEL		√	√
Cancel allocation	ABORT	CANCEL_ ALLOCATION		√	√
Canceled	ABORT	CANCELLED		√	√
Running	RUNNING	EXECUTING	√	√	√
Completed successfully	GREEN	SUCCEEDED_ EXECUTION		√	√

Note: You can define an affinity relationship with a job in Canceled state only if the job was canceled while running.

Monitoring submitted jobs

A job instance is a job that is submitted to run at a specific time. You can track the outcome of a submitted job from the Dynamic Workload Console.

Prerequisite:A job must be submitted to dynamic workload broker before you can view its instances. Submitted jobs are stored in the Job Repository for a default time interval. See the *Tivoli Workload Scheduler: Administration Guide, SC23-9113* for information about configuring this interval in the JobDispatcherConfig.properties file. You can access the following information about job instances:

- Status of the job instance.
- The host name of the computer where the job instance ran.

- The return code of the job instance.
- The date and time the job was submitted.
- The date and time the job started and finished running.

For example, to view all jobs that have resulted in error within the last 24 hours, follow these steps:

1. In the console navigation tree, expand **Tracking** and click **Job Instances**. The **Track Job Instance Search Criteria** page is displayed
2. Specify the search criteria for the job instances as follows:
   a. In the Submission Time section, select the **Last 24 Hours** radio button.
   b. In the Job Status section, select **Error Conditions**.
   c. Click **Search**.

   The results are displayed in the **Job Tracking** page.

As an alternative, you can take the following steps:

1. In the console navigation tree, expand **Definitions** and click **Jobs**. The **Job Definition Search Criteria** page is displayed.
2. Specify the search criteria for the job definition associated with the job instance that you want to view.
3. Select the job for which you want to show instances.
4. Click **Show Instances**. The results are displayed in the **Job Definitions Search Result** page.

Once a job is submitted to the Job Dispatcher, it goes through the phases of job scheduling, allocation of resources, and finally, job execution. Problems might occur along the way, and there are specific job statuses that identify at which point things went wrong. The following is a list of job statuses that a job can assume after it is submitted to be run:

Job completing successfully
The job goes through the following statuses: **Submitted** > **Waiting for resources** > **Resource allocation received** > **Submitted to agent** > **Running** > **Completed successfully**.

Job being canceled
The job goes through the following statuses: **Submitted** > **Waiting for resources** > **Resource allocation received** > **Submitted to agent** > **Running** > **Cancel pending** > **Canceled**.

Job being reallocated
The job is allocated to a computer which is temporarily unreachable, for example because of a network problem. The job goes through the following statuses: **Submitted** > **Waiting for resources** > **Resource allocation received** > **Waiting for reallocation** > **Waiting for resources** .

Job encountering an error
There can be several reasons for the error. Here are some examples:

- The job encounters an error because the selected working directory does not exist on the target system. The job goes through the following statuses: **Submitted** > **Waiting for resources** > **Resource allocation received** > **Submitted to agent** > **Unable to start**. As the job cannot start, no output is available.

- The job requires an operating system which is not available in the environment. The job goes through the following statuses: **Submitted** > **Waiting for resources** > **Resource allocation failed**.
- The job encounters an error because one of the parameters specified in the job is not supported on the target system. The job goes through the following statuses: **Submitted** > **Waiting for resources** > **Resource allocation received** > **Submitted to agent** > **Running** > **Run failed**.

When viewing the job instance details for this job **Job Tracking** page, the reason for the error is displayed. You can also use the ID indicated in the **Identifier** field to retrieve more information on the job results, which is stored in a series of log files on the computer where the job ran. The name of the computer where the job ran is also indicated in the **Job Tracking** page. Locate the computer and analyze the log files available in the folder named with the job ID in the following path:

TWA_home/TWS/stdlist/JM/yyyy.mm.dd/archive

Every job has a compressed file whose name is the job ID, for example:

ed1d4933-964b-3f5e-8c73-f720919491d6.zip

The compressed file contains the following:

diagnostics.log
: May or may not include diagnostic information.

jm_exit.properties
: Includes the return code as well as other job statistics, like CPU and memory usage.

out.log
: Includes the full job output.

trace.log
: Includes the output trace of the task launcher process spawned by the Tivoli Workload Scheduler agent to run the job.

trace.log_cmd
: Includes the command used to run the task launcher.

Chapter 5. Using the command line interface

Dynamic workload broker provides a command line for running a set of commands. You can use the command line interface to save, submit, query, monitor, cancel jobs, and view the job output. You can also archive database tables.

Commands are stored in the following location on the master domain manager:
TWA_home/TDWB/bin

The following commands are available:

Table 6. Dynamic workload broker commands

Command	Purpose	See
exportserverdata	Downloads the list of dynamic workload broker instances from the Tivoli Workload Scheduler database to a temporary file. Use to record a port number or host name change.	the section about exportserverdata command in *Tivoli Workload Scheduler: User's Guide and Reference*
importserverdata	Uploads the list of dynamic workload broker instances from the temporary file to the Tivoli Workload Scheduler database after you are done recording a port number or host name change.	the section about importserverdata command in *Tivoli Workload Scheduler: User's Guide and Reference*
jobsubmit	Submits a job to the Job Dispatcher.	"jobsubmit command - Submitting jobs" on page 47
jobdetails	Returns property information for the specified job.	"jobdetails command - Viewing details on jobs" on page 52
jobquery	Returns a list of submitted jobs matching the selection criteria.	"jobquery command - Performing queries on jobs" on page 49
jobcancel	Cancels a submitted job.	"jobcancel command - Canceling jobs" on page 54
jobstore	Manages job definitions.	"jobstore command - Managing job definitions" on page 55
jobgetexecutionlog	Displays the results of submitted jobs.	"jobgetexecutionlog command - Viewing job output" on page 57
movehistorydata	Moves data present in the **Job Repository** database to the archive tables.	the section about movehistorydata command in *Tivoli Workload Scheduler: User's Guide and Reference*
resource	Creates and manages resources and groups. Manages associated computers.	the section about resource command in *Tivoli Workload Scheduler: User's Guide and Reference*

Command-line syntax

This chapter uses the following special characters to define the syntax of commands:

[] Identifies optional attributes. Attributes not enclosed in brackets are required.

... Indicates that you can specify multiple values for the previous attribute.

| Indicates mutually exclusive information. You can use the attribute to the left of the separator or the attribute to its right. You cannot use both attributes in a single use of the command.

{} Delimits a set of mutually exclusive attributes when one of the attributes is required. If the attributes are optional, they are enclosed in square brackets ([]).

\ Indicates that the syntax in an example wraps to the next line.

Command-line configuration file

The `CLIConfig.properties` file contains configuration information which is used when typing commands. By default, arguments required when typing commands are retrieved from this file, unless explicitly specified in the command syntax.

The `CLIConfig.properties` file is created at installation time and is located on the master domain manager in the following path:

TWA_home`/TDWB/config`

The `CLIConfig.properties` file contains the following set of parameters:

Dynamic workload broker default properties

ITDWBServerHost
Specifies the IP address of dynamic workload broker.

ITDWBServerPort
Specifies the number of the dynamic workload broker port. The default value is **9550**.

ITDWBServerSecurePort
Specifies the number of the dynamic workload broker port when security is enabled. The default value is **9551**.

use_secure_connection
Specifies whether secure connection must be used. The default value is **false**.

KeyStore and trustStore file name and path

keyStore
Specifies the name and path of the keyStore file containing private keys. A keyStore file contains both public keys and private keys. Public keys are stored as signer certificates while private keys are stored in the personal certificates. The default value is /Certs/TDWBClientKeyFile.jks.

trustStore
Specifies the name and path of the trustStore file containing public

keys. A trustStore file is a key database file that contains public keys. The public key is stored as a signer certificate. The keys are used for a variety of purposes, including authentication and data integrity. The default value is /Certs/TDWBClientTrustFile.jks.

Passwords for keyStore and trustStore files

keyStorepwd
Specifies the password for the keyStore file.

trustStorepwd
Specifies the password for the trustStore file.

File types for keyStore and trustStore files

keyStoreType
Specifies the file type for the keyStore file. The default value is JKS.

trustStoreType
Specifies the file type for the trustStore file. The default value is JKS.

Default user ID and password for dynamic workload broker

tdwb_user
Specifies the user name for a user authorized to perform operations on dynamic workload broker when security is enabled. The default value is **ibmschedcli**. This password must be previously defined on IBM WebSphere. For more information on security considerations, see the *Tivoli Workload Scheduler: Administration Guide*, *SC23-9113*.

tdwb_pwd
Specifies the password for a user authorized to perform operations on dynamic workload broker when security is enabled. This password must be previously defined on IBM WebSphere . For more information on security considerations, refer to *Tivoli Workload Scheduler: Administration Guide*.

Detail level for command-line log and trace information

logger.Level
Specifies the detail level for the command-line trace and log files. The command-line trace and log files are created in the following location:

log file
TWA_home/TDWB/logs/Msg_cli.log.log

trace file
TWA_home/TDWB/logs/Trace_cli.log

The default value is INFO.

logger.consoleLevel
Specifies the detail level for the log and trace information to be returned to standard output. The default value is FINE. Supported values for both the **consoleLevel** and **loggerLevel** parameters are as follows:

ALL Indicates that all messages are logged.

SEVERE
Indicates that serious error messages are logged.

WARNING
Indicates that warning messages are logged.

INFO Indicates that informational messages are logged.

CONFIG
Indicates that static configuration messages are logged.

FINE Indicates that tracing information is logged.

FINER
Indicates that detailed tracing information is logged.

FINEST
Indicates that highly detailed tracing information is logged.

OFF Indicates that logging is turned off.

logger.limit
Specifies the maximum size of a log file in bytes. The default value is 400000. When the maximum size is reached, a new file is created, until the maximum number of files is reached. When all files reach the maximum size and the maximum number of files is exceeded, the oldest file is re-written.

logger.count
Specifies the maximum number of log files. The default value is 6. When the maximum size is reached, a new file is created, until the maximum number of files is reached. When all files reach the maximum size and the maximum number of files is exceeded, the oldest file is re-written. When a new file is created the 0 suffix is appended after the file extension. The file with the 0 suffix is always the current file. Any older files are renumbered accordingly.

java.util.logging.FileHandler.pattern
Specifies that the trace information for the Java™ Virtual Machine is logged in the Trace_cli.log file. The default value is INFO.

java.util.logging.FileHandler.limit
Specifies the maximum size of a trace file in bytes. The default value is 400000. When the maximum size is reached, a new file is created, until the maximum number of files is reached. When all files reach the maximum size and the maximum number of files is exceeded, the oldest file is re-written.

java.util.logging.FileHandler.count
Specifies the maximum number of trace files. The default value is 6. When the maximum size is reached, a new file is created, until the maximum number of files is reached. When all files reach the maximum size and the maximum number of files is exceeded, the oldest file is re-written. When a new file is created the 0 suffix is appended after the file extension. The file with the 0 suffix is always the current file. Any older files are renumbered accordingly.

java.util.logging.FileHandler.formatter
Specifies the formatter to be used for the Trace_cli.log file. The default value is `com.ibm.logging.icl.jsr47.CBEFormatter`.

DAO common configuration
This section defines the RDBMS settings for the **exportserverdata**, **importserverdata**, and **movehistorydata** commands. These commands use the RDBMS installed on dynamic workload broker These parameters are

valorized at installation time and should not be modified, except for `com.ibm.tdwb.dao.rdbms.useSSLConnections` as noted below.

com.ibm.tdwb.dao.rdbms.rdbmsName
Specifies the RDBMS name.

com.ibm.tdwb.dao.rdbms.useDataSource
Specifies the data source to be used.

com.ibm.tdwb.dao.rdbms.jdbcPath
Specifies the path to the JDBC driver.

com.ibm.tdwb.dao.rdbms.jdbcDriver
Specifies the JDBC driver.

com.ibm.tdwb.dao.rdbms.userName
Specifies the name of the RDBMS user.

com.ibm.tdwb.dao.rdbms.password
Specifies the password of the RDBMS user.

com.ibm.tdwb.dao.rdbms.useSSLConnections
Specifies that access to the Tivoli Workload Scheduler DB2 database by some of the CLI commands is over SSL. Is set to `FALSE` by default. You must set to `TRUE`, if the database is DB2 and you use FIPS security, for the following commands to work:

- **exportserverdata**
- **importserverdata**
- **movehistorydata**

jobsubmit command - Submitting jobs

Use the **jobsubmit** command to submit jobs to the Job Dispatcher.

Syntax

jobsubmit ?

jobsubmit [**-usr** *user_name* **-pwd** *password*] {**-jsdl** *jsdl_file* | **-jdname** *job_definition_name*} [**-alias** *job_alias*] [**-var** *variable=value...*] [**-affinity** {**jobid**=*job_id* | **alias**=*alias*}] [**-configFile** *configuration_file*]

Description

This command submits a job to the Job Dispatcher. When the job is submitted, it is assigned a unique ID, which can be used for retrieving information on and canceling jobs.

You can use this command to submit jobs saved locally on the dynamic workload broker server or saved in the Job Repository. To submit a local job, use the **-jsdl** option and specify the path to the JSDL file. To submit a job saved in the Job Repository, use the **-jdname** option and specify the job definition name.

When submitting jobs, you can also define an alias to be used as an alternative to the job ID when performing queries on the job, or for defining subsequent jobs as affine. To define affinity between two or more jobs, use the **-affinity** option when submitting the subsequent jobs. You define jobs as affine when you want them to run on the same resource, for example when the second job must use the results generated by the previous job.

Options

? Displays help information.

-usr *user_name*
Specifies the username for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the username is not defined in the CLIConfig.properties configuration file (with the `tdwb_user` keyword).

-pwd *password*
Specifies the password for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the password is not defined in the CLIConfig.properties configuration file (with the `tdwb_pwd` keyword).

-jsdl *jsdl_file*
Specifies the name and path to a local JSDL file which provides the parameters for a job when it is submitted. This parameter is required when the **jdname** parameter is not specified.

-jdname *job_definition_name*
Specifies the name of a job definition stored in the **Job Repository** database. The job definition name is defined within the JSDL file and can be modified only by editing the JSDL file. This parameter is required when the **jsdl** parameter is not specified. To obtain this name, you can use the **Definitions** > **Jobs** task from the Dynamic Workload Console console navigation tree, or the **jobstore** command specifying one or more of the query options. For more information on the **jobstore** command, see "jobstore command - Managing job definitions" on page 55.

-alias *job_alias*
Indicates that an alias must be generated for the job being submitted. You can use the alias as a user-friendly alternative to the job ID when performing queries on the job. You can also use the alias when submitting new jobs so that the new job is affine to the job having this alias. To define affinity between two or more jobs, use the **-affinity** option when submitting the new jobs. You define jobs as affine when you want them to run on the same resource. On Windows systems, the maximum length for the alias is 200 characters, if you used the default installation paths for WebSphere Application Server and dynamic workload broker.

-var *variable=value*
Specifies a variable and the associated value. You can also specify a list of variables by separating them with a comma. This value overrides the value specified when creating the JSDL file. You can also specify new variables without previously defining them in the JSDL file.

-affinity jobid=*job_id*
Specifies that the current job is affine to a previously submitted job. To establish the affinity relationship, specify the job ID for the previous job. The job ID is automatically generated at submission time.

-affinity alias=*alias*
Specifies that the current job is affine to a previously submitted job. To establish the affinity relationship, specify the job alias for the previous job. The job alias is generated at submission time when you specify the **-alias** option.

-configFile *configuration_file*
Specifies the name and path of a custom configuration file. This parameter is

optional. If this parameter is not specified, the default configuration file is assumed. For more information on the configuration file, see "Command-line configuration file" on page 44.

Authorization

The user name and password for the command are defined in the CLIConfig.properties file. To override the settings defined in this file, you can enter the user name and password when typing the command. For more information on the CLIConfig.properties file, see "Command-line configuration file" on page 44.

Return Values

The **jobsubmit** command returns one of the following values:

0 Indicates that **jobsubmit** completed successfully.

< > 0 Indicates that **jobsubmit** failed.

Examples

1. To submit the local job `test_job` located in the `/staging_area/accounts/` path using the configuration parameters specified in the `custom_config.properties` configuration file, type the following command:

   ```
   jobsubmit -jsdl /staging_area/accounts/test_job -configFile
    /opt/test/custom_config.properties
   ```

2. To submit the job definition `domestic_accounts` saved in the Job Repository, type the following command:

   ```
   jobsubmit -jdname domestic_accounts
   ```

See Also

"jobdetails command - Viewing details on jobs" on page 52

jobquery command - Performing queries on jobs

Use the **jobquery** command to perform advanced queries on submitted jobs.

Syntax

jobquery ?

jobquery [**-usr** *user_name* **-pwd** *password*] {[**-status** *status*...] [**-submitter** *submitter*] [**-name** *job_definition_name*] [**-alias** *job_alias*] [**-sdf** *submit_date_from*] [**-sdt** *submit_date_to*] [**-jsdf** *job_start_date_from*] [**-jsdt** *job_start_date_to*] [**-jedf** *job_end_date_from*] [**-jedt** *job_end_date_to*]} [**-configFile** *configuration_file*]

Description

This command performs advanced queries on submitted jobs based on the following attributes:

- job status
- name of the user who submitted the job
- job name
- job alias
- job submission date
- job start date

- job completion date

You can also use this command to retrieve the job ID generated at submission time, which is required when running the **jobstatus**, **jobdetails** and **jobcancel** commands. To retrieve the job ID, specify the **-name** option.

Options

? Displays help information.

-usr *user name*
Specifies the user name for a user authorized to perform operations on the command line. This option is required when security is enabled and the user name is not defined in the CLIConfig.properties configuration file (with the `tdwb_user` keyword).

-pwd *password*
Specifies the password for a user authorized to perform operations on the command line. This option is required when security is enabled and the password is not defined in the CLIConfig.properties configuration file (with the `tdwb_pwd` keyword).

-status *status*
Specifies the status of the jobs to be searched. Separate statuses using commas; spaces are not supported. Supported statuses are as follows:

0 all supported statuses

1 submitted

2 waiting for resources

3 resource allocation received

4 submitted to agent

5 running

6 cancel pending

7 canceling allocation

8 waiting for reallocation

10 bound

41 resource allocation failed

42 run failed

43 completed successfully

44 canceled

45 unknown job

46 job not started

48 error

-submitter *submitter*
Specifies the name of the user who submitted the job.

-name *job_definition_name*
Specifies the job name. This option returns the unique job ID, which can be used for retrieving information on and canceling jobs. This option supports the asterisk (*) wildcard character as described below:

as a single parameter
it must be enclosed in inverted commas, for example

```
C:\Program Files\TDWB\bin>jobquery -name "*"
```

This command returns a list of all submitted jobs.

to complete a job name
it does not require inverted commas, for example

```
C:\Program Files\TDWB\bin>jobquery -name batchsub*
```

This command returns a list of all submitted jobs starting with the **batchsub** suffix.

-alias *job_alias*
Specifies the job alias. The job alias is generated at submission time using the **-alias** option. For more information see "jobsubmit command - Submitting jobs" on page 47.

-sdf *submit_date_from*
Specifies a time range starting from the date when the job was submitted. The query is performed starting from the date you specified to the present date, unless the **-sdt** option is specified. Use both the **-sdf** and **-sdt** options to define a specific time range. Specify the date in the dd/MM/yyyy-hh:mm:ss format.

-sdt *submit_date_to*
Specifies a time range starting from the date when the job was submitted. The query is performed starting from the date when the dynamic workload broker database was populated to the date you specified, unless the **-sdf** option is specified. Use both the **-sdf** and **-sdt** options to define a specific time range. Specify the date in the dd/MM/yyyy-hh:mm:ss format.

-jsdf *job_start_date_from*
Specifies a time range starting from the date when the job started. The query is performed starting from the date you specified to the present date, unless the **-jsdt** option is specified. Use both the **-jsdf** and **-jsdt** options to define a specific time range. Specify the date in the dd/MM/yyyy-hh:mm:ss format.

-jsdt *job_start_date_to*
Specifies a time range starting from the date when the job started. The query is performed starting from the date when the dynamic workload broker database was populated to the date you specified, unless the **-jsdf** option is specified. Use both the **-jsdf** and **-jsdt** options to define a specific time range. Specify the date in the dd/MM/yyyy-hh:mm:ss format.

-jedf *job_end_date_from*
Specifies a time range starting from the date when the job completed. The query is performed starting from the date you specified to the present date, unless the **-jedt** option is specified. Use both the **-jedf** and **-jedt** options to define a specific time range. Specify the date in the dd/MM/yyyy-hh:mm:ss format.

-jedt *job_end_date_to*
Specifies a time range starting from the date when the job completed. The query is performed starting from the date when the dynamic workload broker database was populated to the date you specified, unless the **-jedf** option is specified. Use both the **-jedf** and **-jedt** options to define a specific time range. Specify the date in the dd/MM/yyyy-hh:mm:ss format.

-configFile *configuration_file*
Specifies the name and path of a custom configuration file. This option is

optional. If this option is not specified, the default configuration file is assumed. For more information on the configuration file, see "Command-line configuration file" on page 44.

Authorization

The user name and password for the command are defined in the CLIConfig.properties file. To override the setting defined in this file, you can enter the user name and password when typing the command. For more information on the CLIConfig.properties file, see "Command-line configuration file" on page 44.

Return Values

The **jobquery** command returns one of the following values:

0 Indicates that **jobquery** completed successfully.

< > 0 Indicates that **jobquery** failed.

Examples

1. To retrieve the job ID for a job named CLIJSB11, type the following command:

   ```
   jobquery -usr john -pwd BCA12EDF -name CLIJSB11
   ```

   The following output is displayed. The job ID is associated to the Job Identifier key:

   ```
   Call Job Dispatcher to query jobs. There are 10 Jobs found for your request
   Details are as follows:

   Job Name: CLIJSB11
   Job Alias: alias
   Job Identifier: 617c9bf7095787c83e1c36744e569ceb
   Status: FAILED_running
   Job EPR: http://lab135200.romelab.it.ibm.com:955
   /JDServiceWS/services/Job
   Job Submitter Name:
   Submit Time: Tue May 23 15:41:54 CEST 2006
   Start Time: Tue May 23 14:48:09 CEST 2006
   End Time: Tue May 23 14:48:09 CEST 2006
   Job Last Status Message:
   Job Duration: PT0S
   Returncode: 0
   Job Resource Name: LAB237010
   Job Resource Type: ComputerSystem
   ```

2. To retrieve all jobs submitted by test_user in submitted, resource allocation failed, and canceled state, type the following command:

   ```
   jobquery -status 1,3,44 -submitter test_user
   ```

See Also

"jobsubmit command - Submitting jobs" on page 47

jobdetails command - Viewing details on jobs

Use the **jobdetails** command to view details on submitted jobs.

Syntax

jobdetails ?

jobdetails [**-usr** *user_name* **-pwd** *password*] **-id** *job_ID* [**-v**][**-configFile** *configuration_file*]

Description

This command displays details on submitted jobs using the unique ID created at job submission. To retrieve the job ID after submitting the job, use the **jobquery** command specifying the **-name** parameter.

Options

? Displays help information.

-usr *username*
Specifies the username for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the username is not defined in the CLIConfig.properties configuration file (with the `tdwb_user` keyword).

-pwd *password*
Specifies the password for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the password is not defined in the CLIConfig.properties configuration file (with the `tdwb_pwd` keyword).

-id *job_ID*
Specifies the unique job ID created at submission time. This parameter is required.

-v
Displays job details.

-configFile *configuration_file*
Specifies the name and path of a custom configuration file. This parameter is optional. If this parameter is not specified, the default configuration file is assumed. For more information on the configuration file, see "Command-line configuration file" on page 44.

Authorization

The user name and password for the command are defined in the CLIConfig.properties file. To override the setting defined in this file, you can enter the user name and password when typing the command. For more information on the CLIConfig.properties file, see "Command-line configuration file" on page 44.

Return Values

The **jobdetails** command returns one of the following values:
0 Indicates that **jobdetails** completed successfully.
< > 0 Indicates that **jobdetails** failed.

Examples

1. To view run information on a job with ID `617c9bf7095787c83e1c36744e569ceb`, type the following command:

   ```
   jobdetails -id 617c9bf7095787c83e1c36744e569ceb
   ```

   An output similar to the following is displayed:

```
Call Job Dispatcher to get the job properties.
Success return from Job Dispatcher.
Job Identifier: 617c9bf7095787c83e1c36744e569ceb
Job Name: CLIJSB11
Job Alias: alias
Job State: SUBMITTED
Job Submitter: null
Client Notification: http://lab135200.romelab.it.ibm.com:9550
/RAServiceWS/services/Allocation
Job Last Status Message:
Job Submit Time: Tue May 23 15:43:44 CET 2009
Job Start Time: Tue May 23 14:49:51 CET 2009
Job End Time: Tue May 23 14:49:51 CET 2009
Job Duration: PT0S
Job Return Code: 0
Job Resource Name: LAB237010
Job Resource Type: ComputerSystem
Job Usage Metric Name: StartTime
Job Usage Metric Type: null
Job Usage Metric Value: 1148388591000
Job Usage Metric Name: EndTime
Job Usage Metric Type: null
Job Usage Metric Value: 1148388591000
```

2. To submit the job with ID `617l9jw7095787g83f1c36744e569glf` using the configuration parameters specified in the `custom_config.properties` configuration file, type the following command:

   ```
   jobdetails -id 617l9jw7095787g83f1c36744e569glf -configFile
   /opt/test/custom_config.properties
   ```

3. To view the status of a job with ID `617c9bf7095787c83e1c36744e569ceb`, type the following command:

   ```
   jobdetails -id 617c9bf7095787c83e1c36744e569ceb
   ```

   An output similar to the following is displayed:

   ```
   Call Job Dispatcher to get the job properties.
   Success return from Job Dispatcher.
   Job ID: 617c9bf7095787c83e1c36744e569ceb
   Status: SUBMITTED
   ```

4. To view details on the job with ID `617c9bf7095787c83e1c36744e569ceb` using the configuration parameters specified in the `custom_config.properties` configuration file, type the following command:

   ```
   jobdetails -jsdl 617c9bf7095787c83e1c36744e569ceb -configFile
   /opt/test/custom_config.properties
   ```

See Also

- "jobsubmit command - Submitting jobs" on page 47
- "jobquery command - Performing queries on jobs" on page 49

jobcancel command - Canceling jobs

Use the **jobcancel** command to cancel a submitted job.

Syntax

jobcancel ?

jobcancel [**-usr** *user_name* **-pwd** *password*] **-id** *job_ID* [**-configFile** *configuration_file*]

Description

This command cancels the running of submitted jobs using the unique ID created at job submission. To retrieve the job ID after submitting the job, you can use the **jobquery** command specifying the job name.

Options

? Displays help information.

-usr *user_name*
Specifies the username for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the username is not defined in the CLIConfig.properties configuration file (with the `tdwb_user` keyword).

-pwd *password*
Specifies the password for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the password is not defined in the CLIConfig.properties configuration file (with the `tdwb_pwd` keyword).

-id *job_ID*
Specifies the unique job ID created at submission time. This parameter is required.

-configFile *configuration_file*
Specifies the name and path of a custom configuration file. This parameter is optional. If this parameter is not specified, the default configuration file is assumed. For more information on the configuration file, see "Command-line configuration file" on page 44.

Authorization

The user name and password for the command are defined in the CLIConfig.properties file. To override the settings defined in this file, you can enter the user name and password when typing the command. For more information on the CLIConfig.properties file, see "Command-line configuration file" on page 44.

Return Values

The **jobcancel** command returns one of the following values:
0 Indicates that **jobcancel** completed successfully.
< > 0 Indicates that **jobcancel** failed.

Examples

1. To cancel the running of a job with ID `617l9jq7037529f83x1w36185e569fwl`, type the following command:

```
jobcancel -id 617l9jq7037529f83x1w36185e569fwl
```

See Also

"jobsubmit command - Submitting jobs" on page 47

jobstore command - Managing job definitions

Use the **jobstore** command to manage job definitions.

Syntax

jobstore ?

jobstore [**-usr** *user_name* **-pwd** *password*]{[**-create** *jsdl_file*] | [**-update** *jsdl_file*] | [**-del** *job_definition_name*] | [**-get** *job_definition_name*] | [**-queryall**] | [[**-queryname** *job_definition_name*...] [**-querydesc** *job_definition_desc*...] [**-queryowner** *job_definition_owner*...]]} [**-configFile** *configuration_file* }

Description

This command saves and updates JSDL files in the **Job Repository**. JSDL files are saved in the database as job definitions with unique names. After saving JSDL files in the database, you can perform the following operations on job definitions:

- Delete job definitions
- Print job definitions to standard output or save them to a file
- Perform queries on job definitions based on several attributes

You can submit job definitions using the **jobsubmit** command. For more information about the **jobsubmit** command, see "jobsubmit command - Submitting jobs" on page 47.

Options

? Displays help information.

-usr *username*
Specifies the user name for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the user name is not defined in the CLIConfig.properties configuration file (with the `tdwb_user` keyword).

-pwd *password*
Specifies the password for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the password is not defined in the CLIConfig.properties configuration file (with the `tdwb_pwd` keyword).

-create *jsdl_file*
Specifies the name and path to a JSDL file to be saved in the **Job Repository** database. The JSDL file is saved as a job definition. The name for the job definition is saved within the JSDL file and can be only modified by editing the JSDL file. Delete and retrieve (get) operations are performed on the job definition.

-update *jsdl_file*
Specifies the name and path to a JSDL file to be updated in the **Job Repository** database. The JSDL file must be existing in the database.

-del *job_definition_name*
Deletes a job definition from the **Job Repository** database.

-get *job_definition_name*
Prints the JSDL file contained in the job definition to standard output or to a file you specify. You can use this command for performing minor editing on job definitions.

-queryall
Performs a query without any filters. This query returns all job definitions stored in the dynamic workload broker database.

-queryname *job_definition_name*
Performs a search on job definitions based on the job definition name. The job definition name is unique. This parameter is case-sensitive. Wildcards (*, ?) are supported.

-querydesc *job_definition_desc*
Performs a search on job definitions based on the job definition description. Wildcards are supported.

-queryowner *job_definition_owner*
Performs a search on job definitions based on the user who created the job definition.

-configFile *configuration_file*
Specifies the name and path of a custom configuration file. This parameter is optional. If this parameter is not specified, the default configuration file is assumed. For more information about the configuration file, see "Command-line configuration file" on page 44.

Authorization

The user name and password for the command are defined in the CLIConfig.properties file. To override the setting defined in this file, you can enter the user name and password when typing the command. For more information about the CLIConfig.properties file, see "Command-line configuration file" on page 44.

Return Values

The **jobstore** command returns one of the following values:
0 Indicates that **jobstore** completed successfully.
< > 0 Indicates that **jobstore** failed.

Examples

1. To retrieve all jobs created by user `Administrator`, type the following command:
   ```
   jobstore -queryuser Administrator
   ```
2. To update the job branch_update already stored in the Job repository database, type the following command:
   ```
   jobstore -update ../jsdl/branch_update.xml
   ```

See Also

- "jobsubmit command - Submitting jobs" on page 47
- "jobquery command - Performing queries on jobs" on page 49

jobgetexecutionlog command - Viewing job output

Use the **jobgetexecutionlog** command to view the output of a submitted job.

Syntax

jobgetexecutionlog ?

jobgetexecutionlog [**-usr** *user_name* **-pwd** *password*] **-id** *job_ID* **-sizePage** *size Page* **-offset** *offset* [**-configFile** *configuration_file*]

Description

This command displays the job output for submitted jobs using the unique ID created at job submission. To retrieve the job ID after submitting the job, you can use the **jobquery** command specifying the job name. You can also specify the length of the output page to be displayed and the number of the byte in the job output from where you want to start displaying the output.

Options

? Displays help information.

-usr *user_name*
Specifies the username for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the username is not defined in the CLIConfig.properties configuration file (with the `tdwb_user` keyword).

-pwd *password*
Specifies the password for a user authorized to perform operations on the command line. This parameter is required when security is enabled and the password is not defined in the CLIConfig.properties configuration file (with the `tdwb_pwd` keyword).

-id *job_ID*
Specifies the unique job ID created at submission time. This parameter is required.

-sizePage *size Page*
Specifies the number of bytes to be displayed in the job output page.

-offset *offset*
Specifies the number of the first byte to be displayed in the job output page. This option can be used to view large job outputs.

-configFile *configuration_file*
Specifies the name and path of a custom configuration file. This parameter is optional. If this parameter is not specified, the default configuration file is assumed. For more information on the configuration file, see "Command-line configuration file" on page 44.

Authorization

The user name and password for the command are defined in the CLIConfig.properties file. To override the settings defined in this file, you can enter the user name and password when typing the command. For more information on the CLIConfig.properties file, see "Command-line configuration file" on page 44.

Return Values

The **jobgetexecutionlog** command returns one of the following values:

0 Indicates that **jobgetexecutionlog** completed successfully.

< > 0 Indicates that **jobgetexecutionlog** failed.

Examples

1. To view the output of a job with ID 617l9jq7037529f83x1w36185e569fwl displaying the output in pages containing 400® bytes starting from the first byte in the page, type the following command:

   ```
   jobgetexecutionlog -id 617l9jq7037529f83x1w36185e569fwl -sizePage 400 -offset 1
   ```

   The following output is displayed:

   ```
   Call Job Dispatcher to get the output of the job
   Success returned from Job Dispatcher
   Get Execution Log request submitted
   The Execution Log Page requested is:
    al 5
   drwxrwxrwx  7 root root 200 Aug 24 16:39 .
   drwxrwxrwx  8 root root 208 Aug 22 15:11 ..
   drwxrwxrwx  6 root root 248 Aug 22 15:11 eclipse
   -rw-rw-rw-  1 root root 139 Aug 24 16:39 jsdef
   drwxr-xr-x  2 root root 552 Aug 24 16:54 logs
   drwxrwxrwx  5 root root 240 Aug 22 15:11 rcp
   drwxrwxrwx  3 root root  72 Aug 22 15:11 shared
   drwxrwxrwx  3 root root  80 Aug 22 15:11 workspace

   The  file size is:
   381
   ```

See Also

- "jobsubmit command - Submitting jobs" on page 47
- "jobquery command - Performing queries on jobs" on page 49

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

Provides information about the trademarks and registered trademarks of IBM and of the companies with which IBM has trademark acknowledgement agreements.

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.

Intel is a trademark of Intel Corporation in the United States, other countries, or both.

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Product Number: 5698-WSH

Printed in USA

SC23-9856-03