Workload Scheduler

Version 8.6 (Revised June 2012)

User's Guide and Reference

IBM

SC32-1274-12

Workload Scheduler
Version 8.6 (Revised June 2012)

User's Guide and Reference

IBM

SC32-1274-12

Note

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

This edition applies to version 8, release 6, modification level 0 Fix Pack 1 of IBM Tivoli Workload Scheduler (program number 5698-WSH) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC32–1274–11.

Contents

Figures

Tables

About this publication

IBM® Tivoli® Workload Scheduler simplifies systems management across distributed environments by integrating systems management functions. Tivoli Workload Scheduler plans, automates, and controls the processing of your enterprise's entire production workload. The *Tivoli Workload Scheduler User's Guide and Reference* provides detailed information about the command line interface, scheduling language, and utility commands for Tivoli Workload Scheduler.

What is new in this release

For information about the new or changed functions in this release, see the *IBM Tivoli Workload Automation Overview*.

For information about the APARs that this release addresses, see the Tivoli Workload Scheduler Download Document at http://www.ibm.com/support/docview.wss?rs=672&uid=swg24027501.

Publications

Full details of Tivoli Workload Automation 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 more information see Appendix D, "Accessibility," on page 653.

Tivoli technical training

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

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 *IBM Tivoli Workload Scheduler: Troubleshooting Guide*.

Conventions used in this publication

This publication uses several conventions for special terms and actions, operating system-dependent commands and paths, command syntax, and margin graphics.

Typeface conventions

This publication uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations**:)
- Keywords and parameters in text

Italic

- Words defined in text
- Emphasis of words (words as words)
- New terms in text (except in a definition list)
- Variables and values you must provide

`Monospace`

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This publication uses the UNIX convention for specifying environment variables and for directory notation, except where the context or the example path is specifically Windows.

When using the Windows command line, replace **$***variable* with **%** *variable***%** for environment variables and replace each forward slash (**/**) with a backslash (****) in directory paths. The names of environment variables are not always the same in Windows and UNIX environments. For example, %TEMP% in Windows is equivalent to $tmp in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Command syntax

This publication uses the following syntax wherever it describes commands:

Table 1. Command syntax

Syntax convention	Description	Example
Name of command	The first word or set of consecutive characters.	**conman**
Brackets ([ ])	The information enclosed in brackets ([ ]) is optional. Anything not enclosed in brackets must be specified.	[**-file** *definition_file*]
Braces ({ })	Braces ({ }) identify a set of mutually exclusive options, when one option is required.	{**-prompts** | **-prompt** *prompt_name* }
Underscore (_)	An underscore (_) connects multiple words in a variable.	*prompt_name*
Vertical bar (|)	Mutually exclusive options are separated by a vertical bar (|). You can enter one of the options separated by the vertical bar, but you cannot enter multiple options in a single use of the command.	{**-prompts** | **-prompt** *prompt_name* }
Bold	**Bold** text designates literal information that must be entered on the command line exactly as shown. This applies to command names and non-variable options.	**composer add** *file_name*
Italic	Italic text is variable and must be replaced by whatever it represents. In the example to the right, the user would replace file_name with the name of the specific file.	*file_name*
Ellipsis (...)	An ellipsis (...) indicates that the previous option can be repeated multiple times with different values. It can be used inside or outside of brackets.	[**–x** *file_name*]... An ellipsis outside the brackets indicates that **–x** *file_name* is optional and may be repeated as follows: *–x file_name1 –x file_name2–x file_name3* [**–x** *file_name*...] An ellipsis inside the brackets indicates that **–x** *file_name* is optional, and the file variable can be repeated as follows: *–x file_name1 file_name2 file_name3* **–x** *file_name* [**–x** *file_name*]... An ellipsis used with this syntax indicates that you must specify **–x** *file_name* at least once.

Chapter 1. Tivoli Workload Scheduler overview

IBMTivoli Workload Scheduler provides you with the ability to manage your production environment and automate many operator activities. Tivoli Workload Scheduler manages job processing, resolves interdependencies, and launches and tracks jobs. Because jobs start as soon as their dependencies are satisfied, idle time is minimized and throughput is significantly improved. If a job fails, Tivoli Workload Scheduler manages the recovery process with little or no operator intervention.

This chapter is divided into the following sections:
- "Understanding basic concepts"
- "Tivoli Workload Scheduler user interfaces" on page 18
- "Starting production" on page 19

Understanding basic concepts

This section describes the basic concepts of Tivoli Workload Scheduler and is divided into the following sections:
- "Tivoli Workload Scheduler database objects"
- "The Tivoli Workload Scheduler network" on page 12
- "Configuring your Tivoli Workload Scheduler runtime environment" on page 13
- "Defining scheduling activities using Tivoli Workload Scheduler" on page 13
- "Managing production scheduling activities with Tivoli Workload Scheduler" on page 17

Tivoli Workload Scheduler database objects

This section introduces the Tivoli Workload Scheduler database objects that you work with. The following database objects are described:
- Job, see "Job"
- Job stream, see "Job stream" on page 2
- Run cycle, see "Run cycle" on page 2
- Calendar, see "Calendar" on page 3
- Prompt, see "Prompt" on page 4
- Workstation, see "Workstation" on page 4
- Workstation class, see "Workstation class" on page 7
- Domain, see "Domain" on page 8
- Event rule, see "Event rule" on page 11
- Resource, see "Resource" on page 11
- Parameter, see "Parameter" on page 11
- Variable table, see "Variable table" on page 11

Job

A *job* is a unit of work specifying an action, such as a weekly data backup, to be performed on specific workstations in the Tivoli Workload Scheduler network. In a Tivoli Workload Scheduler distributed environment, jobs can be defined either independently from job streams or within a job stream definition.

Job types can be divided between existing Tivoli Workload Scheduler jobs and job types with advanced options. The existing job types are generic scripts or commands you customize according to your needs. The job types with advanced options are jobs designed to perform specific operations, such as database, file transfer, Java, and web service operations. You schedule these jobs types only on dynamic agents, pools and dynamic pools.

If you want to leverage the dynamic capability when scheduling job types with advanced options, you schedule them on pools and dynamic pools, which assign dynamically the job to the best available resource. If you are interested only in defining job types with advanced options, without using the dynamic scheduling capability, you schedule these jobs on a specific dynamic agent, on which the job runs statically.

Regardless whether the Tivoli Workload Scheduler engine is distributed or z/OS based, you can define locally a *shadow job* to map a remote job instance running on a different Tivoli Workload Scheduler engine.

For information about how to define jobs, see "Job definition" on page 145.

For information about how to define workstations, see "Workstation definition" on page 127.

Job stream

A *job stream* is a sequence of jobs to be run, together with times, priorities, and other dependencies that determine the order of processing. Each job stream is assigned a time to run, represented by run cycle with type calendar, set of dates, or repetition rates.

Dependencies in a distributed environment:
You can have dependencies between both jobs and job streams. They can be:

Internal dependencies
These are dependencies established between jobs belonging to the same job stream.

External dependencies
These are dependencies between job streams, or between job streams and jobs belonging to other job streams, or between jobs belonging to different job streams.

Internetwork dependencies
These are dependencies on jobs or job streams running in another Tivoli Workload Scheduler network. Internetwork dependencies require a network agent workstation to communicate with the external Tivoli Workload Scheduler network.

Dependencies on resources are supported by Tivoli Workload Scheduler both in the distributed and in the z/OS environments.

For information about how to define job streams, see "Job stream definition" on page 183.

Run cycle

A ***run cycle*** specifies the days that a job stream is scheduled to run. A cycle is defined for a specific job stream and cannot be used by multiple job streams. You can specify the following types of run cycle:

simple
A specific set of user-defined days a job stream is run. A simple run cycle is defined for a specific job stream and cannot be used by other job streams.

daily A run cycle that specifies that the job stream runs according to a day frequency and type that you set. For example, it might run daily, every three days, or just on working days.

weekly
A run cycle that specifies the days of the week that a job stream is run. For example, a job stream can be specified to run every Monday, Wednesday, and Friday using a weekly run cycle.

monthly
A run cycle that specifies that the job stream runs according to a monthly day or date that you set. For example, it might run every 1st and 2nd day of the month, every two months, or every 1st Monday and 2nd Tuesday of the month, every three months.

yearly A run cycle that specifies that a job stream runs, for example, yearly or every three years.

offset-based
A run cycle that uses a combination of user-defined periods and offsets. For example, an offset of 3 in a period of 15 days is the third day from the beginning of the period. It is more practical to use offset-based run cycles when the cycle is based on cyclic periods. This term is only used as such in Tivoli Workload Scheduler for z/OS, but the concept applies also to the distributed product.

rule-based
A run cycle that uses rules based on lists of ordinal numbers, types of days, and common calendar intervals (or period names in Tivoli Workload Scheduler for z/OS). For example, the last Thursday of every month. Rule-based run cycles are based on conventional periods, such as calendar months, weeks of the year, and days of the week. In Tivoli Workload Scheduler for z/OS, run cycles can also be based on periods that you define, such as a semester. This term is only used as such in Tivoli Workload Scheduler for z/OS, but the concept applies also to the distributed product. You can also specify a rule to establish when a job stream runs if it falls on a free day.

Any of these run cycle types can be either inclusive or exclusive; that is:

inclusive
A run cycle that specifies the days and times that a job stream is scheduled to run. Inclusive run cycles give precedence to inclusive run cycles.

exclusive
A run cycle that specifies the days and times that a job stream cannot be run. Exclusive run cycles take precedence over inclusive run cycles.

Calendar

A *calendar* is a list of dates which define if and when a job stream runs.

A calendar can also be designated for use as a *non-working days* calendar in a job stream. A non-working days calendar is a calendar that is assigned to a job stream to represent the days when the job stream and its jobs do not run. It can also be used to designate Saturdays or Sundays, or both, as workdays. By convention many users define a non-working days calendar named *holidays*, where habitually Saturday and Sunday are specified as non-working days.

For information about how to define calendars, see "Calendar definition" on page 174.

Prompt

A *prompt* identifies a textual message that is displayed to the operator and halts processing of the job or job stream until an affirmative answer is received (either manually from the operator or automatically by an event rule action). After the prompt is replied to, processing continues. You can use prompts as dependencies in jobs and job streams. You can also use prompts to alert an operator that a specific task was performed. In this case, an operator response is not required.

There are three types of prompts:

global or named
A prompt that is defined in the database as a scheduling object. It is identified by a unique name and can be used by any job or job stream.

local or ad-hoc
A prompt that is defined within a job or job stream definition. It does not have a name, and it is not defined as a scheduling object in the database, therefore it cannot be used by other jobs or job streams.

recovery or abend
A special type of prompt that you define to be used when a job ends abnormally. The response to this prompt determines the outcome of the job or job stream to which the job belongs. A recovery prompt can also be associated to an action and to a special type of job called a *recovery job*.

For information about how to define prompts, see "Prompt definition" on page 181

Workstation

Note: This section provides information relating to the use of workstations for scheduling jobs and job streams. If, instead, you want to learn about workstations because you are planning your network, you can find the information you need in the *Tivoli Workload Scheduler Planning and Installation Guide*.

The computer system where you run your jobs and job streams is called a *workstation*. When you define a job or job stream in the Tivoli Workload Scheduler database you identify the workstation definitions for the physical or virtual computer systems on which your job is scheduled to run. Workstations can be grouped logically into *workstation classes* and organized hierarchically into *domains*, managed by *domain managers*.

For more information about workstation classes, see "Workstation class" on page 7, and for domains, see "Domain" on page 8.

When you create a workstation definition for a system in your network you define a set of characteristics that uniquely identify the system and affect the way jobs run on it. Some examples of these characteristics are the IP address of the workstation, the position behind a firewall or not , the secure or unsecure communication, the time zone where the workstation is located, and the identity of its domain manager.

Workstations in the Tivoli Workload Scheduler scheduling network cannot only perform job and job stream processing, but can also have other roles. When your network was designed, these roles were assigned to these workstations to suit the specific needs of your business. The full list of workstation roles is as follows:

Master domain manager
A workstation acting as the management hub for the network. It manages all your scheduling objects. The master domain manager workstation must be installed as such. This workstation is recorded in the Tivoli Workload Scheduler database as **master**.

Backup master domain manager
A workstation which can act as a backup for the master domain manager, when problems occur. It is effectively a master domain manager, waiting to be activated. Its use is optional. Learn more about switching to a backup master domain manager in the *Tivoli Workload Scheduler Administration Guide*. The master domain manager workstation must be installed as such. This workstation is recorded in the Tivoli Workload Scheduler database as **fta**.

Domain manager
A workstation that controls a domain, and shares management responsibilities for part of the Tivoli Workload Scheduler network. It is installed as an agent, and then configured as a domain manager workstation when you define the workstation in the database. This workstation is recorded in the Tivoli Workload Scheduler database as **manager**.

Dynamic domain manager
An installed component in a distributed Tivoli Workload Scheduler network that is the management hub in a domain. All communication to and from the dynamic agents in the domain is routed through the dynamic domain manager. When you install a dynamic domain manager the workstation types listed below are created in the database:

fta Fault-tolerant agent component manually configured as domain manager

broker
Broker server component

agent Dynamic agent component

Backup dynamic domain manager
A workstation which can act as a backup for the dynamic domain manager, when problems occur. It is effectively a dynamic domain manager, waiting to be activated. Its use is optional. Learn more about switching to a backup dynamic domain manager in the *Tivoli Workload Scheduler Administration Guide*. When you install a dynamic domain manager the workstation types listed below are created in the database:

fta Fault-tolerant agent component.

broker
Broker server component

agent Dynamic agent component

Fault-tolerant agent
A workstation that receives and runs jobs. If there are communication problems with its domain manager, it can run jobs locally. It is installed as an agent, and then configured as a fault-tolerant agent workstation when

you define the workstation in the database. This workstation is recorded in the Tivoli Workload Scheduler database as **fta**.

Standard agent
A workstation that receives and runs jobs only under the control of its domain manager. It is installed as an agent, and then configured as a standard agent workstation when you define the workstation in the database. This workstation is recorded in the Tivoli Workload Scheduler database as **s-agent**.

Extended agent
A workstation where a Tivoli Workload Scheduler for Applications access method has been installed as a bridge so that you can schedule jobs in the SAP R/3, Oracle E-Business Suite, PeopleSoft, z/OS, or custom applications. It must be physically hosted by a master domain manager, domain manager, standard agent, or a fault-tolerant agent (up to 255 extended agents per fault-tolerant agent) and then defined as an extended agent in the database. For more information, see the *Tivoli Workload Scheduler for Applications User's Guide*. This workstation is recorded in the Tivoli Workload Scheduler database as **x-agent**.

Workload broker
A workstation that runs both existing job types and job types with advanced options. It is the broker server installed with the master domain manager and the dynamic domain manager. It can host one or more of the following workstations:

- extended agent
- remote engine
- pool
- dynamic pool
- agent. This definition includes the following agents:
  - dynamic agent
  - Tivoli Workload Scheduler for z/OS agent
  - agent for z/OS

  For more information about the dynamic agent and Tivoli Workload Scheduler for z/OS agent, see *Scheduling Workload Dynamically*. For more information about the agent for z/OS, see *Scheduling with the agent for z/OS*.

This workstation is recorded in the Tivoli Workload Scheduler database as **broker**.

Dynamic agent
A workstation that manages a wide variety of job types, for example, specific database or FTP jobs, in addition to existing job types. This workstation is automatically created and registered in the Tivoli Workload Scheduler database when you install the dynamic agent. The dynamic agent is hosted by the workload broker workstation. Because the installation and registration processes are performed automatically, when you view the dynamic agent in the Dynamic Workload Console, it results as updated by the Resource Advisor Agent. You can group dynamic agents in pools and dynamic pools. This workstation is recorded in the Tivoli Workload Scheduler database as **agent**.

Pool A logical workstation that groups a set of dynamic agents with similar hardware or software characteristics to submit jobs to. Tivoli Workload Scheduler balances the jobs among the dynamic agents within the pool and automatically reassigns jobs to available dynamic agents if a dynamic agent is no longer available. To create a pool of dynamic agents in your Tivoli Workload Scheduler environment, define a workstation of type **pool** hosted by the workload broker workstation, then select the dynamic agents you want to add to the pool. You can define the pool using the Dynamic Workload Console or the **composer** command. This workstation is recorded in the Tivoli Workload Scheduler database as **pool**.

Dynamic pool
A logical workstation that groups a set of dynamic agents,which is dynamically defined based on the resource requirements you specify and hosted by the workload broker workstation. For example, if you require a workstation with low CPU usage and Windows installed to run your job, you specify these requirements using the Dynamic Workload Console or the **composer** command. When you save the set of requirements, a new workstation is automatically created in the Tivoli Workload Scheduler database. This workstation maps all the dynamic agents in your environment that meet the requirements you specified. The resulting pool is dynamically updated whenever a new suitable dynamic agent becomes available. Jobs scheduled on this workstation automatically inherit the requirements defined for the workstation. This workstation is hosted by the workload broker workstationand recorded in the Tivoli Workload Scheduler database as **d-pool**.

Remote engine
A workstation that manages the exchange of information about cross dependencies resolution between your environment and a remote Tivoli Workload Scheduler for z/OS engine (controller) or a Tivoli Workload Scheduler engine (master domain manager or backup master domain manager). This workstation is hosted by the workload broker workstation and recorded in the Tivoli Workload Scheduler database as **rem-eng**.

Note: If you plan to change the workstation types, consider the following rules:
- you can change fault-tolerant agent, standard agent, extended agent, domain manager and dynamic workload broker workstations to any workstation type, with the exception of dynamic agent, pool, dynamic pool, and remote engine.
- you cannot change the type of dynamic agent, pool, dynamic pool, and remote engine.

For information about how to define workstations, see “Workstation definition” on page 127.

Workstation class

Workstations can be grouped into classes. A *workstation class* is a group of workstations with similar job scheduling characteristics. Any number of workstations can be grouped in a class, and a workstation can be in many classes. Jobs and job streams can be assigned to run on a specific workstation class and this makes the running of jobs and job streams across multiple workstations easier.

For example, you can set up the following types of workstation classes:
- Workstation classes that group workstations according to your internal departmental structure, so that you can define a job to run on all the workstations in a department

- Workstation classes that group workstations according to the software installed on them, so that you can define a job to run on all the workstations that have a particular application installed
- Workstation classes that group workstations according to the role of the user, so that you can define a job to run on all the workstations belonging to, for example, managers

In this example, an individual workstation can be in one workstation class for its department, another for its user, and several for the software installed on it.

Workstations can also be grouped into domains. This is done when your network is set up. The domain name is not one of the selection criteria when choosing where to run a job, so you might need to mirror your domain structure with workstation classes if you want to schedule a job to run on all workstations in a domain.

For more information about domains, see “Domain”

For more information about how to define workstation classes, see “Workstation class definition” on page 142.

Domain

All workstations in a distributed Tivoli Workload Scheduler network are organized in one or more *domains*, each of which consists of one or more agents and a domain manager acting as the management hub. Most communication to and from the agents in the domain is routed through the domain manager.

All networks have a master domain where the domain manager is the master domain manager. It maintains the database of all scheduling objects in the domain and the central configuration files. The master domain manager generates the plan and creates and distributes the Symphony file. In addition, logs and reports for the network are maintained on the master domain manager.

You can organize all agents in your network in a single domain, or in multiple domains.

Single-domain networks

A single domain network consists of a master domain manager and any number of agents. The following shows an example of a single domain network. A single domain network is well suited to companies that have few locations and business functions. All communication in the network is routed through the master domain manager. With a single location, you are concerned only with the reliability of your local network and the amount of traffic it can handle.

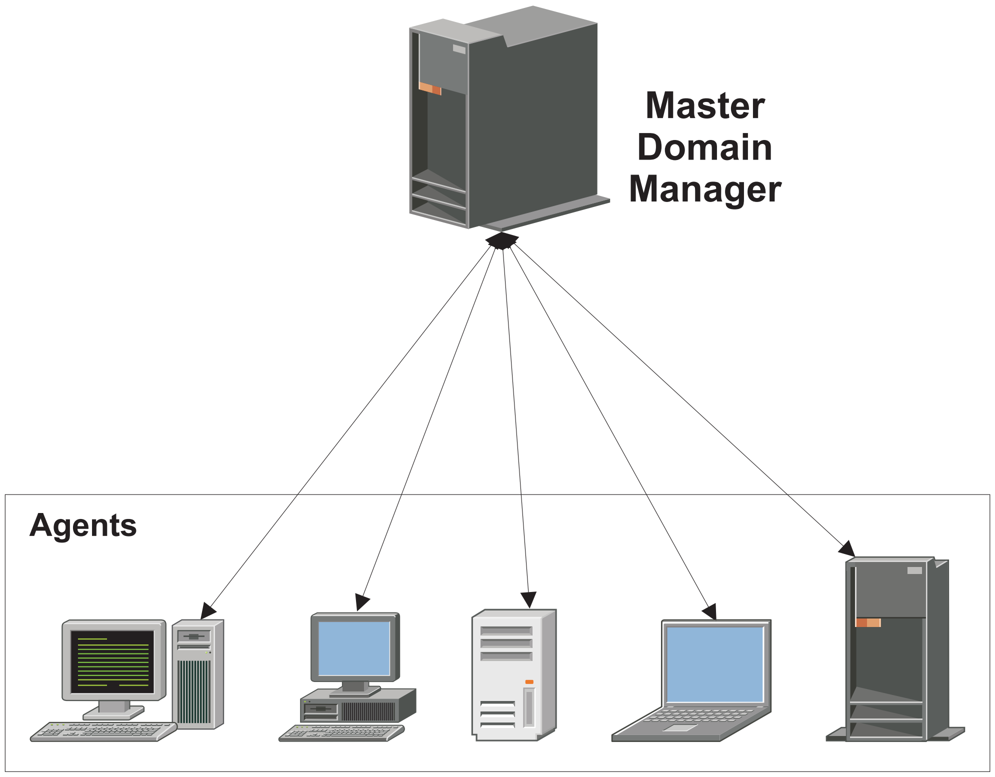


Figure 1. Single-domain network

Multiple-domain network
Multiple domain networks are especially suited to companies that span multiple locations, departments, or business functions. A multiple domain network consists of a master domain manager, any number of lower tier domain managers, and any number of agents in each domain. Agents communicate only with their domain managers, and domain managers communicate with their parent domain managers. The hierarchy of domains can go down to any number of levels.

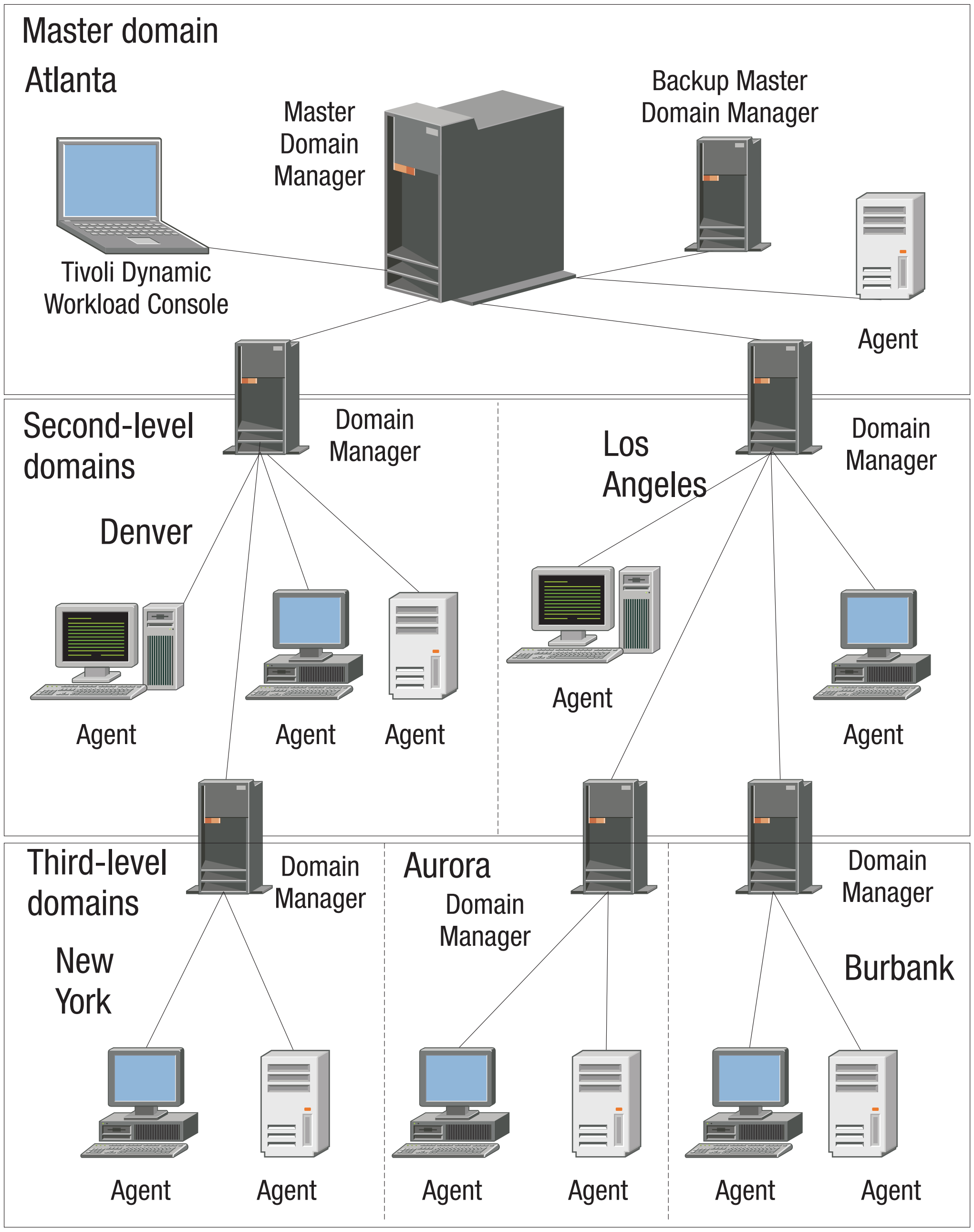


Figure 2. Multiple-domain network

In this example, the master domain manager is located in Atlanta. The master domain manager contains the database files used to document the scheduling objects, and distributes the Symphony file to its agents and the domain managers in Denver and Los Angeles. The Denver and Los Angeles domain managers then distribute the Symphony file to their agents and subordinate domain managers in New York, Aurora, and Burbank. The master domain manager in Atlanta is responsible for broadcasting inter-domain information throughout the network.

All communication to and from the Boulder domain manager is routed through its parent domain manager in Denver. If there are schedules or jobs in the Boulder domain that are dependent on schedules or jobs in the Aurora domain, those dependencies are resolved by the Denver domain manager. Most inter-agent dependencies are handled locally by the lower tier domain managers, greatly reducing traffic on the network.

You can change the domain infrastructure dynamically as you develop your network. To move a workstation to a different domain, just requires you to change the domain name in its database definition.

Tip:	You cannot schedule jobs or job streams to run on all workstations in a domain by identifying the domain in the job or job stream definition. To achieve this, create a *workstation class* that contains all workstations in the domain.

For more information about workstation classes, see "Workstation class" on page 7

For information about how to define domains, see "Domain definition" on page 144.

Event rule

An *event rule* defines a set of actions that run when specific event conditions occur. An event rule definition correlates events and trigger actions.

For information about how to define event rules, see "Defining event rules" on page 114.

Resource

A *resource* is either a physical or logical system resource that you use as a dependency for jobs and job streams. A job or job stream with a resource dependency cannot start to run until the required quantity of the defined resource is available.

For information about how to define resources, see "Resource definition" on page 182.

Parameter

A *parameter* is an object to which you assign different values to be substituted in jobs and job streams, either from values in the database or at run time. Parameters are useful when you have values that change depending on your job or job stream. Job and job stream definitions that use parameters are updated automatically with the value at the start of the production cycle. Use parameters as substitutes for repetitive values when defining jobs and job streams. For example, using parameters for user logon and script file names in job definitions and for file and prompt dependencies allows the use of values that can be maintained centrally in the database on the master.

For more information about how to define parameters, see "Variable and parameter definition" on page 175.

Variable table

A *variable table* is a table containing multiple variables and their values. All global parameters, now called *variables*, are contained in at least one variable table.

You are not required to create variable tables to be able to use variables, because the scheduler provides a default variable table.

However, you might want to define a variable with the same name, but different values, depending on when and where it is used. You do this by assigning different values to the same variable in different variable tables. You can then use the same variable name in different job definitions and when defining prompts and file dependencies. Variable tables can be assigned at run cycle, job stream, and workstation level.

Variable tables can be particularly useful in job definitions when a job definition is used as a template for a job that belongs to more than one job stream. For example, you can assign different values to the same variable and reuse the same job definition in different job streams.

For information about how to define variable tables, see "Variable table definition" on page 179.

The Tivoli Workload Scheduler network

A Tivoli Workload Scheduler network consists of a set of linked ***workstations*** on which you perform batch job processing using Tivoli Workload Scheduler management capabilities.

Workstations communicate using TCP/IP links, and a store-and-forward technology to maintain consistency and fault-tolerance across the network. This means that if a workstation is not linked, all the information is stored in the messages file and only sent when the link is reestablished.

The Tivoli Workload Scheduler network consists of one or more domains, each having a ***domain manager*** workstation acting as a management hub, and one or more ***agent*** workstations.

There are four types of agent: ***standard***, ***fault-tolerant***, ***extended***, and ***workload broker***. Standard and fault-tolerant agents can be defined on UNIX and Windows computers. Extended agents are logical definitions, each hosted by a physical workstation, and are used to perform job processing where an agent is not installed. For example, extended agents are available for Peoplesoft, SAP R/3, z/OS®, CA-7, JES, OPC, Oracle EBS, and VMS but you can also install them on UNIX and Windows systems. Workload broker agents are workstations that manage the lifecycle of Tivoli Workload Scheduler Workload Broker type jobs in dynamic workload broker.

Another type of workstation that you can define in your network is a ***remote engine workstation***. This kind of workstation is used to manage the communication with a remote Tivoli Workload Scheduler engine, either distributed or z/OS based, to manage dependencies for local jobs from jobs defined on the remote engine. For more information, see Chapter 19, "Defining and managing cross dependencies," on page 559.

For information about workstations, see "Workstation definition" on page 127.

In the hierarchical Tivoli Workload Scheduler topology, the ***master domain manager*** is the domain manager of the topmost domain. All production setup tasks and the generation of the ***production plan*** are performed on the master domain manager. A production plan contains all job management activities to be performed across the Tivoli Workload Scheduler network during a specific time frame. A copy of the production plan is distributed from the master domain manager to the other workstations. On each workstation Tivoli Workload Scheduler launches and tracks its own jobs, and sends job processing status to the master domain manager.

For more information about Tivoli Workload Scheduler plan management capabilities, refer to Chapter 4, “Managing the production cycle,” on page 49.

Configuring your Tivoli Workload Scheduler runtime environment

This section gives you a high level overview of how you can configure your Tivoli Workload Scheduler runtime environment.

Configuring properties

You can set two types of properties to configure your Tivoli Workload Scheduler runtime environment, properties that are set on the master domain manager and affect processing on all workstations in the Tivoli Workload Scheduler network, and properties that are set locally on a workstation and affect processing on that workstation only. The former are managed using the Tivoli Workload Scheduler command line program named **optman**, and the latter you define locally on the workstation by customizing the configuration files **`useropts`**, **`localopts`**, and **`jobmanrc`**.

For more information on how to use the **optman** command line to manage global options and about local options defined in the `localopts` file, refer to *IBM Tivoli Workload Scheduler Administration Guide*.

For more information about the local options defined in the `useropts` file, refer to “Setting up options for using the user interfaces” on page 45.

Configuring security

Each time you run a Tivoli Workload Scheduler program, or invoke a Tivoli Workload Scheduler command, security information is read from a special file, the ***Security file***, to determine your user capabilities. This file contains one or more *user definitions*. A user definition is a group of one or more users who are either allowed or denied to perform specific actions against specific scheduling object types.

The main Tivoli Workload Scheduler user, ***TWS_user***, is defined at installation time in the security file. That user ID can be used to complete the setup procedure, to set properties, and to manage user definitions inside the security file. You can modify the security file at any time to meet your system requirements.

For more information about managing user authorizations, refer to *Tivoli Workload Scheduler Administration Guide*.

Defining scheduling activities using Tivoli Workload Scheduler

To perform scheduling activities using Tivoli Workload Scheduler you must first define the environment you want to manage in terms of *scheduling objects* and in terms of rules to be applied when scheduling operations to run on these objects. This information is stored by Tivoli Workload Scheduler in a DB2® or Oracle Relational Data Base, from now on called the ***database***.

In addition to the definitions of scheduling objects, such as jobs, job streams, resources, workstations, and so on, the database also contains statistics about processed jobs and job streams, as well as information about the user who created an object and when an object was last modified. You can manage the scheduling object definitions in the database using either the Tivoli Workload Scheduler command-line program named **composer** or the graphical user interfaces, the

Dynamic Workload Console. You can retrieve statistics or history information about processed jobs and job streams in the database using:

- The Tivoli Workload Scheduler **report utilities** from the command line.
- The Dynamic Workload Console.
- The database views.

For more information about how to define scheduling objects, see Chapter 8, "Defining objects in the database," on page 125.

For more information about report utility commands, refer to Chapter 15, "Getting reports and statistics," on page 491.

For more information about the Dynamic Workload Console, refer to the corresponding documentation.

For more information on database views, refer to *IBM Tivoli Workload Scheduler: Database Views*.

Controlling job and job stream processing

You can control how jobs and job streams are processed by setting one or more rules from the following:

Defining dependencies

A ***dependency*** is a prerequisite that must be satisfied before processing can proceed. You can define dependencies for both jobs and job streams to ensure the correct order of processing. Within your Tivoli Workload Scheduler distributed scheduling environment you can choose from four different types of dependencies:

- ***On completion of jobs and job streams***: a job or a job stream, named *successor*, must not begin processing until other jobs and job streams, named *predecessor*, have completed successfully. For more information, see "follows" on page 198.
- ***Resource***: a job or a job stream needs one or more resources available before it can begin to run. For more information, refer to "needs" on page 205.
- ***File***: a job or a job stream needs to have access to one or more files before it can begin to run. For more information, refer to "opens" on page 211.
- ***Prompt***: a job or a job stream needs to wait for an affirmative response to a prompt before it can begin processing. For more information, refer to "Prompt definition" on page 181 and "prompt" on page 214.

You can define up to 40 dependencies for a job or job stream.

In a Tivoli Workload Scheduler network, dependencies can cross workstation boundaries. For example, you can make `job1`, which runs on your Tivoli Workload Scheduler local environment `site1`, dependent on the successful completion of `job2`, which runs on a remote Tivoli Workload Scheduler environment `site2`. The remote scheduling environment can be either Tivoli Workload Scheduler for z/OS engines (controller) or another Tivoli Workload Scheduler engines (master domain manager). Two types of dependencies implement such requirement:

Internetwork dependency

It is a simple and distributed based implementation. Use this type of dependency when:

- The local Tivoli Workload Scheduler environment is distributed.
- You want to search for a remote predecessor job instance only in the plan currently running (production plan) on the remote environment.

- You need to match a predecessor instance in the remote plan, not *that* specific predecessor instance.
- You can wait for the polling interval to expire before being updated about the remote job status transition.
- You do not mind using different syntaxes and configurations based on whether the remote Tivoli Workload Scheduler environment is distributed rather than z/OS.
- You do not mind using a proprietary connection protocol for communicating with the remote engine.

For more information, see Chapter 18, "Managing internetwork dependencies," on page 549.

Cross dependency
It is a more comprehensive and complete implementation. Use this type of dependency when:

- Your local Tivoli Workload Scheduler environment can be either distributed or z/OS.
- You want to search for the remote predecessor instance also among the scheduled instances that are not yet included in the plan currently running on the remote engine.
- You want to match a precise remote predecessor instance in the remote engine plan. To do this you can use different out-of-the-box matching criteria.
- You want your dependency to be updated as soon as the remote job instance changes status. To do this the product uses an asynchronous notifications from the remote engine to the local engine.
- You want to use the same syntax and configuration regardless of whether the local Tivoli Workload Scheduler environment is distributed or z/OS.
- You want to use HTTP or HTTPS connections for communicating with the remote engine.

For more information, see Chapter 19, "Defining and managing cross dependencies," on page 559.

Setting time constrains

Time constraints can be specified for both jobs and job streams. For a specific run cycle you can specify the time that processing begins, by using the keyword **at**, or the time after which processing is no longer started, by using the keyword **until**. By specifying both, you define a time window within which a job or job stream runs. Both **at** and **until** represent time dependencies.

Another time setting that can be specified is the **schedtime**; it indicates the time that is referred to when calculating jobs and job streams dependencies. You can also specify a ***repetition rate***; for example, you can have Tivoli Workload Scheduler launches the same job every 30 minutes between 8:30 a.m. and 1:30 p.m.

For more information, refer to "at" on page 188, "deadline" on page 191, "every" on page 193, "schedtime" on page 215, and "until" on page 218.

Setting job priority and workstation fence

Tivoli Workload Scheduler has its own queuing system, consisting of levels of ***priority***. Assigning a priority to jobs and job streams gives you added control over their precedence and order of running.

The ***job fence*** provides another type of control over job processing on a workstation. When it is set to a priority level, it only allows jobs and job streams whose priority exceeds the job fence to run on that workstation. Setting the fence to 40, for example, prevents jobs with priorities of 40 or less from being launched.

For more information, refer to “fence” on page 331 and “priority” on page 213.

Setting limits

The ***limit*** provides a means of setting the highest number of jobs that Tivoli Workload Scheduler is allowed to launch. You can set a limit:

- In the job stream definition using the ***job limit*** argument
- In the workstation definition using the ***limit cpu*** command

Setting the limit on a workstation to 25, for example, allows Tivoli Workload Scheduler to have no more than 25 jobs running concurrently on that workstation.

For more information, refer to “limit cpu” on page 334, and “limit sched” on page 335.

Defining resources

You can define ***resources*** to represent physical or logical assets on your system. Each resource is represented by a name and a number of available units. If you have three tape units, for example, you can define a resource named `tapes` with three available units. A job that uses two units of the `tapes` resource would then prevent other jobs requiring more than the one remaining unit from being launched. However because a resource is not strictly linked to an asset, you can use a mock resource as a dependency to control job processing.

For more information, refer to “Resource definition” on page 182.

Asking for job confirmation

There might be scenarios where the completion status of a job cannot be determined until you have performed some tasks. You might want to check the results printed in a report, for example. In this case, you can set in the job definition that the job requires *confirmation*, and Tivoli Workload Scheduler waits for your response before marking the job as successful or failed.

For more information, refer to “confirm” on page 322.

Defining job recovery actions

When you schedule a job, you can specify the type of recovery you want performed by Tivoli Workload Scheduler if the job fails. The predefined recovery options are:

- Continue with the next job.
- Stop and do not start the next.
- Run the failed job again.

In addition, you can specify other actions to be taken in terms of recovery jobs and recovery prompts. For example, if a job fails, you can have Tivoli Workload Scheduler automatically run a recovery job, issue a recovery prompt that requires an affirmative response, and then run the failed job again.

For more information, refer to “Job” on page 636.

Managing production scheduling activities with Tivoli Workload Scheduler

Each time a new production plan is generated, Tivoli Workload Scheduler selects the job streams that run in the time window specified for the plan, and carries forward uncompleted job streams from the previous production plan. All the required information are written in a file, named ***Symphony***, which is continually updated while processing to indicate work completed, work in progress, and work to be done. The Tivoli Workload Scheduler **conman** (Console Manager) command-line program is used to manage the information in the `Symphony` file. The **conman** command-line program can be used to:

- Start and stop Tivoli Workload Scheduler control processes.
- Display the status of jobs and job streams.
- Alter priorities and dependencies.
- Alter the job fence and job limits.
- Rerun jobs.
- Cancel jobs and job streams.
- Submit new jobs and job streams.
- Reply to prompts.
- Link and unlink workstations in the Tivoli Workload Scheduler network.
- Modify the number of available resources.

Automating workload using event rules

Beside doing plan-based job scheduling, you can automate workload based on demand with the aid of event rules. The objective of event rules is to carry out a predefined set of actions in response to specific events affecting Tivoli Workload Scheduler and non-Tivoli Workload Scheduler objects.

With respect to Tivoli Workload Scheduler objects, the product provides a plug-in that you can use to detect the following events:

- A specific job or job stream:
  – Changes status
  – Is beyond its latest start time
  – Is submitted
  – Is cancelled
  – Is restarted
  – Becomes late
- A certain workstation:
  – Changes status
  – Changes its link status from its parent workstation
  – Changes its link status from its child workstation
- A specific prompt is displayed or replied to
- The embedded application server on a certain workstation is started or stopped

When any of these events takes place, any of the following actions can be triggered:

- Submit a job stream, a job, or a task
- Reply to a prompt
- Run non-Tivoli Workload Scheduler commands
- Log an operator message
- Notify users via email
- Send messages to Tivoli Enterprise Console

You can also define and run event rules that act either on the detection of one or more of these events or on a sequence or set of these events not completing within a specific length of time.

More information is available on Chapter 7, "Running event-driven workload automation," on page 107.

Tivoli Workload Scheduler user interfaces

A combination of graphical and command-line and API interface programs are provided to work with Tivoli Workload Scheduler. In particular, the command-line interface is available for certain advanced features which are not available in the graphical user interface. The available Tivoli Workload Scheduler user interface programs are:

Dynamic Workload Console

A Web-based user interface available for viewing and controlling scheduling activities in production on both the Tivoli Workload Scheduler distributed and z/OS environments. With the Dynamic Workload Console you can use any supported browser to access the Tivoli Workload Scheduler environment from any location in your network.

You can use the Dynamic Workload Console to:

- Define scheduling objects in the Tivoli Workload Scheduler database
- Browse and manage scheduling objects involved in current plan activities
- Create and control connections to Tivoli Workload Scheduler environments
- Submit jobs and job streams in production
- Set user preferences
- Create and manage event rules
- Define and manage mission-critical jobs

Dynamic Workload Console must be installed on a server that can reach the Tivoli Workload Scheduler nodes using network connections. See the Tivoli Workload Scheduler *Planning and Installation* guide for information.

composer

A command-line program used to define and manage scheduling objects in the database. This interface program and its use are described in Chapter 8, "Defining objects in the database," on page 125 and Chapter 9, "Managing objects in the database - composer," on page 235.

conman

A command-line program used to monitor and control the Tivoli Workload Scheduler production plan processing. This interface program is described in Chapter 10, "Managing objects in the plan - conman," on page 291.

Java™ API and plugins

A set of available classes and methods running in a JAVA environment that you use to create your custom interface to manage scheduling objects in the database and in the plan. This API cannot be used to create your custom interface to set global options. In addition, you can use and modify a set of plug-ins that perform specific tasks, or create your own plug-ins. The API is available through a Software Development Kit, which is part of the product. For more information and to learn how to access the documentation of the API and plug-ins, refer to *IBM Tivoli Workload Scheduler Developer's Guide: Software Development Kit (Integration Workbench)*.

optman
A command-line program used to manage the settings that affect the entire Tivoli Workload Scheduler environment. These settings, also called global options, are stored in the database. This interface program is described in the *Tivoli Workload Scheduler Administration Guide*.

planman
A command-line program used to manage the Tivoli Workload Scheduler planning capability. This interface program is described in "Planman command line" on page 73.

Web Services Interface
An interface that provides you with a Web Services based access mechanism to a subset of functionality used to manage jobs and job streams in the plan. It does not allow you to manage the plan, to set global options, to manage objects in the database. For more information refer to *IBM Tivoli Workload Scheduler Developer's Guide: Web Services*.

You must install the Tivoli Workload Scheduler Command Line Client feature on fault-tolerant agents and systems outside the Tivoli Workload Scheduler network to use the **composer** and **optman** command-line programs and to run **planman showinfo** and **planman unlock** commands.

For information on how to set the options needed to allow a user to access the command-line interfaces, refer to "Setting up options for using the user interfaces" on page 45.

Starting production

This section provides you with a step-by-step path of basic operations you can perform quickly implement Tivoli Workload Scheduler in your environment using the command-line interface. It is assumed that:

- These steps are performed on the master domain manager immediately after successfully installing the product on the systems where you want to perform your scheduling activities.
- The user ID used to perform the operations is the same as the one used for installing the product.

If you are not familiar with Tivoli Workload Scheduler you can follow the non-optional steps to define a limited number of scheduling objects, and add more as you become familiar with the product. You might start, for example, with two or three of your most frequent applications, defining scheduling objects to meet their requirements only.

Alternatively, you can use the Dynamic Workload Console to perform both the modeling and the operational tasks. Refer to the corresponding product documentation for more information.

The first activity you must perform is to access the Tivoli Workload Scheduler database and to define the environment where you want to perform your scheduling activities using the Tivoli Workload Scheduler scheduling object types. To do this perform the following steps:

1. **Set up the Tivoli Workload Scheduler environment variables**

Run one of the following scripts:

`. ./TWS_home/tws_env.sh` for Bourne and Korn shells in UNIX

`. ./TWS_home/tws_env.csh` for C shells in UNIX

`TWS_home\tws_env.cmd` in Windows

in a system shell to set the *PATH* and *TWS_TISDIR* variables.

2. **Connect to the Tivoli Workload Scheduler database**

   You can use the following syntax to connect to the master domain manager as *TWS_user*:

   ```
   composer -user <TWS_user> -password <TWS_user_password>
   ```

   where *TWS_user* is the user ID you specified at installation time.

   Note: If you want to perform this step and the following ones from a system other than the master domain manager you must specify the connection parameters when starting **composer** as described in "Setting up options for using the user interfaces" on page 45.
3. **Optionally add in the database the definitions to describe the topology of your scheduling environment in terms of:**
   - **Domains**

     Use this step if you want to create a hierarchical tree of the path through the environment. Using multiple domains decreases the network traffic by reducing the communications between the master domain manager and the other workstations. For additional information, refer to "Domain definition" on page 144.
   - **Workstations**

     Define a workstation for each machine belonging to your scheduling environment with the exception of the master domain manager which is automatically defined during the Tivoli Workload Scheduler installation. For additional information, refer to "Workstation definition" on page 127. The master domain manager is automatically defined in the database at installation time.
4. **Optionally define the users allowed to run jobs on Windows workstations**

   Define any user allowed to run jobs using Tivoli Workload Scheduler by specifying user name and password. For additional information, refer to "User definition" on page 170.
5. **Optionally define calendars**

   Calendars allow you to determine if and when a job or a job stream has to run. You can use them to include or exclude days and times for processing. Calendars are not strictly required to define scheduling days for the job streams (`simple` or `rule` run cycles may be used as well); their main goal is to define *global* sets of dates that can be reused in multiple job streams. For additional information refer to "Calendar definition" on page 174.
6. **Optionally define parameters, prompts, and resources**

   For additional information refer to "Variable and parameter definition" on page 175, "Prompt definition" on page 181, and "Resource definition" on page 182.
7. **Define jobs and job streams**

   For additional information refer to "Job" on page 636, and to "Job stream definition" on page 183.
8. **Optionally define restrictions and settings to control when jobs and job streams run.**

You can define dependencies for jobs and job streams. There can be up to 40 dependencies for a job stream. They can be:
- Resource dependencies
- File dependencies
- Job and job stream follow dependencies
- Prompt dependencies

You can define time settings for jobs and job streams to run in terms of:
- Run cycles
- Time constraints

You can tailor the way jobs run concurrently either on a workstation or within a job stream by setting:
- Limit
- Priority

9. **Automate the plan extension at the end of the current production term**

   Add the `final` job stream to the database to perform automatic production plan extension at the end of each current production term by running the following command:

   ```
   add Sfinal
   ```

   For additional information, refer to "Automating production plan processing" on page 88.

10. **Generate the plan**

    Run the **JnextPlan** command to generate the production plan. This command starts the processing of the scheduling information stored in the database and creates the production plan for the time frame specified in the **JnextPlan** command. The default time frame is 24 hours. If you automated the plan generation as described in the previous step, you only need to run the **JnextPlan** command the first time.

When you complete this step-by-step process, your scheduling environment is up and running, with batch processing of an ordered sequence of jobs and job streams being performed against resources defined on a set of workstations, if defined. By default, the first time you run the **JnextPlan** command, the number of jobs that can run simultaneously on a workstation is zero, so make sure that you increase this value by changing the `limit cpu` to allow job execution on that workstation, see the section "limit cpu" on page 334 for more details.

If you want to modify anything while the production plan is already in process, use the **conman** program. While the production plan is processing across the network you can still continue to define or modify jobs and job streams in the database. Consider however that these modifications will only be used if you submit the modified jobs or job streams, using the command **sbj** for jobs or **sbs** for job streams, on a workstation which has already received the plan, or after a new production plan is generated using **JnextPlan**. See Chapter 10, "Managing objects in the plan - conman," on page 291 for more details about the **conman** program and the operations you can perform on the production plan in process.

Chapter 2. Understanding basic workstation processes

In a multi-tier Tivoli Workload Scheduler network, locally on each workstation a group of specialized scheduling processes performs job management and sends back the information about job processing throughout the hierarchical tree until the master domain manager is reached. Using the information received from the workstations, the master domain manager then updates its copy of the `Symphony` file, which contains the records describing the job processing activities to be performed across the Tivoli Workload Scheduler network during the current production plan, and sends the updates on the activities to be performed to the workstations involved.

This chapter describes both the job processing performed at each workstation and the network communication established across the hierarchical tree. It is made up by the following sections:
- "Tivoli Workload Scheduler workstation processes"
- "Starting and stopping processes on a workstation" on page 28
- "Workstation inter-process communication" on page 30
- "Tivoli Workload Scheduler network communication" on page 31

Tivoli Workload Scheduler workstation processes

The management of communication between workstations and local job processing, together with the notification of state updates, are performed on each Tivoli Workload Scheduler workstation by a series of management processes that are active while the engine is running. On fault-tolerant agents and domain managers these processes are based on the WebSphere Application Server infrastructure. This infrastructure is automatically installed with the workstation and allows Tivoli Workload Scheduler to:
- Communicate across the Tivoli Workload Scheduler network.
- Manage authentication mechanisms for remote clients, such as command-line programs, connecting to the master domain manager using the HTTP or HTTPS protocols.

For information on how to start and stop both the WebSphere Application Server infrastructure and the Tivoli Workload Scheduler processes on a workstation refer to "Starting and stopping processes on a workstation" on page 28. Except for manually starting and stopping the WebSphere Application Server and managing connection parameters when communicating across the Tivoli Workload Scheduler network, the WebSphere Application Server infrastructure is transparent when using Tivoli Workload Scheduler.

In this guide *Tivoli Workload Scheduler processes* or *workstation processes* are used to identify the following processes:

netman
monman
writer
mailman
batchman
jobman

With the exception of standard agents, these processes are started in the following order on the Tivoli Workload Scheduler workstations:

netman
: **Netman** is the Network Management process. It is started by the **Startup** command and it behaves like a network listener program which receives start, stop, link, or unlink requests from the network. **Netman** examines each incoming request and spawns a local Tivoli Workload Scheduler process.

monman
: **Monman** is a process started by **netman** and is used in event management. It starts monitoring and **ssmagent** services that have the task of detecting the events defined in the event rules deployed and activated on the specific workstation. When these services catch any such events, after a preliminary filtering action, they send them to the event processing server that runs usually in the master domain manager. If no event rule configurations are downloaded to the workstation, the monitoring services stay idle.

  The communication process between the monitoring agents and the event processing server is independent of the Tivoli Workload Scheduler network topology. It is based directly on the EIF port number of the event processor and the event information flows directly from the monitoring agents without passing through intermediate domain managers. A degree of fault-tolerance is guaranteed by local cache memories that temporarily store the event occurrences on the agents in case communication with the event processor is down.

writer
: **Writer** is a process started by **netman** to pass incoming messages to the local **mailman** process. The **writer** processes (there might be more than one on a domain manager workstation) are started by link requests (see "link" on page 336) and are stopped by unlink requests (see "unlink" on page 412) or when the communicating **mailman** ends.

mailman
: **Mailman** is the Mail Management process. It routes messages to either local or remote workstations. On a domain manager, additional **mailman** processes can be created to divide the load on **mailman** due to the initialization of agents and to improve the timeliness of messages. When the domain manager starts up, it creates a separate **mailman** process instance for each *ServerID* specified in the workstation definitions of the fault-tolerant agents and standard agents it manages. Each workstation is contacted by its own *ServerID* on the domain manager. For additional information, refer to "Workstation definition" on page 127.

batchman
: **Batchman** is the Production Control process. It interacts directly with the copy of the `Symphony` file distributed to the workstations at the beginning of the production period and updates it. **Batchman** performs several functions:

  - Manages locally plan processing and updating.
  - Resolves dependencies of jobs and job streams.
  - Selects jobs to be run.
  - Updates the plan with the results of job processing.

  Batchman is the only process that can update the `Symphony` file.

jobman

Jobman is the Job Management process. It launches jobs under the direction of **batchman** and reports job status back to **mailman**. It is responsible for tracking job states and for setting the environment as defined in the `jobmanrc` and `.jobmanrc` scripts when requesting to launch jobs. For information about these scripts, see Chapter 3, "Configuring the job environment," on page 35. When the **jobman** process receives a launch job message from **batchman**, it spawns a job monitor process. The maximum number of job monitor processes that can be spawned on a workstation is set using the **limit cpu** command from the **conman** command line prompt (see "limit cpu" on page 334).

job monitor (jobman on UNIX, JOBMON.exe and joblnch.exe on Windows operating system)

The job monitor process first performs a set of actions that set the environment before the job is launched, and then launches the job by running the script file or command specified in the job definition. For additional details on how to specify the script file or the command launched with the job, refer to "Job" on page 636.

The setup activities consist of launching the standard configuration file (*TWS_home*/`jobmanrc` in UNIX and *TWS_home*/`jobmanrc.cmd` in Windows) which contains settings that apply to all jobs running on the workstation. In addition, on UNIX workstations a local configuration script *TWS_user*/`.jobmanrc` is launched, if it exists in the home directory of the user launching the job. This local configuration file contains settings that apply only to jobs launched by the specific user. If any of these steps fails, the job ends in the FAIL state.

Attention: If, on Windows systems, a system variable called `TEMP` exists, user *TWS_user* must be authorized to create files in the directory to which the variable is set. If this requirement is not met, the **JOBMON.exe** binary file fails to start successfully.

All processes, except **jobman**, run as the *TWS_user*. **Jobman** runs as root.

On standard agent workstations, the **batchman** process is not launched because this type of workstation does not manage job scheduling. These workstations only launch jobs under the direction of their domain manager. Locally on the workstation the management processes wait for a request to launch a job from the domain manager in LISTEN mode. When the request is received the job is launched locally and the result is sent back to the domain manager. For additional information on standard agent workstations refer to *IBM Tivoli Workload Scheduler: Planning and Installation Guide*.

Figure 3 on page 26 shows the process tree on Tivoli Workload Scheduler workstations, other than standard agents, installed on UNIX:

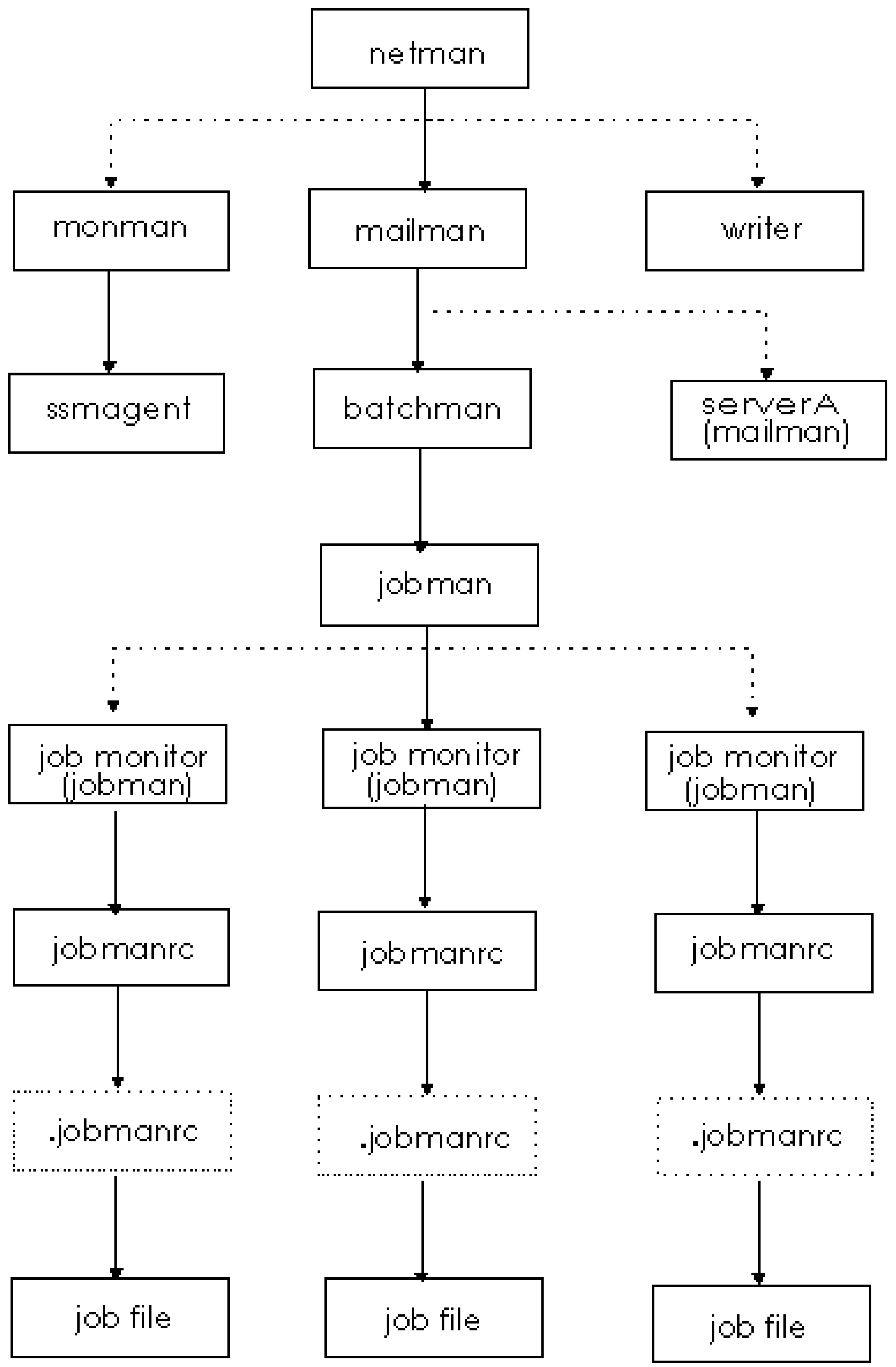


Figure 3. Process tree in UNIX

Figure 4 on page 27 shows the process tree on Tivoli Workload Scheduler workstations, other than standard agents, installed on Windows:

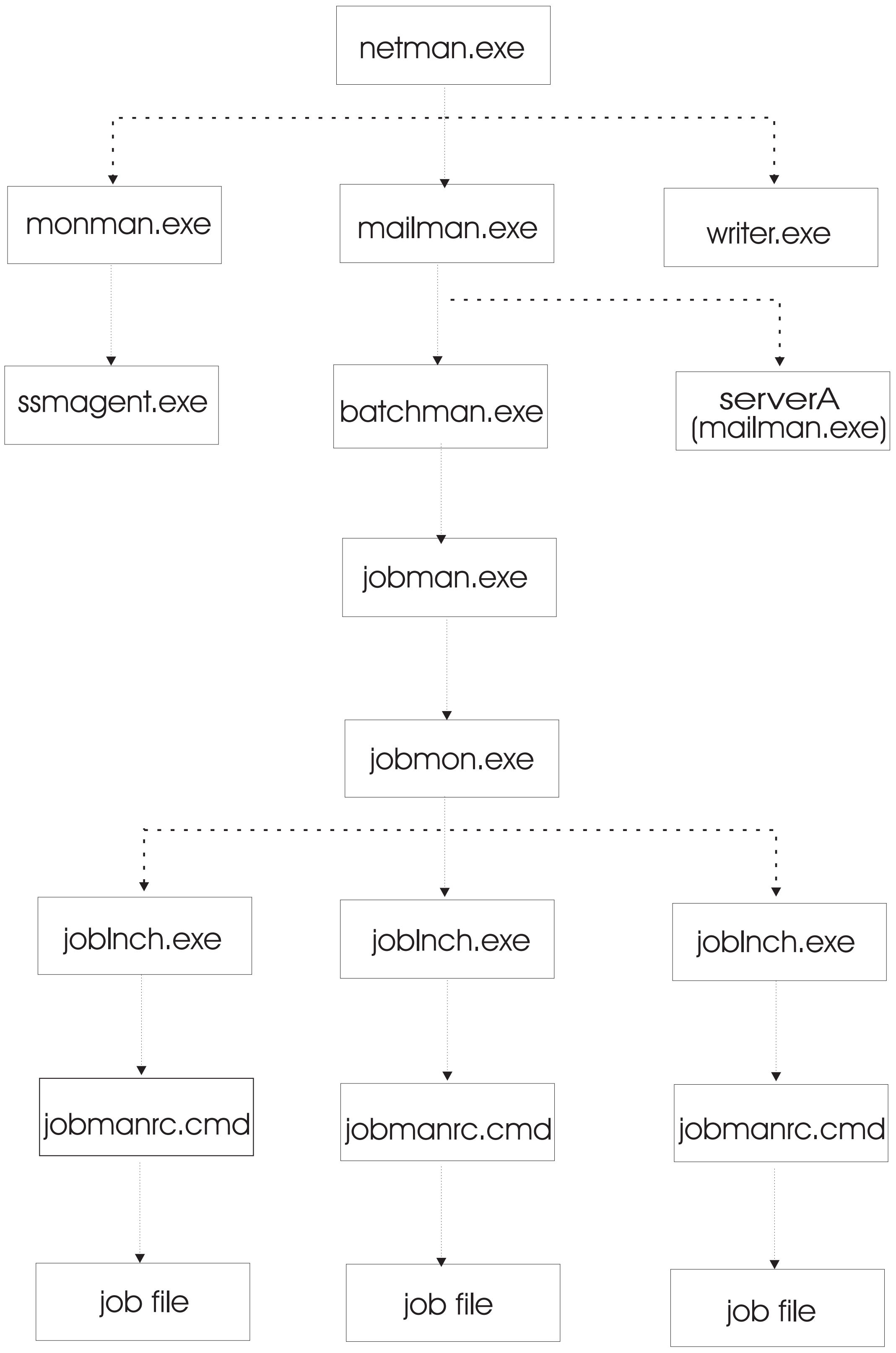


Figure 4. Process tree in Windows

On Windows platforms there is an additional service, the `Tivoli Token Service`, which enables Tivoli Workload Scheduler processes to be launched as if they were issued by the Tivoli Workload Scheduler user.

Starting and stopping processes on a workstation

The type of operating system installed on the workstation determines how Tivoli Workload Scheduler processes can be started from the command line. Table 2 explains how you can start and stop both the WebSphere Application Server infrastructure and Tivoli Workload Scheduler processes on a workstation based on the operating system installed.

Table 2. Starting and stopping Tivoli Workload Scheduler on a workstation

Action	**Commands used on UNIX platform**	**Commands used on Windows platform**
Start all Tivoli Workload Scheduler processes including WebSphere Application Server and the event monitoring engine.	conman start conman startappserver conman startmon	conman start conman startappserver conman startmon
Start netman and WebSphere Application Server. On Windows starts also the `Tivoli Token Service`	./StartUp.sh	StartUp
Stop all Tivoli Workload Scheduler processes and WebSphere Application Server.	conman shutdown ./stopWas.sh	conman shutdown -appsrv shutdown -appsrv
Stop all Tivoli Workload Scheduler processes with the exception of WebSphere Application Server.	conman shutdown	conman shutdown shutdown
Start all Tivoli Workload Scheduler processes with the exception of WebSphere Application Server and the event monitoring engine.	conman start	conman start
Stop all Tivoli Workload Scheduler processes but **netman**, **monman**, **writer**, and **appservman**.	conman stop	conman stop
Stop all Tivoli Workload Scheduler processes (including **netman**).	conman shutdown	conman shutdown shutdown
Start WebSphere Application Server	./startWas.sh or conman startappserver	startWas or conman startappserver
Stop WebSphere Application Server	./stopWas.sh or conman stopappserver	stopWas or conman stopappserver
Start the event monitoring engine	conman startmon	conman startmon
Stop the event monitoring engine	conman stopmon	conman stopmon

Table 2. Starting and stopping Tivoli Workload Scheduler on a workstation (continued)

Action	**Commands used on UNIX platform**	**Commands used on Windows platform**
Start the dynamic agent locally	./StartUpLwa.sh **Note:** can be run by *TWS_user* or root user only.	startuplwa **Note:** On Windows 2008 must be run as Administrator.
Stop the dynamic agent locally	./ShutDownLwa.sh **Note:** can be run by *TWS_user* or root user only.	shutdownlwa **Note:** On Windows 2008 must be run as Administrator.

Note: On Windows systems refrain from using Windows services to stop WebSphere Application Server. Use one of the commands listed in this table instead. If you use Windows services to stop WebSphere Application Server, the *appserverman* process, which continues to run, will start it again.

Refer to "StartUp" on page 464 for more information on the **StartUp** utility command.

Refer to "shutdown" on page 464 for more information on the **shutdown** utility command.

Refer to *IBM Tivoli Workload Manager Administration Guide* for more information on **startWas** and **stopWas** commands.

Refer to "start" on page 385 for more information on the **conman start** command.

Refer to "stop" on page 390 for more information on the **conman stop** command.

Refer to "shutdown" on page 384 for more information on the **conman shutdown** command.

Refer to "startappserver" on page 387 for more information on the **conman startappserver** command.

Refer to "stopappserver" on page 393 for more information on the **conman stopappserver** command.

Refer to "startmon" on page 389 for more information on the **conman startmon** command.

Refer to "stopmon" on page 395 for more information on the **conman stopmon** command.

If the agent is installed on a Windows system, WebSphere Application Server and the **netman** processes are automatically started at start time as services together with the `Tivoli Token Service`. If the agent is installed on a UNIX system, WebSphere Application Server and the **netman** processes can be automatically started at start time by adding a statement invoking **Startup** in the `/etc/inittab` file.

Starting and stopping the dynamic agent

The type of operating system installed on the workstation determines how dynamic agents can be started from the command line.

Table 3. Starting and stopping the dynamic agent

Action	**Commands used on UNIX platform**	**Commands used on Windows platform**
Start the dynamic agent locally	./StartUpLwa.sh **Note:** can be run by *TWS_user* or root user only.	startuplwa **Note:** On Windows 2008 must be run as Administrator.
Stop the dynamic agent locally	./ShutDownLwa.sh **Note:** can be run by *TWS_user* or root user only.	shutdownlwa **Note:** On Windows 2008 must be run as Administrator.

For more infomation about stopping and starting the dynamic agent, see ShutDownLwa and StartUpLwa.

Workstation inter-process communication

Tivoli Workload Scheduler uses message queues for local inter-process communication. There are four message files, which reside in the *TWS_home* directory:

NetReq.msg
This message file is read by the **netman** process for local commands. It receives messages such as START, STOP, LINK, and UNLINK.

Mailbox.msg
This message file is read by the **mailman** process. It receives messages, through the graphical user interface (Dynamic Workload Console) or the console manager (**conman**), incoming from the local **batchman** and **jobman** processes and from other Tivoli Workload Scheduler workstations in the network.

Intercom.msg
This message file is read by the **batchman** process and contains instructions sent by the local **mailman** process.

Courier.msg
This message file is written by the **batchman** process and read by the **jobman** process.

PlanBox.msg
This message file is written by the **batchman** process and read by the engine.

Server.msg
This message file is written by the **batchman** process and read by the engine.

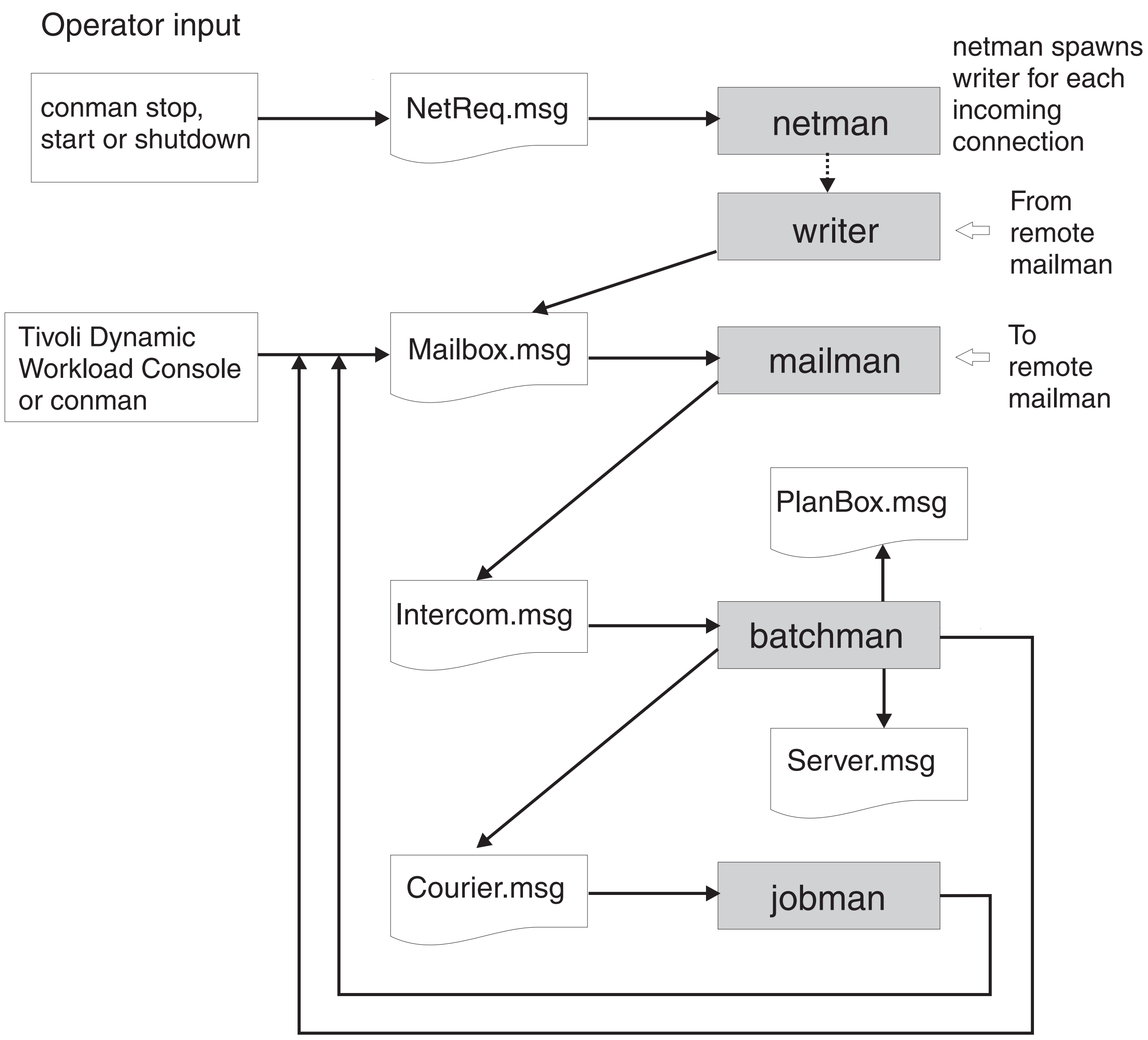


Figure 5. Inter-process communication

These files have a default maximum size of 10MB. The maximum size can be changed using the **evtsize** utility (see "evtsize" on page 447).

Tivoli Workload Scheduler network communication

Tivoli Workload Scheduler uses the TCP/IP protocol for network communication. The node name and the port number used to establish the TCP/IP connection are set for each workstation in its workstation definition. Refer to "Workstation definition" on page 127 for additional details.

A *store-and-forward* technology is used by Tivoli Workload Scheduler to maintain consistency and fault-tolerance at all times across the network by queuing messages in message files while the connection is not active. When TCP/IP communication is established between systems, Tivoli Workload Scheduler provides bi-directional communication between workstations using links. Links are controlled by the **autolink** flag set in the "Workstation definition" on page 127, and by the "link" on page 336 and "unlink" on page 412 commands issued from the **conman** command-line program.

When a link is opened, messages are passed between two workstations. When a link is closed, the sending workstation stores messages in a local message file and sends them to the destination workstation as soon as the link is re-opened.

There are basically two types of communication that take place in the Tivoli Workload Scheduler environment, connection initialization and scheduling event delivery in the form of change of state messages during the processing period. These two types of communication are now explained in detail.

Connection initialization and two-ways communication setup

These are the steps involved in the establishment of a two-ways Tivoli Workload Scheduler link between a domain manager and a remote fault-tolerant agent:

1. On the domain manager, the **mailman** process reads the host name, TCP/IP address, and port number of the fault-tolerant agent from the `Symphony` file.
2. The **mailman** process on the domain manager establishes a TCP/IP connection to the **netman** process on the fault-tolerant agent using the information obtained from the `Symphony` file.
3. The **netman** process on the fault-tolerant agent determines that the request is coming from the **mailman** process on the domain manager, and spawns a new **writer** process to handle the incoming connection.
4. The **mailman** process on the domain manager is now connected to the **writer** process on the fault-tolerant agent. The **writer** process on the fault-tolerant agent communicates the current run number of its copy of the `Symphony` file to the **mailman** process on the domain manager. This run number is the identifier used by Tivoli Workload Scheduler to recognize each `Symphony` file generated by **JnextPlan**. This step is necessary for the domain manager to check if the current plan has already been sent to the fault-tolerant agent.
5. The **mailman** process on the domain manager compares its `Symphony` file run number with the run number of the `Symphony` file on the fault-tolerant agent. If the run numbers are different, the **mailman** process on the domain manager sends to the **writer** process on the fault-tolerant agent the latest copy of the `Symphony` file.
6. When the current `Symphony` file is in place on the fault-tolerant agent, the **mailman** process on the domain manager sends a `start` command to the fault-tolerant agent.
7. The **netman** process on the fault-tolerant agent starts the local **mailman** process. At this point a one-way communication link is established from the domain manager to the fault-tolerant agent.
8. The **mailman** process on the fault-tolerant agent reads the host name, TCP/IP address, and port number of the domain manager from the `Symphony` file and uses them to establish the uplink back to the **netman** process on the domain manager.
9. The **netman** process on the domain manager determines that the request is coming from the **mailman** process on the fault-tolerant agent, and spawns a new **writer** process to handle the incoming connection. The **mailman** process on the fault-tolerant agent is now connected to the **writer** on the domain manager and a full two-way communication

link is established. As a result of this, the **writer** process on the domain manager writes messages received from the fault-tolerant agent into the `Mailbox.msg` file on the domain manager, and the **writer** process on the fault-tolerant agent writes messages from the domain manager into the `Mailbox.msg` file on the fault-tolerant agent.

Job processing and scheduling event delivery in the form of change of state messages during the processing day performed locally by the fault-tolerant agent
During the production period, the `Symphony` file present on the fault-tolerant agent is read and updated with the state change information about jobs that are run locally by the Tivoli Workload Scheduler workstation processes. These are the steps that are performed locally on the fault-tolerant agent to read and update the `Symphony` file and to process jobs:

1. The **batchman** process reads a record in the `Symphony` file that states that `job1` is to be launched on the workstation.
2. The **batchman** process writes in the `Courier.msg` file that `job1` has to start.
3. The **jobman** process reads this information in the `Courier.msg` file, starts `job1`, and writes in the `Mailbox.msg` file that `job1` started with its *process_id* and *timestamp*.
4. The **mailman** process reads this information in its `Mailbox.msg` file, and sends a message that `job1` started with its *process_id* and *timestamp*, to both the `Mailbox.msg` file on the domain manager and the local `Intercom.msg` file on the fault-tolerant agent.
5. The **batchman** process on the fault-tolerant agent reads the message in the `Intercom.msg` file and updates the local copy of the Symphony file.
6. When job `job1` completes processing, the **jobman** process updates the `Mailbox.msg` file with the information that says that `job1` completed.
7. The **mailman** process reads the information in the `Mailbox.msg` file, and sends a message that `job1` completed to both the `Mailbox.msg` file on the domain manager and the local `Intercom.msg` file on the fault-tolerant agent.
8. The **batchman** process on the fault-tolerant agent reads the message in the `Intercom.msg` file, updates the local copy of the `Symphony` file, and determines the next job that has to be run.

For information on how to tune job processing on a workstation, refer to the *IBM Tivoli Workload Scheduler: Administration Guide*.

Support for Internet Protocol version 6

Tivoli Workload Scheduler supports Internet Protocol version 6 (IPv6) in addition to the legacy IPv4. To assist customers in staging the transition from an IPv4 environment to a complete IPv6 environment, Tivoli Workload Scheduler provides IP dual-stack support. In other terms, the product is designed to communicate using both IPv4 and IPv6 addresses simultaneously with other applications using IPv4 or IPv6.

To this end, the `gethostbyname` and the `gethostbyaddr` functions were dropped from Tivoli Workload Scheduler as they exclusively support IPv4. They are replaced by the new `getaddrinfo` API that makes the client-server mechanism entirely protocol independent.

The `getaddrinfo` function handles both name-to-address and service-to-port translation, and returns `sockaddr` structures instead of a list of addresses These `sockaddr` structures can then be used by the socket functions directly. In this way, `getaddrinfo` hides all the protocol dependencies in the library function, which is where they belong. The application deals only with the socket address structures that are filled in by `getaddrinfo`.

Chapter 3. Configuring the job environment

This chapter describes how to customize the way job management is performed on a workstation. This customization is made by assigning locally on each workstation values to variables that have an impact on the processing of **jobman**. This chapter includes the following sections:

Job environment overview

On each workstation, jobs are launched by the **batchman** production control process. The **batchman** process resolves all job dependencies to ensure the correct order of job processing, and then queues a job launch message to the **jobman** process.

Each of the processes launched by **jobman**, including the configuration scripts and the jobs, retains the user name recorded with the logon of the job. Submitted jobs (jobs, files, or commands submitted not through a scheduled plan, but *manually* by a user) retain the submitting user's name.

The **jobman** process starts a job monitor process that begins by setting a group of environment variables, and then runs a standard configuration script named *TWS_home*/**jobmanrc** which can be customized. The **jobmanrc** script sets variables that are used to configure locally on the workstation the way jobs are launched, regardless of the user.

On UNIX workstations, if the user is allowed to use a local configuration script, and the script *USER_HOME*/**.jobmanrc** exists, the local configuration script **.jobmanrc** is *also* run. The job is then run either by the standard configuration script, or by the local one. The results of job processing are reported to **jobman** which, in turn, updates the Mailbox.msg file with the information on job completion status. To have jobs run with the user's environment, add the following instruction in the local configuration script:

```
. $USER_home/.profile
```

Note: Before adding the .profile in the .jobmanrc file, make sure that it does not contain any *stty* setting or any step that requires user manual intervention. In case it does, add in the .jobmanrc file only the necessary steps contained in the .profile.

On Windows workstations the local configuration script djobmanrc.cmd is run if it exists in the user's Documents and Settings directory which is represented by the environment variable %USERPROFILE% and depends on the Windows language installation. The djobmanrc.cmd script will be ran by jobmanrc.cmd script.

Environment variables exported by jobman

The variables listed in Table 4 are set locally on the workstation and exported by **jobman** on Windows operating systems:

Table 4. Job environment variables for Windows

Variable Name	**Value**
COMPUTERNAME	The value of the *COMPUTERNAME* set in the user environment.
HOME	The path where the Tivoli Workload Scheduler instance was installed.
HOMEDRIVE	The value of the *HOMEDRIVE* set in the user environment.
HOMEPATH	The value of the *HOMEPATH* set in the user environment.
LANG	The value of the *LANG* set in the user environment. If not set, its value is set to C.
LOGNAME	The login user's name.
MAESTRO_OUTPUT_STYLE	The setting for output style for long object names. The default value is LONG.
SystemDrive	The value of the *SYSTEMDRIVE* set in the user environment.
SystemRoot	The value of the *SYSTEMROOT* set in the user environment.
TEMP	The value of the *TEMP* set in the user environment. If not specified, its value is set to `c:\temp`.
TIVOLI_JOB_DATE	The scheduled date for a job.
TMPTEMP	The value of the *TMP* set in the user environment. If not specified, its value is set to `c:\temp`.
TMPDIR	The value of the *TMPDIR* set in the user environment. If not specified, its value is set to `c:\temp`.
TWS_PROMOTED_JOB	Applies to the Workload Service Assurance functions. Can be `YES` or `No`. When the value is `YES`, it means that the job (a critical job or one of its predecessors) was promoted.
TZ	The time zone, if it was set in the operating system environment.
UNISON_CPU	The name of this workstation.
UNISON_DIR	The value of the *UNISON_DIR* set in the user environment.
UNISON_EXEC_PATH	The `jobmanrc` fully qualified path.
UNISONHOME	The path where the Tivoli Workload Scheduler instance was installed.
UNISON_HOST	The name of the host CPU.
UNISON_JOB	The absolute job identifier: `worktation#sched_id.job`.

Table 4. Job environment variables for Windows (continued)

Variable Name	Value
UNISON_JOBNUM	The job number.
UNISON_MASTER	The name of the master workstation.
UNISON_RUN	The Tivoli Workload Scheduler current production run number.
UNISON_SCHED	The job stream name.
UNISON_SCHED_DATE	The Tivoli Workload Scheduler production date (yymmdd) reported in the header of the `Symphony` file.
UNISON_SCHED_ID	The *jobstreamID* of the job stream containing the job in process.
UNISON_SCHED_IA	The *StartTime* of the job stream containing the job in process.
UNISON_SCHED_EPOCH	The Tivoli Workload Scheduler production date expressed in epoch form.
UNISON_SHELL	The login shell of the user running the job.
UNISON_STDLIST	The path name of the standard list file of the job.
UNISON_SYM	The `Symphony` record number of the launched job.
USERDOMAIN	The value of the *USERDOMAIN* set in the user environment.
USERNAME	The value of the *USERNAME* set in the user environment.
USERPROFILE	The value of the *USERPROFILE* set in the user environment.

The variables listed in Table 5 are set locally on the workstation and exported by **jobman** on UNIX operating systems:

Table 5. Job environment variables for UNIX

Variable Name	Value
HOME	The home directory of the user.
LANG	The value of the *LANG* set in the user environment.
LD_LIBRARY_PATH	The value of the *LD_LIBRARY_PATH* set in the user environment.
LD_RUN_PATH	The value of the *LD_RUN_PATH* set in the user environment.
LOGNAME	The login user name.
MAESTRO_OUTPUT_STYLE	The setting for output style for long object names. The default value is LONG.
PATH	`/bin:/usr/bin`
TIVOLI_JOB_DATE	The scheduled date for a job.
TWS_PROMOTED_JOB	Applies to the Workload Service Assurance functions. Can be `YES` or `No`. When the value is `YES`, it means that the job (a critical job or one of its predecessors) was promoted.

Table 5. Job environment variables for UNIX (continued)

Variable Name	Value
TWS_TISDIR	The value of the *TWS_TISDIR* set in the user environment.
TZ	The time zone set.
UNISON_CPU	The name of this workstation.
UNISON_DIR	The value of the *UNISON_DIR* set in the user environment.
UNISON_EXEC_PATH	The `.jobmanrc` fully qualified path.
UNISONHOME	The path where the Tivoli Workload Scheduler instance was installed.
UNISON_HOST	The name of the host CPU.
UNISON_JOB	The absolute job identifier: `worktation#sched_id.job`.
UNISON_JOBNUM	The job number.
UNISON_MASTER	The name of the master workstation.
UNISON_RUN	The Tivoli Workload Scheduler current production run number.
UNISON_SCHED	The job stream name.
UNISON_SCHED_DATE	The Tivoli Workload Scheduler production date (yymmdd) reported in the header of the `Symphony` file.
UNISON_SCHED_ID	The *jobstreamID* of the job stream containing the job in process.
UNISON_SCHED_IA	The *StartTime* of the job stream containing the job in process.
UNISON_SCHED_EPOCH	The Tivoli Workload Scheduler production date, expressed in epoch form.
UNISON_SHELL	The login shell of the user running the job.
UNISON_STDLIST	The path name of the standard list file of the job.
UNISON_SYM	The `Symphony` record number of the launched job.

Customizing date formatting in the stdlist

You can use an environment variable named ***UNISON_DATE_FORMAT*** to specify the date format that is used for the date in the header and in the footer of the `stdlist` file. This variable can be set on both UNIX and Windows workstations and must be set before starting Tivoli Workload Scheduler processes on that workstation to become effective. To set this variable, follow these steps:

On UNIX workstations

1. Add the statement to export the *UNISON_DATE_FORMAT* variable in the root `.profile` file.
2. Run the `.profile` file.
3. Run **conman shutdown** and then **./StartUp.sh**.

On Windows workstations

1. From the System Properties set the *UNISON_DATE_FORMAT* in the System Variable.
2. Run **conman shutdown** and then **StartUp**.

These are some examples of the settings used to display the year format in the date field in the header and footer of the `stdlist` file. The setting:

```
UNISON_DATE_FORMAT = "%a %x %X %Z %Y"
```

produces an output with the following format:

```
Fri 15/10/04 11:05:24 AM GMT 2004
```

The setting:

```
UNISON_DATE_FORMAT = "%a %x %X %Z"
```

produces an output with the following format:

```
Fri 15/10/04 11:05:24 AM GMT
```

Set this variable locally on every workstation for which you want to display the 4-digit year format. If omitted, the standard 2-digit format is used.

Customizing job processing on a UNIX workstation - jobmanrc

A standard configuration script template named *TWS_home*/config/**jobmanrc** is supplied with Tivoli Workload Scheduler. It is installed automatically as *TWS_home*/jobmanrc. This script can be used by the system administrator to set the desired environment before each job is run. To alter the script, make your modifications in the working copy (*TWS_home*/jobmanrc), leaving the template file unchanged. The file contains variables which can be configured, and comments to help you understand the methodology. Table 6 describes the `jobmanrc` variables.

Table 6. Variables defined by default in the jobmanrc file

Variable Name	Value
UNISON_JCL	The path name of the job's script file.
UNISON_STDLIST	The path name of the job's standard list file.
UNISON_EXIT	**yes** | **no** If set to **yes**, the job ends immediately if any command returns a nonzero exit code. If set to **no**, the job continues to run if a command returns a nonzero exit code. Any other setting is interpreted as **no**.
LOCAL_RC_OK	**yes** | **no** If set to **yes**, the user's local configuration script is run (if it exists), passing *$UNISON_JCL* as the first argument. The user might be allowed or denied this option. See "Customizing job processing for a user on UNIX workstations - .jobmanrc" on page 41 for more information. If set to **no**, the presence of a local configuration script is ignored, and *$UNISON_JCL* is run. Any other setting is interpreted as **no**.

Table 6. Variables defined by default in the jobmanrc file (continued)

Variable Name	Value

MAIL_ON_ABEND	**yes** | **no**
	
	For UNIX operating systems: If set to **yes**, a message is mailed to the login user's mailbox if the job ends with a non zero exit code. This can also be set to one or more user names, separated by spaces so that a message is mailed to each user. For example, "root mis sam mary". If set to **no**, no messages are mailed if the job abends. Abend messages have the following format:
	`cpu#sched.job`
	`jcl-file failed with exit-code`
	`Please review standard-list-filename`
	
	You can change the wording of the message or translate the message into another language. For an explanation of how to do this, see "Customizing the MAIL_ON_ABEND section of jobmanrc."

SHELL_TYPE	**standard** | **user** | **script** If set to **standard**, the first line of the JCL file is read to determine which shell to use to run the job. If the first line does not start with **#!**, then `/bin/sh` is used to run the local configuration script or *$UNISON_JCL*. Commands are echoed to the job's standard list file. If set to user, the local configuration script or *$UNISON_JCL* is run by the user's login shell (*$UNISON_SHELL*). Commands are echoed to the job's standard list file. If set to **script**, the local configuration script or *$UNISON_JCL* is run directly, and commands are not echoed unless the local configuration script or *$UNISON_JCL* contains a **set -x** command. Any other setting is interpreted as **standard**.
USE_EXEC	**yes** | **no** If set to **yes**, the job, or the user's local configuration script is run using the **exec** command, thus eliminating an extra process. This option is overridden if *MAIL_ON_ABEND* is also set to **yes**. Any other setting is interpreted as **no**, in which case the job or local configuration script is run by another shell process.

Customizing the MAIL_ON_ABEND section of jobmanrc

You can modify the wording used in the message sent to the users specified in the ***MAIL_ON_ABEND*** field of the `TWS_home/jobmanrc` configuration file by accessing that file and changing the wording in the parts highlighted in bold:

```
# Mail a message to user or to root if the job fails.

if [ "$MAIL_ON_ABEND" = "YES" ]
then
  if [ $UNISON_RETURN -ne 0 ]
  then
    mail $LOGNAME <<-!
        $UNISON_JOB
        \'$UNISON_JCL\' failed with $UNISON_RETURN
        Please review $UNISON_STDLIST
!
  fi
elif [ "$MAIL_ON_ABEND" = "ROOT" ]
then
  if [ $UNISON_RETURN -ne 0 ]
  then
    mail root <<-!
        $UNISON_JOB
        \'$UNISON_JCL\' failed with $UNISON_RETURN
        Please review $UNISON_STDLIST
!
  fi
elif [ "$MAIL_ON_ABEND" != "NO" ]
then
  if [ $UNISON_RETURN -ne 0 ]
  then
    mail $MAIL_ON_ABEND <<-!
        $UNISON_JOB
        \'$UNISON_JCL\' failed with $UNISON_RETURN
        Please review $UNISON_STDLIST
!
  fi
fi
```

Customizing job processing for a user on UNIX workstations - .jobmanrc

On UNIX workstations, the local configuration script **.jobmanrc** permits users to establish a desired environment when processing their own jobs. Unlike the **jobmanrc** script, the **.jobmanrc** script can be customized to perform different actions for different users. Each user defined as *tws_user* can customize in the home directory the **.jobmanrc** script to perform pre- and post-processing actions. The **.jobmanrc** script is an extra step that occurs before the job is actually launched.

The **.jobmanrc** script runs only under the following conditions:

- The standard configuration script, `jobmanrc`, is installed, and the environment variable ***LOCAL_RC_OK*** is set to **yes** (see Table 6 on page 39).
- If the file *TWS_home*/`localrc.allow` exists, the user's name must appear in the file. If the *TWS_home*/`localrc.allow` file does not exist, the user's name must not appear in the file, *TWS_home*/`localrc.deny`. If neither of these files exists, the user is permitted to use a local configuration script.
- The local configuration script is installed in the user's home directory (*USER_home*/`.jobmanrc`), and it has execute permission.

Jobs are not automatically run, the command or script must be launched from inside the **.jobmanrc**. Depending on the type of process activity you want to perform, the command or script is launched differently. Follow these general rules when launching scripts from inside **.jobmanrc**:

- Use **eval** if you want to launch a command.

- Use either **eval** or **exec** if you want to launch a script that does not need post processing activities.
- Use **eval** if you want to launch a script that requires post processing activities.

If you intend to use a local configuration script, it must, at a minimum, run the job's script file (*$UNISON_JCL*). Tivoli Workload Scheduler provides you with a standard configuration script, `jobmanrc`, which runs your local configuration script as follows:

```
$EXECIT $USE_SHELL $USER_home/.jobmanrc "$UNISON_JCL" $IS_COMMAND
```

where:

- The value of *USE_SHELL* is set to the value of the `jobmanrc` ***SHELL_TYPE*** variable (see Table 6 on page 39).
- *IS_COMMAND* is set to **yes** if the job was scheduled or submitted in production using **submit docommand**.
- *EXECIT* is set to **exec** if the variable ***USE_EXEC*** is set to **yes** (see Table 6 on page 39), otherwise it is null.

All the variables exported into **`jobmanrc`** are available in the **`.jobmanrc`** shell, however, variables that are defined, but not exported, are not available.

The following example shows how to run a job's script file, or command, in your local configuration script:

```
#!/bin/ksh
PATH=TWS_home:TWS_home/bin:$PATH
export PATH
/bin/sh -c "$UNISON_JCL"
```

The following is an example of a `.jobmanrc` that does processing based on the exit code of the user's job:

```
#!/bin/sh
#
PATH=TWS_home:TWS_home/bin:$PATH
export PATH
/bin/sh -c "$UNISON_JCL"
#or use eval "$UNISON_JCL" and the quotes are required
RETVAL=$?
if [ $RETVAL -eq 1 ]
then
 echo "Exit code 1 - Non Fatal Error"
 exit 0
elif [ $RETVAL -gt 1 -a $RETVAL -lt 100 ]
then
 conman "tellop This is a database error - page the dba"
elif [ $RETVAL -ge 100 ]
then
 conman "tellop Job aborted. Please page the admin"
fi
```

Customizing job processing on a Windows workstation - jobmanrc.cmd

A standard configuration script template named *TWS_home*\config\jobmanrc.cmd is supplied with Tivoli Workload Scheduler. It is installed automatically as *TWS_home*\jobmanrc.cmd. You can use this command file to set the desired environment before each job is run. To alter the file, make your modifications in the working copy (*TWS_home*\jobmanrc.cmd), leaving the template file unchanged. The file contains variables which can be configured, and comments to help you understand the methodology. Table 7 describes the jobmanrc.cmd variables.

Table 7. Variables defined by default in the jobmanrc.cmd file

Variable Name	**Value**
HOME	The path to the *TWS_home* directory
POSIXHOME	The path to the *TWS_home* directory in a POSIX complaint format
LOCAL_RC_OK	• If set to **yes**, the user's local configuration script is run, if existing. • If set to **no**, the presence of a local configuration script is ignored. Any other setting is interpreted as **no**.
MAIL_ON_ABEND	• If set to **YES**, an email is sent to the email ID defined in the *email_ID* variable, if the job ends in error. • If set to any value other than **YES** or **NO**, an email is sent to the email ID specified in this variable if the job ends in error. • If set to **NO**, no messages are sent if the job ends in error. For more details, see "Customizing the MAIL_ON_ABEND section of jobmanrc.cmd."

Customizing the MAIL_ON_ABEND section of jobmanrc.cmd

You can modify the wording used in the message sent to the users specified in the ***MAIL_ON_ABEND*** field of the *TWS_home*/jobmanrc.cmd configuration file by accessing that file and changing the wording in the parts highlighted in bold. To clarify how to generate the email message, a sample mail program with name bmail.exe is used.

```
if /I "%MAIL_ON_ABEND%"=="NO" (goto :out) else (goto :mail_on_abend)

:mail_on_abend
REM ******email, task or other action inserted here *******************
if /I "%MAIL_ON_ABEND%"=="YES" (goto :email) else (goto :email_spec)

:email
c:\"Program Files"\utils\bmail.exe -s smtp.yourcompany.com -t %EMAIL_ID%
-f %COMPUTERNAME%@yourcompany.com -h -a "Subject: Job %UNISON_JOB% abended"
-b "Job %UNISON_JOB% Job Number %UNISON_JOBNUM% abended"
goto :out

:email_spec
REM set > c:\tmp\abended_jobs\%UNISON_JOB%.j%UNISON_JOBNUM%
```

```
c:\"Program Files"\utils\bmail.exe -s smtp.yourcompany.com -t %MAIL_ON_ABEND%
-f %COMPUTERNAME%@yourcompany.com -h -a "Subject: Job %UNISON_JOB% abended"
-b "Job %UNISON_JOB% Job Number %UNISON_JOBNUM% abended"
```

Customizing job processing on a Windows workstation - djobmanrc.cmd

On Windows workstations, you can use the local configuration script djobmanrc.cmd to establish a specific environment when processing your custom jobs. Unlike the jobmanrc.cmd script, you can customize the djobmanrc.cmd script to perform different actions for different users.

The following conditions apply:

- The script must contain all environment application variables or paths necessary for Tivoli Workload Scheduler to launch correctly.
- The script must exist if a user-specific environment for running job is required or if an email must be sent to the job logon user when the Tivoli Workload Scheduler job ends in error.

To create a custom djobmanrc.cmd script, perform the following steps:

1. Logon as the user who defines environment variables for launching Tivoli Workload Scheduler jobs.
2. Open a DOS command prompt.
3. Type the **set** command redirecting standard output to a flat file named `user_env`.
4. Create a file named djobmanrc.cmd in the user's Documents and Settings directory with the following default text at the beginning:
   ```
   @ECHO OFF
       echo Invoking %USERNAME% DJOBMANRC.CMD V.1
       set USERPROFILE=%USERPROFILE%
       ::Setup User Environment Phase
   ```
5. Edit the `user_env` file created in step 3.
6. Insert the **set** command on each line before each environment variable.
7. Add the changes to the PATH variable at the end of the djobmanrc.cmd in a string similar to the following:
   ```
   set PATH=<TWSHOME>;<TWSHOME>\bin;%PATH%
   ```
8. Add the following text at the end of the `user_env` file and replace the string *user email id* with the email ID of the user that receives the email notification if the job ends in error.
   ```
   set EMAIL_ID=<user email id>
       ::Launch Operation Phase
       %ARGS%
       ::Post Operations Phase
       :out
   ```
9. Add the updated `user_env` file to the end of the djobmanrc.cmd file. The edited djobmanrc.cmd file should look like the following example:
   ```
   @ECHO OFF
       echo Invoking %USERNAME% DJOBMANRC.CMD V.1
       set USERPROFILE=%USERPROFILE%
       ::Setup User Environment Phase
       set ALLUSERSPROFILE=C:\Documents and Settings\All Users
       set APPDATA=C:\Documents and Settings\petes\Application Data
       set CommonProgramFiles=C:\Program Files\Common Files
       set COMPUTERNAME=PSOTOJ
       set ComSpec=C:\WINDOWS\system32\cmd.exe
       set CURDRIVE=C
       set FP_NO_HOST_CHECK=NOset
       set HOMEDRIVE=c:
       set HOMEPATH=\docs
   ```

```
set LOGONSERVER=\\PSOTOJ
set NEWVAR=c:\tmp\tmp\mlist1
set NUMBER_OF_PROCESSORS=1
set OPC_CLIENT_ROOT=C:\opc\Client
set OS=Windows_NT
set Path=C:\Program Files\utils;C:\PROGRAM
FILES\THINKPAD\UTILITIES;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program
Files\IBM\Infoprint Select;C:\Utilities;C:\Notes;C:\Program Files\IBM\Trace Facility\;C:\Program
Files\IBM\Personal Communications\;C:\Program Files\XLView\;C:\lotus\compnent\;C:\WINDOWS\Downloaded
Program Files;C:\Program Files\Symantec\pcAnywhere\;"C:\Program Files\Symantec\Norton Ghost
2003\";C:\Infoprint;
set PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
set PCOMM_Root=C:\Program Files\IBM\Personal Communications\
set PDBASE=C:\Program Files\IBM\Infoprint Select
set PDHOST=
set PD_SOCKET=6874
set PROCESSOR_ARCHITECTURE=x86
set PROCESSOR_IDENTIFIER=x86 Family 6 Model 9 Stepping 5, GenuineIntel
set PROCESSOR_LEVEL=6
set PROCESSOR_REVISION=0905
set ProgramFiles=C:\Program Files
set PROMPT=$P$G
set SESSIONNAME=Console
set SystemDrive=C:
set SystemRoot=C:\WINDOWS
set TEMP=C:\DOCUME⌂1\petes\LOCALS⌂1\Temp
set TMP=C:\DOCUME⌂1\petes\LOCALS⌂1\Temp
set tvdebugflags=0x260
set tvlogsessioncount=5000
set TWS4APPS_JDKHOME=c:\win32app\TWS\pete\methods\_tools
set USERDOMAIN=PSOTOJ
set USERNAME=petes
set USERPROFILE=C:\Documents and Settings\petes
set windir=C:\WINDOWSPATH=c:\win32app\tws\twsuser:c:\win32app\tws\twsuser\bin:%PATH%
set PATH=c:\win32app\TWS\twsuser;c:\win32app\TWS\twsuser\bin;%PATH%
set EMAIL_ID=johndoe@yourcompany.com
::Launch Operation Phase
%ARGS%
::Post Operations Phase
:out
```

The Launch Operations Phase in the script is where script, binary or command defined for job is completed. The “%ARGS%” text is required.

The Post Operations Phase in the script is where a job exit code might be re-adjusted from ABEND to SUCC state, changing a non-zero exit code to a zero exit code. Some applications might have exit codes that might be warnings. Tivoli Workload Scheduler evaluates exit codes as either zero or non-zero. Zero exit codes indicate a job in “SUCC” state. All other codes indicate a job in ABEND state. Specific non-zero exit codes can be adjusted if necessary. The following example shows what might be included in the Post Operations Phase. The example retrieves the exit code of the defined job to determine how to handle itbased on the `If` statements:

```
set EMAIL_ID=johndoe@yourcompany.com
    ::Launch Operation Phase
    %ARGS%
    ::Post Operations Phase
    set RETVAL=%ERRORLEVEL%
    if "%RETVAL%"=="0" goto out
    if "%RETVAL%"=="1" set RETVAL=0
    if "%RETVAL%"=="6" set RETVAL=0
    :out
    exit %RETVAL%
```

Setting up options for using the user interfaces

To use the Dynamic Workload Console, the connection parameters are supplied within the console and saved as part of its configuration.

To use the Tivoli Workload Scheduler command line client, you need to provide the following setup information (called the ***connection_parameters***) to connect to the master domain manager via HTTP/HTTPS using the WebSphere Application Server infrastructure:

hostname
The hostname of the master domain manager.

port number
The port number used when establishing the connection with the master domain manager.

username, password
The credentials, username and password, of the *TWS_user*.

proxy hostname
The proxy hostname used in the connection with the HTTP protocol.

proxy port number
The proxy port number used in the connection with the HTTP protocol.

protocol
The protocol used during the communication. This can be HTTP with basic authentication, or HTTPS with certificate authentication.

timeout
The timeout indicating the maximum time the connecting user interface program can wait for the master domain manager response before considering the communication request as failed.

default workstation
The workstation name of the master domain manager you want to connect to.

SSL parameters
If you have configured your network to use SSL to communicate between the interfaces and the master domain manager, you need also to supply the appropriate set of SSL parameters (which depends on how your SSL is configured.

In the case of the command line client installed on the master domain manager, this configuration is performed automatically at installation.

For the command line client installed on other workstations, this information can be supplied either by storing it in properties files on those workstations, or by supplying the information as part of the command string of the commands you use.

The properties files referred to are the **`localopts`** and **`useropts`** files:

localopts
This contains a set of parameters applicable to the local workstation for a specific instance of the installed product.

useropts
This contains a subset of those localopts parameters that have custom values for a specific user. The path of this file is within the user's home directory, which maintains the privacy of this information.

Because Tivoli Workload Scheduler supports multiple product instances installed on the same machine, there can be more than one `useropts` file instance of each user. The possibility to have more `useropts` files, having a different name each, provides the ability to specify different sets of connection settings for users defined on more than one instance of the product installed on the same machine.

In the `localopts` file of each instance of the installed product the option named ***useropts*** identifies the file name of the `useropts` file that has to be accessed to connect to that installation instance.

This means that, if two Tivoli Workload Scheduler instances are installed on a machine and a system user named `operator` is defined as user in both instances, then in the `localopts` file of the first instance the local option *`useropts`* `= useropts1` identifies the `useropts1` file containing the connection parameters settings that user `operator` needs to use to connect to that Tivoli Workload Scheduler instance. On the other hand, in the `localopts` file of the second Tivoli Workload Scheduler instance the local option *`useropts`* `= useropts2` identifies the `useropts2` file containing the connection parameters settings that user `operator` needs to use to connect to that Tivoli Workload Scheduler instance.

Full details of how to configure this access are given in the *Tivoli Workload Scheduler: Administration Guide*, in the topic entitled "Configuring command-line client access authentication"

Chapter 4. Managing the production cycle

The core part of a job management and scheduling solution is the creation and management of the *production plan*. The production plan is the to-do list that contains the actions to be performed in a stated interval of time on the workstations of the scheduling network using the available resources and preserving the defined relationships and restrictions.

This chapter describes how Tivoli Workload Scheduler manages plans.

The chapter is divided into the following sections:
- "Plan management basic concepts"
- "Customizing plan management using global options" on page 66
- "Creating and extending the production plan" on page 70
- "Planman command line" on page 73
- "Starting production plan processing" on page 88
- "Automating production plan processing" on page 88

Plan management basic concepts

The *production plan* contains information about the jobs to run, on which fault-tolerant agent, and what dependencies must be satisfied before each job is launched. Tivoli Workload Scheduler creates the production plan starting from the modeling data stored in the database and from an intermediate plan called *preproduction plan*. The preproduction plan is automatically created and managed by the product. To avoid problems the database is locked during the generation of the plan, and is unlocked when the generation completes or if an error condition occurs. The preproduction plan is used to identify in advance the job stream instances and the external follows job stream dependencies involved in a specified time-window.

You use the **JnextPlan** script on the master domain manager to generate the production plan and distribute it across the Tivoli Workload Scheduler network. For additional information on the **JnextPlan** script, refer to "Creating and extending the production plan" on page 70.

To generate and start a new production plan Tivoli Workload Scheduler performs the following steps:
1. Updates the preproduction plan with the objects defined in the database that were added or updated since the last time the plan was created or extended.
2. Retrieves from the preproduction plan the information about the job streams to run in the specified time period and saves it in an intermediate production plan.
3. Includes in the new production plan the uncompleted job streams from the previous production plan.
4. Creates the new production plan and stores it in a file named `Symphony`.
5. Distributes a copy of the `Symphony` file to the workstations involved in the new product plan processing.
6. Logs all the statistics of the previous production plan into an archive

7. Updates the job stream states in the preproduction plan.

The copy of the newly generated Symphony file is deployed starting from the top domain's fault-tolerant agents and domain managers of the child domains and down the tree to all subordinate domains.

Each fault-tolerant agent that receives the production plan can continue processing even if the network connection to its domain manager goes down.

At each destination fault-tolerant agent the Tivoli Workload Scheduler processes perform the following actions to manage job processing:

1. Access the copy of the Symphony file and read the instructions about which jobs to run.
2. Make calls to the operating system to launch jobs as required.
3. Update its copy of the Symphony file with the job processing results and send notification back to the master domain manager and to all full status fault-tolerant agents. The original copy of the Symphony file stored on the master domain manager and the copies stored on the backup master domain managers, if defined, are updated accordingly.

This means that during job processing, each fault-tolerant agent has its own copy of the Symphony file updated with the information about the jobs it is running (or that are running in its domain and child domains if the fault-tolerant agent is full-status or a domain manager). Also the master domain manager (and backup master domain manager if defined) has the copy of the Symphony file that contains all updates coming from all fault-tolerant agents. In this way the Symphony file on the master domain manager is kept up to date with the jobs that need to be run, the ones that are running, and the ones that have completed.

The processing that occurs on each workstation involved in the current production plan activities is described in more detail in “Tivoli Workload Scheduler workstation processes” on page 23.

Note: While the current production plan is in process, any changes you make to the plan using **conman** do not affect the definitions in the database. Changes to the objects in the database do not affect the plan until the production plan is extended or created again using the **JnextPlan** script or **planman** command-line interface. Updates to objects in the database do not affect instances of those objects already in the production plan.

Preproduction plan

The preproduction plan is used to identify in advance the job stream instances and the job stream dependencies involved in a specified time period.

This improves performance when generating the production plan by preparing in advance a high-level schedule of the anticipated production workload.

The preproduction plan contains:

- The job stream instances to be run during the covered time interval.
- The external follows dependencies that exist between the job streams and jobs included in different job streams.

A job or job stream that cannot start before another specific external job or job stream is successfully completed is named *successor*. An external job or job stream that must complete successfully before the successor job or job stream can start is named *predecessor*.

Tivoli Workload Scheduler automatically generates, expands, and updates, if necessary, the preproduction plan by performing the following steps:

- Removes the job stream instances in COMPLETE and CANCEL states.
- Selects all the job streams scheduled after the end of the current production plan and generates their instances.
- Resolves all job and job stream dependencies, including external follows dependencies, according to the defined matching criteria.

To avoid any conflicts the database is locked during the generation of the preproduction plan and unlocked when the generation completes or if an error condition occurs.

At this stage only the job streams with the time they are scheduled to start and their dependencies are highlighted. All the remaining information about the job streams and the other scheduling objects (calendars, prompts, domains, workstations, resources, files, and users) that will be involved in the production plan for that time period are not included, but are retrieved from the database as soon as the production plan is generated.

When the production plan is extended, old job stream instances are automatically removed. The criteria used in removing these instances takes into account this information:

- The first job stream instance that is not in COMPLETE state at the time the new plan is generated (FNCJSI). This job stream instance can be both a planned instance, that is an instance added to the plan when the production plan is generated, and a job stream instance submitted from the command line during production using the **conman sbs** command.
- The time period between the time FNCJSI is planned to start and the end time of the old production plan.

Assuming **T** is this time period, the algorithm used to calculate which job stream instances are removed from the preproduction plan is the following:

if T < 7
: All job stream instances older than 7 days from the start time of the new production plan are removed from the preproduction plan; all job stream instances closer than 7 days to the start time of the new production plan are kept regardless of their states.

if T > 7
: All job stream instances older than FNCJSI are removed from the preproduction plan; all job stream instances younger than FNCJSI are kept.

This algorithm is used to ensure that the preproduction plan size does not increase continuously and, at the same time, to ensure that no job stream instance that is a potential predecessor of a job stream newly added to the new preproduction plan is deleted.

Note: In the Tivoli Workload Scheduler for z/OS terminology the concept that corresponds to the preproduction plan is *long term plan* (LTP).

Identifying job stream instances in the plan

In earlier versions than 8.3 the plan had a fixed duration of one day. Since version 8.3 the plan can cover a period lasting several days or less than one day. This change has added the possibility to have in the same plan more than one instance of the same job stream with the same name, and also the need to define a new convention to uniquely identify each job stream instance in the plan. Each job stream instance is identified in the plan by the following values:

workstation
: Specifies the name of the workstation on which the job stream is scheduled to run.

jobstreamname
: Corresponds to the job stream name used in earlier versions of Tivoli Workload Scheduler.

scheddateandtime
: Represents when the job stream instance is planned to start in the preproduction plan. It corresponds to the day specified in the run cycle set in the job stream definition by an **on** clause and the time set in the job stream definition by an **at** or **schedtime** keyword. If set, the **schedtime** keyword is used only to order chronologically the job stream instances in the preproduction plan while, if set, the **at** keyword also represents a dependency for the job stream. For more information about these keywords refer to “on” on page 206, “at” on page 188 and “schedtime” on page 215.

Together with these two values that you can set in the job stream definition, Tivoli Workload Scheduler generates and assigns a unique alphanumeric identifier to each job stream instance, the ***jobstream_id***, for its internal processing. For more information on the format of the *jobstream_id* refer to “showjobs” on page 360.

You can use any of the two types of identifiers, *workstation#jobstreamname* and *scheddateandtime* instead of *workstation#jobstream_id*, to uniquely identify a job stream instance when managing job streams in the plan using the **conman** command-line program. The default convention used to identify a job stream instance, both in this guide and in the command-line interfaces of the product, is the one that uses *workstation#jobstreamname* and *scheddateandtime*. For more information on how to specify a job stream instance in a command using **conman**, refer to “Selecting job streams in commands” on page 305.

Managing external follows dependencies for jobs and job streams

During the creation of the preproduction plan, all external follows dependencies to job streams and jobs are resolved using four different possible *matching criteria*:

Same day
: Considering the job or job stream instances planned to run on the same day. In this case you set the clause **follows**...**sameday** in the object definition. Figure 6 on page 53 shows a job stream named `Js1` which has an external follows dependency on the instance of the job stream `Js2` that is scheduled to start on the same day.

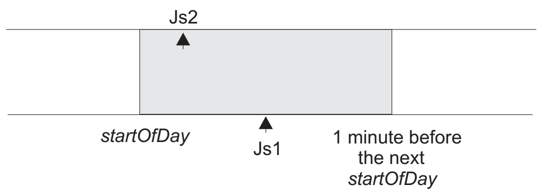


Figure 6. Sameday matching criteria

Below is an example of how to define the involved job streams.

schedule Js2	**schedule Js1**
on everyday	on everyday
at 0700	at 1000
:job2	follows wk1#Js2 sameday
end	:job1
	end

The job stream `Js1` in not launched until the job stream instance of `Js2` on the workstation `wk1` completes successfully.

Closest preceding

Using the closest job or job stream instance (earlier or same time). The job or job stream instance that Tivoli Workload Scheduler uses to resolve the dependency is the closest in time before the instance that includes the dependency. In this case you set the **follows ... previous** clause in the object definition. Figure 7 shows a job stream named `Js1` which has an external follows dependency on the closest earlier instance of job stream `Js2`. The time frame where the predecessor is searched is greyed out in the figure.

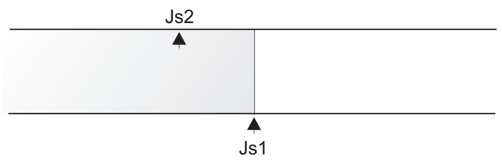


Figure 7. Closest preceding matching criteria

Below is an example of how to define the involved job streams.

schedule Js2	**schedule Js1**
on Th	on Fr
at 0700	at 1000
:job2	follows wk1#Js2 previous
end	:job1
	end

The job stream `Js1` in not launched until the closest preceding job stream instance of `Js2` on the workstation `wk1` completes successfully.

Within a relative interval

Considering the job or job stream instances defined in a range with an offset relative to the start time of the dependent job or job stream. For example, from 25 hours before the dependent job stream start time to 5 hours after the dependent job stream start time. In this case you set the **follows ... relative from ... to ...** clause in the object definition. Figure 8 on page 54 shows a job stream named `Js1` which has an external follows dependency on the job stream instance of `Js2` that starts with an offset of 2 hours with respect to Js1. The job or job stream instance that Tivoli Workload Scheduler considers to resolve the dependency is the closest one within the relative time interval you chose.

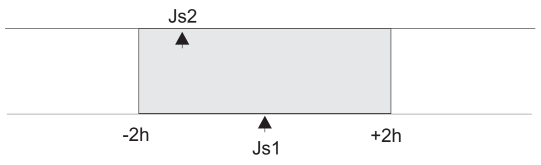


Figure 8. Within a relative interval matching criteria

Below is an example of how to define the involved job streams.

schedule Js2	**schedule Js1**
on everyday	on everyday
at 0900	at 1000
:job2	follows wk1#Js2 relative from 0200 to 0200
end	:job1
	end

The job stream Js1 in not launched until the job stream instance of Js2 on the workstation wk1 that runs in the 08:00 to 12:00 time frame completes successfully.

Within an absolute interval
Using only the job or job stream instances defined in a range. For example from today at 6:00 a.m. to the day after tomorrow at 5:59 a.m. In this case you set the **follows ... from ... to ...** clause in the object definition. Figure 9 shows a job stream named Js1 which has an external follows dependency on the instance of job stream Js2 that is positioned in the preproduction plan between 7 a.m. and 9 a.m. The job or job stream instance that Tivoli Workload Scheduler considers to resolve the dependency is the closest one within the absolute time interval you chose. The time interval specifies the time of the day on which the interval starts and ends, either on the same day as the instance that include the dependency or on a day defined relative to that day.

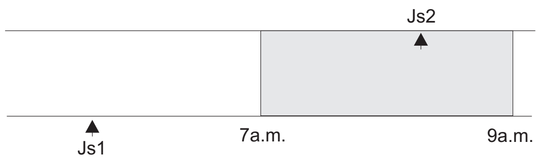


Figure 9. Within an absolute interval matching criteria

Below is an example of how to define the involved job streams.

schedule Js1	**schedule Js2**
on everyday	on everyday
at 0800	at 1000
:job2	follows wk1#Js1 from 0700 to 0900
end	:job1
	end

The job stream Js1 in not launched until the job stream instance of Js2 on the workstation wk1 that runs in the 07:00 to 09:00 time frame on the same day completes successfully.

Regardless of which matching criteria are used, if multiple instances of potential predecessor job streams exist in the specified time interval, the rule used by the product to identify the correct predecessor instance is the following:

1. Tivoli Workload Scheduler searches for the closest instance that precedes the depending job or job stream start time. If such an instance exists, this is the predecessor instance.
2. If there is no preceding instance, Tivoli Workload Scheduler considers the correct predecessor instance as the closest instance that starts after the depending job or job stream start time.

This behavior applies for external follows dependencies between job streams. For external follows dependencies of a job stream or job from another job the criteria are matched by considering the start time of the job stream hosting the predecessor job instead of the start time of the predecessor job itself. Figure 10 shows in bold the instances of `job1` the successor job or job stream is dependent on.

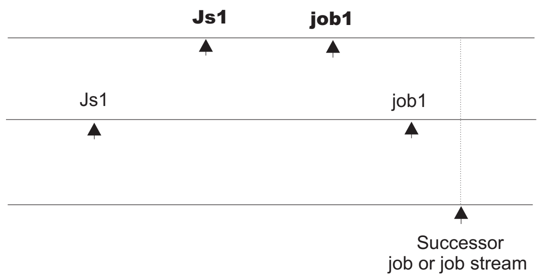


Figure 10. Closest preceding predecessor job

External follows dependencies are identified between jobs and job streams stored in the database whose instances are added to the preproduction plan when the preproduction plan is automatically created or extended. Job and job stream instances submitted in production from the **conman** command line are written in the preproduction plan but they are not used to recalculate predecessors of external follows dependencies already resolved in the preproduction plan.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *jobStreamName.workstationName.jobName* format.

When a job stream includes a job with a follows dependency that shares the same job stream name (for example, job stream `schedA` includes a job named `job6` that has a follows dependency on `schedA.job2`), the dependency is added to the plan as an *external* follows dependency. Since Version 8.3, unlike in previous versions, because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

A job or job stream not yet included in the production plan, but that can be a potential predecessor of instances of jobs and job streams added to the production plan as the production plan is extended, is called a ***pending predecessor***. A pending predecessor is like a dummy occurrence created by the planning process to honor a dependency that has been resolved in the preproduction plan, but that cannot be resolved in the current production plan because the predecessor's start time is not within the current production plan end time. Figure 11 shows how a pending predecessor and its successor are positioned in the preproduction plan.

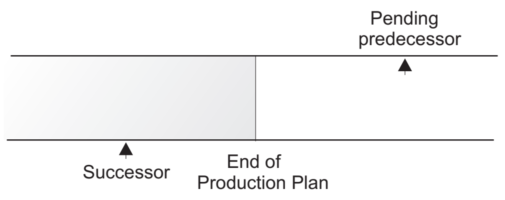


Figure 11. Pending predecessor instance

The way in which pending predecessors are managed is strictly linked to whether or not the successor job or job stream is carried forward:

- If the successor is carried forward when the production plan is extended, the predecessor is included in the new production plan and the dependency becomes current. A pending predecessor job or job stream is marked with a `[P]` in the `Dependencies` column in the output of the conman showjobs and conman showschedules commands.
- If the successor is not carried forward when the production plan is extended, the predecessor is included in the new production plan, but the dependency becomes ***orphaned***. This can happen, for example, if, when extending the production plan, the successor is carried forward and the pending predecessor is not added to the plan because it was flagged as *draft* in the database. The orphaned dependencies are marked with a `[O]` in the `Dependencies` column in the output of the conman showjobs command. When dealing with an orphaned dependency you must verify if it can be released and, if so, cancel it.

Note that when a Tivoli Workload Scheduler network includes agents running on versions older than 8.3 and the `enLegacyId` option is set to `yes` on the master domain manager, having multiple instances of a job stream as pending predecessors produces errors caused by identification problems at submission time.

External follows dependency resolution and status transition examples

This section includes examples for each of the four matching criteria described in the previous paragraphs. In all the examples, the start of day time (SOD) is set to 06:00 AM.

Same day

The job or job stream instance to be considered in resolving the dependency is the closest one on the same day in which the instance that includes the dependency is scheduled to run. In this example, two job streams, `Js1` and `Js2`, each have one job. Job stream `Js1` is scheduled to run every day at 08:00 and on Thursdays also at 07:00. `Js1.Job1` runs at 09:00. Job stream `Js2` has no time restrictions and is scheduled by default at the defined start of day time. `Js2.Job2` is scheduled to run at 15:00 and has an external follows dependency on the closest earlier instance of the job stream `Js1` running on the same day. The two job streams are defined in this way:

```
SCHEDULE MY_MASTER#JS1
ON RUNCYCLE RULE1 "FREQ=WEEKLY;BYDAY=TH"
(AT 0700)
ON RUNCYCLE RULE2 "FREQ=DAILY"
```

```
(AT 0800)
:
MY_MASTER#JOB1
AT 0900
END

SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE2 "FREQ=DAILY;"
FOLLOWS MY_MASTER#JS1.@ SAMEDAY
:
MY_MASTER#JOB2
AT 1500
END
```

When the schedules are included in the plan, the sequence of graphics illustrate how the dependency is resolved:

1. On Thursdays, the instance of `Js2` scheduled at 06:00 depends on the instance of `Js1` scheduled to run at 07:00. On any other day of the week, `Js2` has a dependency on the instance of `Js1` scheduled at 08:00. Figure 12 shows the status of the two job streams in the plan at 06:00 (SOD) on Thursday:

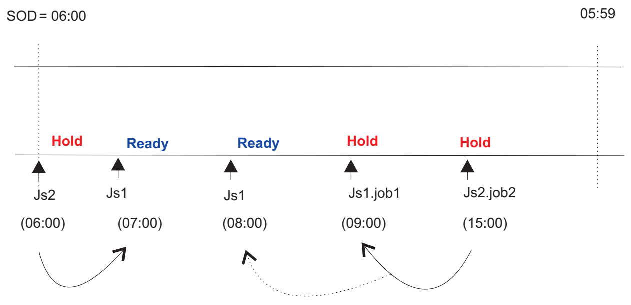


Figure 12. Sameday matching criteria - Step 1: at Start of Day (SOD) on a Thursday

2. At 09:00, `Js1.job1` starts and `Js1` changes status. `Js2.job2` is held until its scheduled time.Figure 13 shows the status of the job streams in the plan at 09:00.

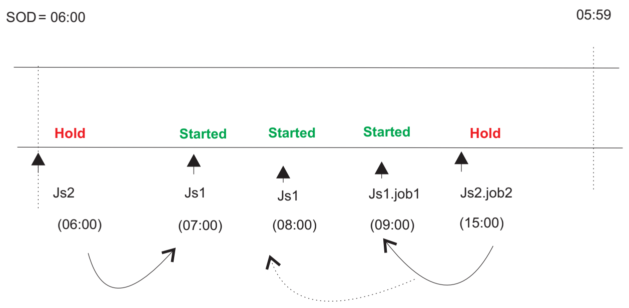


Figure 13. Sameday matching criteria - Step 2: at 9:00

3. On Thursdays at 15:00, `Js2` changes to ready status and `Js2.job2` starts. Figure 14 on page 58 shows the status of the two job streams in the plan at 15:00.

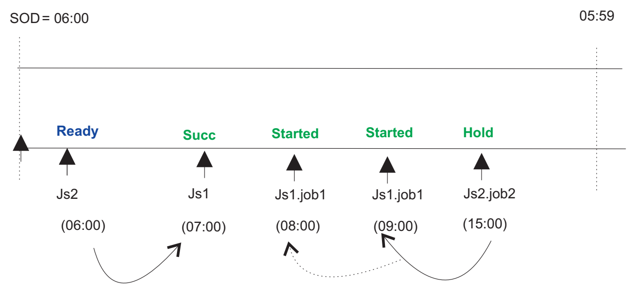


Figure 14. Sameday matching criteria - Step 3: at 15:00

Closest preceding

- In this example, two job streams, `Js1` and `Js2`, each have one job. The job in `Js2` has an external follows dependency on the closest preceding instance of the job in `Js1`. The two job streams are defined in this way:

```
SCHEDULE MY_MASTER#JS1
ON RUNCYCLE RULE1 "FREQ=DAILY;"
(AT 0800)
ON RUNCYCLE RULE2 "FREQ=WEEKLY;BYDAY=TH,FR"
(AT 0900)
:
MY_MASTER#JOB1
END

SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE1 "FREQ=DAILY;"
(AT 1200)
FOLLOWS MY_MASTER#JS1.@ PREVIOUS
:
MY_MASTER#JOB2
AT 1500
END
```

  Job stream `Js1` runs every day at 0800 and on Thursdays and Fridays also at 0900. Job stream `Js2` runs every day at 1200, and has an external dependency on the closest preceding instance of `Js1`. When the job streams are included in the plan, the sequence of graphics illustrates how the dependency is resolved:

  1. Before 12:00 on Thursdays and Fridays, there are two instances of `Js1.Job1`. Job stream `Js2` has a dependency on the instance of `Js1.Job1` that is scheduled to run at 09:00, because it is the closest preceding in terms of time. Figure 15 shows the status of the two job streams in the plan on Thursdays and Fridays.

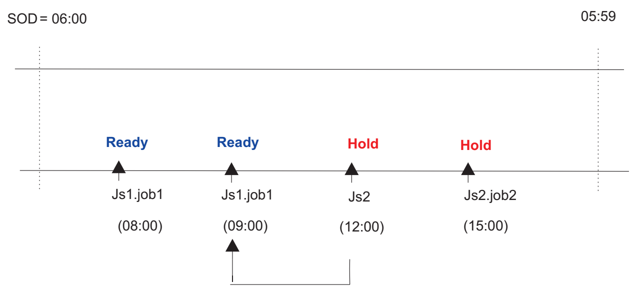


Figure 15. Closest preceding matching criteria - Step 1: before 08:00

2. On any other day of the week, the only instance of `Js1.Job1` in plan, is the one scheduled to run at 08:00. In this case, `Js2` has a dependency on this instance. When `Job1` completes successfully, the status of `Js2` becomes **Ready**. Figure 16 shows the status of the two job streams in the plan on any other weekday except Thursdays and Fridays.

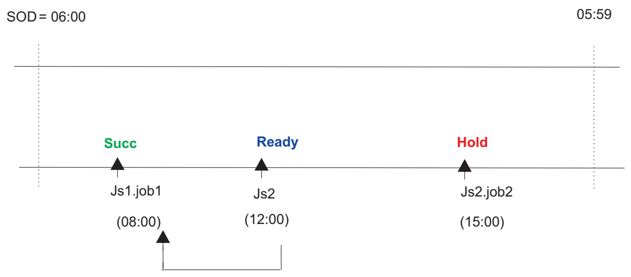


Figure 16. Closest preceding matching criteria - Step 2: at 08:00 on weekdays except Thursdays and Fridays

3. On Thursdays and Fridays at 09:00, the second instance of `Js1.Job1` completes successfully. Job stream `Js2` changes to **Ready**. `Js2.Job2` is held until its scheduled start time. Figure 17 shows the status of the two job streams in the plan.

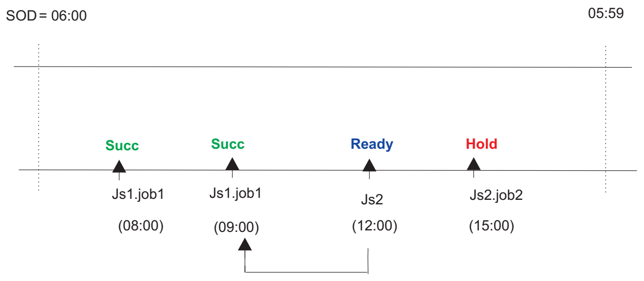


Figure 17. Closest preceding matching criteria - Step 3: at 09:00 on Thursdays and Fridays

4. At 15:00 the time dependency of `Js2.Job2` is satisfied and `Job2` starts. Figure 18 shows the status of the two job streams in the plan at 15:00.

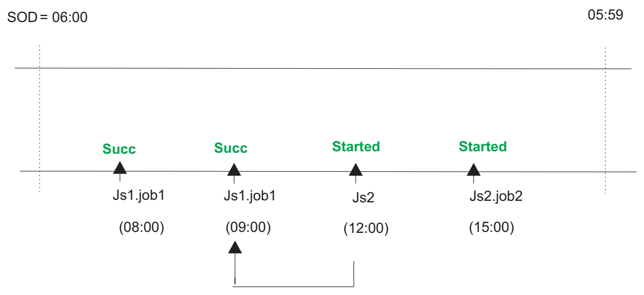


Figure 18. Closest preceding matching criteria - Step 4: at 15:00 on every day

In the job stream definition, run cycle *Rule1* can be substituted by the keywords `ON EVERYDAY`.

- In this second example, the difference between the use of `sameday` and `closest preceding` matching criteria in a plan is described. Job stream

Js1 runs every Friday at 0900, while job stream Js2 and Js3 run every Saturday at 0900. The three job streams are defined in this way:

```
SCHEDULE ACCOUNTING#JS1
ON RUNCYCLE RULE1 "FREQ=WEEKLY;BYDAY=FR"
:
ACCOUNTING#JOB1
 AT 0900
END

SCHEDULE ACCOUNTING#JS2
ON RUNCYCLE RULE2 "FREQ=WEEKLY;BYDAY=SA"
FOLLOWS ACCOUNTING#JS1.@ PREVIOUS
:
ACCOUNTING#JOB1
 AT 0900
END

SCHEDULE ACCOUNTING#JS3
ON RUNCYCLE RULE2 "FREQ=WEEKLY;BYDAY=SA"
FOLLOWS ACCOUNTING#JS1.@
:
ACCOUNTING#JOB1
 AT 0900
END
```

Job stream Js2 has an external dependency on the closest preceding instance of Js1, which is resolved as described in the previous example. Job stream Js3 is defined with sameday matching criteria, so it does not have any dependency on job stream Js1, because Js1 is not defined to run on the same day as Js2.

Within a relative interval

In this example, the job or job stream instance considered to resolve the dependency is the closest one in a time interval of your choice, which is defined relatively to the time when the instance that includes the dependency is scheduled to run. Job stream Js1 is scheduled to run every day at 15:00 and on Thursdays also at 08:00. Js2 is scheduled to run every day at 13:00 and on Thursdays also at 06:00, because no specific time is defined in the run cycle, it is scheduled at start of day time. Js2 uses the relative interval criteria (-04:00 to +04:00) to determine which instance is used to solve the dependency. The interval is based on the time the job stream enters the plan. The job streams are defined as follows:

```
SCHEDULE MY_MASTER#JS1
ON RUNCYCLE RULE1 "FREQ=WEEKLY;BYDAY=TH"
(AT 0800)
ON RUNCYCLE RULE2 "FREQ=DAILY"
(AT 1500)
:
MY_MASTER#JOB1
END

SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE3 "FREQ=WEEKLY;BYDAY=TH"
ON RUNCYCLE RULE2 "FREQ=DAILY;"
(AT 1300)
FOLLOWS MY_MASTER#JS1.@
RELATIVE FROM -0400 TO 0400
:
MY_MASTER#JOB2
AT 1300
END
```

At plan creation time, **conman showjobs** produces the following output:

```
%sj @#@
                                                (Est)  (Est)
CPU      Schedule SchedTime  Job      State Pr Start  Elapse RetCode Deps
MY_MASTER#JS1      0800 11/13 ******** READY 10       (00:06)
                             JOB1     HOLD  10       (00:06)
MY_MASTER#JS1      1500 11/13 ******** READY 10       (00:06)
                            JOB1     HOLD  10       (00:06)
MY_MASTER#JS2      0600 11/13 ******** HOLD  10                JS1(0800 11/13/09).@
                             JOB2     HOLD  10(13:00)
MY_MASTER#JS2      1300 11/13 ******** HOLD  10(13:00)       JS1(1500 11/13/09).@
                             JOB2     HOLD  10(13:00)
```

Figure 19 shows the status of the job streams in the plan at start of day on Thursday.

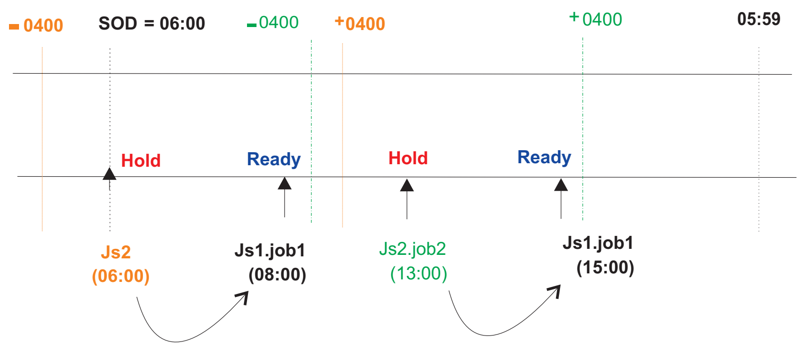


Figure 19. Relative Interval matching criteria - at start of day on Thursday

The instance of `Js2` scheduled at 06:00 has a dependency on `Js1.job1` which is scheduled at 08:00, within the relative interval based on the scheduled time (06:00). `Js2.job2` depends on the instance of `Js1.job1` within the relative interval based on the scheduled time (13:00). When the instance of `Js1.job1` starts at 08:00, the status of`Js2` changes to Ready. From this point onwards, the sequence in which the job streams and jobs run follows the typical process.

Within an absolute interval

In this example, the job or job stream instance considered to resolve the dependency is the closest one in a fixed time interval of your choice. The time interval specifies the time of day on which the interval begins and the time of day on which it ends, either on the same day as the instance that includes the dependency, or on a day defined relatively to that date. `Js1` is scheduled to run every day at 08:00 and on Thursdays also at 07:00. Job `Js1.job1` is scheduled to run at 09:00. Job stream `Js2` is scheduled every day at 10:00 and on Thursdays also at start of day (06:00) and has a dependency on `Js1` based on the absolute interval occurring on the same day between 06:00 and 11:00. The job streams are defined as follows:

```
SCHEDULE MY_MASTER#JS1
ON RUNCYCLE RULE1 "FREQ=WEEKLY;BYDAY=TH"
(AT 0700)
ON RUNCYCLE RULE2 "FREQ=DAILY"
(AT 0800)
:
MY_MASTER#JOB1
AT 0900
END

SCHEDULE MY_MASTER#JS2
ON RUNCYCLE RULE3 "FREQ=WEEKLY;BYDAY=TH"
ON RUNCYCLE RULE2 "FREQ=DAILY;"
```

```
(AT 1000)
FOLLOWS MY_MASTER#JS1.@ FROM 0600 TO 1100
:
MY_MASTER#JOB2
AT 1300
END
```

At plan creation time, **conman showjobs** produces the following output:

```
%sj @#@
                                               (Est)  (Est)
CPU      Schedule SchedTime  Job      State Pr Start  Elapse RetCode Deps
MY_MASTER#JS1      0700 11/13********* READY 10      (00:06)
                             JOB1     HOLD  10(09:00)(00:06)
MY_MASTER#JS1      0800 11/13 ********* READY 10      (00:06)
                             JOB1     HOLD  10(09:00)(00:06)
MY_MASTER#JS2      0600 11/13 ********* HOLD  10              JS1(0700 11/13/09).@
                             JOB2     HOLD  10(15:00)
MY_MASTER#JS2      1000 11/13 ********* HOLD  10(10:00)       JS1(0800 11/1309).@
                             JOB2     HOLD  10(15:00)
```

Figure 20 shows the status of the job streams in the plan at start of day on Thursday.

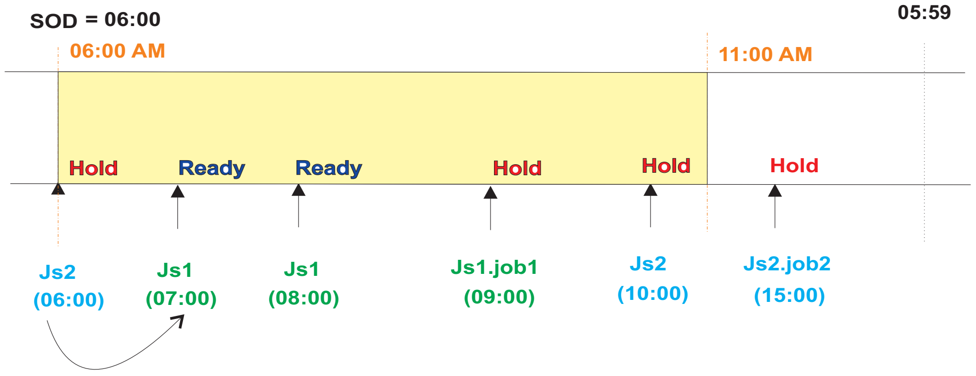


Figure 20. Absolute interval matching criteria - at start of day on Thursday

At 09:00, `Js1.job1` starts, and at 10:00 the dependency is released and `Js2` becomes ready. From this point onwards, the sequence is the same as described in the previous matching criteria.

Production plan

After having created or updated the preproduction plan, Tivoli Workload Scheduler completes the information stored in the preproduction plan with the information stored in the database about the operations to be performed in the selected time period and the other scheduling objects involved when processing the plan and copies it in a new `Symphony` file. It also adds in this file the cross-plan dependencies, such as job streams carried forward from the production plan already processed, and archives the old `Symphony` file in the `schedlog` directory.

At the end of this process the new `Symphony` file contains all the information implementing the new production plan.

A copy of the new `Symphony` file is distributed to all the workstations involved in running jobs or job streams for that production plan.

In the security file the user authorization required to generate the production plan is the *build* access keyword on the `prodsked` and `Symphony` files.

Note: To avoid running out of disk space, keep in mind that each job or job stream instance increases the size of the `Symphony` file by 512 bytes.

For information on how to generate the production plan refer to "Creating and extending the production plan" on page 70.

Understanding carry forward options

Job streams are carried forward when the production plan is generated. How the job stream is carried forward depends on:

- The **carryforward** keyword in the job stream. See "carryforward" on page 190.
- The **enCarryForward** global option. See *IBM Tivoli Workload Scheduler Administration Guide*.
- The **stageman -carryforward** command-line keyword. See "The stageman command" on page 83.
- The **carryStates** global option. See *IBM Tivoli Workload Scheduler Administration Guide*.

Table 8 shows how the carry forward global options work together.

Table 8. Carry forward global options settings

Global options	Carry forward operation
enCarryForward=all **carryStates=()**	Job streams are carried forward only if they did not complete. All jobs are carried forward with the job streams. This is the default setting.
enCarryForward=no	No job streams are carried forward. If this option is set to no, running jobs are moved to the USERJOBS job stream.
enCarryForward=yes **carryStates=(***states***)**	Job streams are carried forward only if they have both jobs in the specified states and the **carryforward** keyword set in the job stream definition. Only the jobs in the specified states are carried forward with the job streams.
enCarryForward=yes **carryStates=()**	Job streams are carried forward only if they did not complete and have the **carryforward** keyword set in the job stream definition. All jobs are carried forward with the job streams.
enCarryForward=all **carryStates=(***states***)**	Job streams are carried forward only if they have jobs in the specified states. Only jobs in the specified states are carried forward with the job streams.

Table 9 shows the result of the carry forward setting based on how the **enCarryForward** global option and the **stageman -carryforward** keywords are set.

Table 9. Resulting carry forward settings

enCarryForward	stageman -carryforward	Resulting carry forward setting
NO	YES	NO
NO	ALL	NO
YES	NO	NO
ALL	NO	NO
ALL	YES	ALL
YES	ALL	ALL

Table 9. Resulting carry forward settings (continued)

enCarryForward	stageman -carryforward	Resulting carry forward setting
YES	YES	YES

The carry forward option set in the job stream definition is persistent. This means that an unsuccessful job stream that is marked as **carryforward**, continues to be carried forward until one of the following occurs:
- It ends in a SUCC state
- Its UNTIL time is reached
- It is cancelled.

The carried forward job stream naming convention is affected by the value assigned to the global option ***enLegacyId***. For more information on the different settings for this option, refer to "Customizing plan management using global options" on page 66.

Note: Regardless of how carry forward options are set, job streams that do not contain jobs are not carried forward.

If you set **`carryStates=(succ)`** and either **enCarryForward=all** or **enCarryForward=yes**, then the next time you run **JnextPlan** there will be misalignment between the preproduction plan and the new `Symphony` file. This happens because, the preproduction plan does not contain the instances of job streams that ended successfully, but the new `Symphony` file does. The result of this misalignment is that dependencies are not resolved according to carried forward successful job stream instances because they no longer exist in the preproduction plan.

The decision to carry forward a repetitive job, that is a job that contains an **every** time setting in its definition, or a chain of rerun jobs is based on the state of its most recent run. Only the first job and the last job of the chain are carried forward.

Trial plan

A trial plan is a projection of what a production plan would be if it covered a longer period of time. For example, if you generate a production plan that covers two days, but you want to know what the plan would be if it covered three days you can generate a trial plan.

These are the characteristics of a trial plan:
- Its start date matches:
  - The preproduction plan start date.
  - The production plan end date.
- It is based on the static information stored in the current preproduction plan.
- It cannot be run to manage production.
- It can be managed by users with access *build* on **trialsked** file object type set in the security file on the master domain manager.
- It produces a file stored in the `schedlog` directory with these properties:
  - The same format as the `Symphony` file.
  - The file name starts with a leading T.

Trial plan generations may result in the extension of the preproduction plan end time. This depends on the settings of the minLen and maxLen global options. When this happens, the database is locked and only unlocked when the operation completes.

There is no restriction on the time period selected for a trial plan, but the size of the resulting file containing all trial plan information must be taken into account.

Because the trial plan is based on the static information stored in the preproduction plan, it does not take into account any dynamic updates made to the `Symphony` file while the production plan is being processed, so all the job streams it contains are in one of these two states:

HOLD
: If they are dependent on other job streams or if their start time is later than the start of plan time.

READY
: If they are free from dependencies and their start time has elapsed.

The operations that can be performed on a trial plan on the master domain manager are:

creation
: Used to create a trial plan to have an overview of production when a production plan does not yet exist.

extension
: Used to create a trial plan of the extension of the current production plan to have an overview of how production evolves in the future.

For information on how to create or extend a trial plan, refer to "Planman command line" on page 73.

Forecast plan

The *forecast plan* is a projection of what the production plan would be in a chosen time frame. For example, if you generated a production plan that covers two days and you want to know what the plan would be for the next week you can generate a forecast plan.

These are the characteristics of a forecast plan:

- It covers any time frame, in the future, in the past or even partially overlapping the time period covered by the current production plan.
- It is based on a sample preproduction plan covering the same time period selected for the forecast plan. This sample preproduction plan is deleted after the forecast plan is created.
- It cannot be run to manage production.
- It can be managed by users with access *build* on **trialsked** file object type set in the security file on the master domain manager.
- It produces a file stored in the `schedlog` directory with these properties:
  - The same format as the `Symphony` file.
  - The file name starts with a leading F.
- When workload service assurance is enabled, it can calculate the predicted start time of each job in the job stream. You can enable and disable this feature using the **enForecastStartTime** global option. Tivoli Workload Scheduler calculates the

average run duration for each job based on all previous runs. For complex plans, enabling this feature could negatively impact the time taken to generate the forecast plan.

While creating a forecast plan the database is locked, and only unlocked when the operation completes.

There is no restriction on the time period selected to generate a forecast plan, but the size of the resulting file containing all forecast plan information must be taken into account.

Because the forecast plan is based on the static information stored in the database, it does not take into account any dynamic updates made to the `Symphony` file while the production plan is being processed or the preproduction plan, so all the job streams it contains are in one of these two states:

HOLD
If they are dependent on other job streams or if their start time is later than the start of plan time.

READY
If they are free from dependencies and their start time has elapsed.

The operation that can be performed on a forecast plan on the master domain manager is:

creation
It is used to create a forecast plan to have an overview of production in a chosen time frame.

For information on how to create a forecast plan, refer to "Planman command line" on page 73.

Customizing plan management using global options

You can customize some criteria for Tivoli Workload Scheduler to use when managing plans by setting specific options on the master domain manager using the **optman** command-line program. You need to generate the plan again to activate the new settings. The options you can customize are:

Properties impacting the generation of the preproduction plan:

minLen
It is the minimum length, calculated in days, of the preproduction plan which is left, as a buffer, after the end of the newly generated production plan. The value assigned to this option is used when the script **UpdateStats** is run from within **JnextPlan**. The value can be from 7 to 365 days. The default is 8 days.

maxLen
It is the maximum length, calculated in days, of the preproduction plan which is left, as a buffer, after the end of the newly generated production plan. The value can be from 8 to 365 days. The default is 8 days.

If the values of `minLen` and `maxLen` are equal, the preproduction plan is updated during the `MakePlan` phase. In general, the value of `maxLen` should exceed the value of `minLen` by at least 1 day, so that the preproduction plan can be updated during the `UpdateStats` phase.

Properties impacting the generation or extension of the production plan:

startOfDay
: It represents the start time of the Tivoli Workload Scheduler processing day in 24-hour format: hhmm (0000-2359). The default setting is 0000.

enCarryForward
: This is an option that affects how the **stageman** command carries forward job streams. Its setting determines whether or not job streams that did not complete are carried forward from the old to the new production plan. The available settings for *enCarryForward* are **yes, no,** and **all**. The default setting is **all.**

carryStates
: This is an option that affects how the **stageman** command manages jobs in carried forward job streams. Its setting determines, based on their state, the jobs to be included in job streams that are carried forward. For example if :

```
carryStates='abend exec hold'
```

then all jobs that are in states `abend`, `exec`, or `hold` are included in carried forward job streams. The default setting is:

```
carryStates=null
```

that means that all jobs are included regardless of their states.

enCFInterNetworkDeps
: This is an option that affects how the **stageman** command manages internetwork dependencies. Enter **yes** to have all `EXTERNAL` job streams carried forward. Enter **no** to completely disable the carry forward function for internetwork dependencies. The default setting is **yes**.

enCFResourceQuantity
: This is an option that affects how the **stageman** command manages resources. When the production plan is extended, one of the following situations occurs:

- A resource not used by any of the job streams carried forward from the previous production plan is referenced by new job or job stream instances added to the new production plan. In this case the quantity of the resource is obtained from the resource definition stored in the database.
- A resource used by one or more job streams carried forward from the previous production plan is not referenced by job or job stream instances added to the new production plan. In this case the quantity of the resource is obtained from the old `Symphony` file.
- A resource used by one or more job streams carried forward from the previous production plan is referenced by job or job stream instances added to the new production plan. In this case the quantity of the resource that is taken into account is based on the value assigned to the *enCFResourceQuantity* option:

If *enCFResourceQuantity* **is set to YES**
: The quantity of the resource is obtained from the old `Symphony` file.

If *enCFResourceQuantity* **is set to NO**
: The quantity of the resource is obtained from the resource definition stored in the database.

The default setting is **yes**.

enEmptySchedsAreSucc
: This option rules the behavior of job streams that do not contain jobs. The available settings are:

  yes The jobs streams that do not contain jobs are marked as SUCC as their dependencies are resolved.

  no The jobs streams that do not contain jobs remain in READY state.

enPreventStart
: This is an option to manage, for multiple day production plan, any job streams without an **at** time constraint set. It is used to prevent job stream instances not time dependent from starting all at once as the production plan is created or extended. The available settings are:

  yes A job stream cannot start before the *startOfDay* of the day specified in its scheduled time even if free from dependencies.

  no A job stream can start immediately as the production plan starts if all its dependencies are resolved.

enLegacyId
: This is an option that affects how job streams are named in the plan. Its function is to keep consistency when identifying job streams in the plan in mixed environments with versions of Tivoli Workload Scheduler prior to 8.3 managed by version 8.4 master domain managers. This option is not supported by the Self Service catalog, which ignores it even if its value is set to YES. The value assigned to this option is read either when the production plan is created or extended or when submitting job streams in production using **conman**. The available settings are:

  yes The *jobstream_id* of job stream named *jobstream_name* is set to *jobstream_nameN* where *N* is an incremental number assigned by an internal counter; it is set to *null* if only one instance of that job stream exists in the plan.

  If the job stream named *jobstream_name* is then carried forward, its identifier is set to *CFjobstream_nameN*.

  This setting is useful to keep consistency when managing job streams, even those carried forward, using **conman** when logged on a Tivoli Workload Scheduler 8.2.x agent in a Tivoli Workload Scheduler network with an 8.4 master domain manager.In particular, if the production period is one day and no multiple instances of the same job stream are submitted, the backward compatibility, when managing job streams in production from a Tivoli Workload Scheduler version 8.2.x agent, is complete.

no The job stream identifier *jobstream_id* is generated as described in "showjobs" on page 360. Carried forward job streams keep their original names and identifiers, and they report between braces {} the date when they were carried forward.

logmanSmoothPolicy
This is an option that affects how the **logman** command handles statistics and history. It sets the weighting factor that favors the most recent job run when calculating the normal (average) run time for a job. This is expressed as a percentage. The default setting is **-1**, which means that this property is not enabled.

logmanMinMaxPolicy
This option defines how the minimum and maximum job run times are logged and reported by **logman**. The available settings for the *logmanMinMaxPolicy* option are:

elapsedtime
The maximum and minimum run times and dates that are logged are based only on a job's elapsed time. Elapsed time, expressed in minutes, is greatly affected by system activity. It includes both the amount of time a job used the CPU and the time the job had to wait for other processes to release the CPU. In periods of high system activity, for example, a job might have a long elapsed time, and yet use no more CPU time than in periods of low system activity. The values are updated only if the latest job run has an elapsed time greater than the existing maximum, or less than the existing minimum.

cputime
The maximum and minimum run times and dates that are logged are based only on a job's CPU time. The CPU time is a measure, expressed in seconds, of the actual time a job used the CPU, and it does not include the intervals when the job was waiting. The values are updated only if the latest job run has a CPU time greater than the existing maximum, or less than the existing minimum.

both The elapsed time and CPU time values are updated independently to indicate their maximum and minimum extremes, but the run dates correspond only to the elapsed time values. No record is kept, in this case, of the run dates for maximum and minimum CPU times.

The default setting is **both**.

enTimeZone
By setting the option you enable or disable the management of time zones across the Tivoli Workload Scheduler network. The available settings for the *enTimeZone* option are:

no Disable time zone management. This means that the values assigned to all **timezone** keywords in the definitions are ignored.

yes Enable time zone management. This means that the values assigned to the **timezone** settings are used to calculate the time when the jobs and jobs streams will run on the target workstations.

Refer to"Enabling time zone management" on page 531 for more information about the *enTimeZone* variable.

enLegacyStartOfDayEvaluation
This option affects the way the *startOfDay* variable is managed across the Tivoli Workload Scheduler network. This option requires the *enTimeZone* variable set to **yes** to become operational. The available settings for the *enLegacyStartOfDayEvaluation* option are:

no The value assigned to the *startOfDay* option on the master domain manager is not converted to the local time zone set on each workstation across the network.

yes The value assigned to the *startOfDay* option on the master domain manager is converted to the local time zone set on each workstation across the network.

Refer to "How Tivoli Workload Scheduler manages time zones" on page 532 for more information about the *enLegacyStartOfDayEvaluation* variable.

For information on how to set options using the **optman** command-line program, refer to the *IBM Tivoli Workload Scheduler Administration Guide*.

Creating and extending the production plan

The entire process of moving from an old to a new production plan, including its activation across the Tivoli Workload Scheduler network, is managed by the **JnextPlan** script. You can run **JnextPlan** at any time during the processing day. The new production plan that is generated is immediately activated on the target workstations regardless the time set in the ***startOfDay*** variable.

Note:

1. When you run the **JnextPlan** script, the workstation processes are stopped and restarted on all the workstations across the Tivoli Workload Scheduler network. For more information about workstation processes, refer to Chapter 2, "Understanding basic workstation processes," on page 23.
2. Check the *IBM Tivoli Workload Scheduler Administration Guide* for information on specific scenarios that might require **JnextPlan** customization.

Authorization

The **JnextPlan** command is issued from a command prompt shell on the master domain manager and can be invoked by one of the following users:

- The *TWS_user* user who installed the product on that machine, if not disabled by the settings defined in the security file.
- Either root or Administrator, depending on the operating system installed on that machine, if not disabled by the settings defined in the security file.
- Any Tivoli Workload Scheduler user authorized in the security file on the master domain manager as follows:

```
file name=prodsked,Symphony access=build
```

JnextPlan can be run at any time while the production plan is in process to update in the production plan the topology of the Tivoli Workload Scheduler network in terms of workstation, workstation class and domain object definitions. For example, if you created a new workstation definition in the database and you want to add that workstation definition into the plan to later submit jobs or job streams on that workstation you run the command:

```
JnextPlan -for 0000
```

Note:

- Make sure the ***enCarryForward*** option is set to ALL before running:
  ```
  JnextPlan -for 0000
  ```
- When **JnextPlan** command is run, the *$MANAGER* variable is managed as follows:
  – The variable is resolved if the workstation is a fault-tolerant agent of a version prior to 8.6.
  – The variable is left unresolved for fault-tolerant agent workstations version 8.6.

Syntax

JnextPlan
  [**-from** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]*]
  {**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* |
   -for *[h]hhmm* [**-days** *n*] | **-days** *n*}

Arguments

-from Sets the start time of the production plan. The format of the date is specified in the `localopts` file; where *hhmm* identifies the hours and the minutes and *tz* is the time zone. This flag is used only if a production plan does not already exist. If the **-from** argument is not specified the default value is `"today +startOfDay"`.

If the time zone is not specified, time zone GMT is used by default.

-to Is the production plan end time. The format for the date is the same as that used for the **-from** argument. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

If the time zone is not specified, time zone GMT is used by default.

-for Is the plan extension expressed in time. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with **-to**.

-days *n*
Is the number of days you want to create or extend the production plan for. The **-days** parameter is mutually exclusive with the **-to** parameter.

If no **-to**, **-for**, or **-days** arguments are specified then the default production plan length is one day.

Note: Assuming that the value assigned to *startOfDay* is 00:00 a.m. and that the date format set in the `localopts` file is *mm/dd/yyyy*, if the values set are **-from** `07/05/2011` and **-to** `07/07/2011`, then the plan is created to span the time frame from 07/05/2011 at 00:00 a.m. to 07/06/2011 at 11:59 p.m. and not to 07/07/2011 at 11:59 p.m.

Comments

The **JnextPlan** script can only be run from the master domain manager. It uses the default connection parameters defined in either the `localopts` or `useropts` files (see "Setting up options for using the user interfaces" on page 45). If you want to run **JnextPlan** using different connection parameter settings you can edit the **MakePlan** script and modify the invocation to the **planman** statement as described in "Planman command line" on page 73.

The **JnextPlan** script is composed of the following sequence of commands and specialized scripts, each managing a specific aspect of the production plan generation:

conman startappserver
: This command is invoked to start the WebSphere Application Server if it is not already running.

MakePlan
: This script inherits the flags and the values assigned to them from **JnextPlan**. Its syntax is:

MakePlan [**-from** *mm/dd/[yy]yy[hhmm[****tz*** | ***timezone*** *tzname]]*] {**-to** *mm/dd/[yy]yy[hhmm[****tz*** | ***timezone*** *tzname]]* | **-for** *[h]hhmm* [-**days** *n*] | -**days** *n*}

MakePlan invokes internally the **planman** command line. **MakePlan** performs the following actions:

1. Creates a new plan or extends the current plan and stores the information in an intermediate production plan containing:
   - All the scheduling objects (jobs, job streams, calendars, prompts, resources, workstations, domains, files, users, dependencies) defined in the selected time period.
   - All dependencies between new instances of jobs and job streams and the jobs and job streams existing in the previous production plan.
   - All bind requests whose scheduled time is included in the selected time period.
2. Deletes all bind requests in final state.
3. Prints preproduction reports.

SwitchPlan
: This script invokes internally the **stageman** command. For more information refer to "The stageman command" on page 83. **SwitchPlan** performs the following actions:

1. Stops Tivoli Workload Scheduler processes.
2. Generates the new `Symphony` file starting from the intermediate production plan created by **MakePlan**.
3. Archives the old plan file with the current date and time in the `schedlog` directory.
4. Creates a copy of the `Symphony` file to distribute to the workstations.
5. Restarts Tivoli Workload Scheduler processes which distribute the copy of the `Symphony` file to the workstation targets for running the jobs in plan.

Note: Make sure no **conman start** command is run while the production plan is been processed.

CreatePostReports
This script prints postproduction reports.

UpdateStats
This script invokes internally the **logman** command. For more information refer to "The stageman command" on page 83. **UpdateStats** performs the following actions:

1. Logs job statistics.
2. Checks the policies and if necessary extends the preproduction plan.
3. Updates the preproduction plan reporting the job stream instance states.

Planman command line

The **planman** command line is used to manage *intermediate* production plans, *trial* plans, and *forecast* plans. It is also used to have information about the currently active production plan, to unlock the database entries locked by the plan management processes, and to deploy scheduling event rules. The command runs on the master domain manager. Use the following syntax when running **planman**:

planman -U

planman -V

planman [*connection_parameters*] *command*

where:

-U Displays command usage information and exit.

-V Displays the command version and exit.

connection_parameters
If you are using **planman** from the master domain manager, the connection parameters were configured at installation and do not need to be supplied, unless you do not want to use the default values.

If you are using **planman** from the command line client on another workstation, the connection parameters might be supplied by one or more of these methods:

- Stored in the `localopts` file
- Stored in the `useropts` file
- Supplied to the command in a parameter file
- Supplied to the command as part of the command string

For an overview of these options see "Setting up options for using the user interfaces" on page 45. For full details of the configuration parameters see the topic on configuring the command-line client access in the *Tivoli Workload Scheduler: Administration Guide*.

command
Represents the command you run against plans using the **planman** interface. These are the actions you can perform against plans:

- "Creating an intermediate production plan" on page 74
- "Creating an intermediate plan for a plan extension" on page 75
- "Retrieving the production plan information" on page 76
- "Creating a trial plan" on page 77

- “Creating a trial plan of a production plan extension” on page 78
- “Creating a forecast plan” on page 79
- “Unlocking the production plan” on page 81
- “Resetting the production plan” on page 81

You can also use **planman** to deploy scheduling event rules. The command is explained in:

“Deploying rules” on page 80.

Refer to the related subsections for additional details on these commands.

Creating an intermediate production plan

The **planman** with the **crt** option is invoked from within the **JnextPlan** command in one of these two situations:

- The first time the **JnextPlan** command is run after having installed the product.
- When generating a production plan after having reset the production plan using the **ResetPlan** command.

The result of running this command is the creation of a new intermediate production plan, named `Symnew`, covering the whole time the new production plan that is being generated will cover. The following syntax is used:

planman [*connection_parameters*] **crt**

[**-from** *mm/dd/*[*yy*]*yy* [*hhmm* [**tz** | **timezone** *tzname*]]]

{**-to** *mm/dd/*[*yy*]*yy*[*hhmm*[***tz*** | ***timezone*** *tzname*]] |

-for *[h]hhmm* [-**days** *n*] |

-**days** *n*}

where:

connection_parameters
: Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to “Planman command line” on page 73.

-from
: Sets the start time of the new production plan.

  If the **-from** argument is omitted, then:
  - The default date is *today*.
  - The default hour is the value set in the ***startOfDay*** global option using **optman** on the master domain manager.

-to
: Is the new production plan end time. The format for the date is the same as that used for the **-from** argument. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for
: Is the plan extension expressed in time. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with **-to** argument.

-days *n*

Is the number of days you want to create the production plan for. The **-days** argument is mutually exclusive with the**-to** argument.

Note:

1. Make sure you run the **planman** command from within the **JnextPlan** command.
2. The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no **-to**, **-for**, or **-days** arguments are specified then the default production plan length is one day.

These are some examples of using the **planman** command assuming the date format set in the `localopts` file is mm/dd/yyyy:

1. This command creates the production plan from 03/21/2011 at 23:07 to 03/22/2011 at 23:06 in the local time zone:
```
planman crt –from 03/21/05 2307
```
2. This command creates the production plan from 03/21/2011 at 09:00 to 03/21/2011 at 15:00:
```
planman crt –from 03/21/2011 0900 for 0600
```
3. If today is 03/21/05 and the value set for the *startOfDay* variable stored in the database is 0600, this command creates the production plan from 03/21/2011 at 6:00 to 03/25/2011 at 5:59:
```
planman crt –to 03/25/2011
```
4. This command creates a production plan from 03/21/2011 at 18:05 to 03/24/2011 at 23:00 in the time zone of Europe\Paris:
```
planman crt –from 03/21/2011 1805 tz Europe\Rome
            –to 03/24/2011 2300 tz Europe\Rome
```

Creating an intermediate plan for a plan extension

The **planman** command with the **ext** option is invoked from within the **JnextPlan** command when:

- **JnextPlan** is invoked.
- A production plan, represented by the `Symphony` file on the master domain manager, already exists.

The result of running this command is the creation of a new intermediate production plan, named `Symnew`, covering the extra time the new production plan that is being generated will span. The following syntax is used:

planman [*connection_parameters*] **ext**

{**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* |

-for *[h]hhmm* [-**days** *n*] |

-**days** *n*}

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to “Planman command line” on page 73.

-to Sets the end time of the extended production plan. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for Sets the length of the production plan extension. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with the **-to** argument.

-days *n*
: Sets the number of days you want to extend the production plan for. The **-days** argument is mutually exclusive with the**-to** argument.

Note:

1. Make sure you run the **planman** command from within the **JnextPlan** command.
2. The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.
3. When the production plan is extended the numbers associated to prompts already present in the plan are modified.

If no **-to**, **-for**, or **-days** arguments are specified then the production plan is extended by one day.

Retrieving the production plan information

The following syntax is used to show information about the current production plan:

planman [*connection_parameters*] **showinfo**

where:

connection_parameters
: Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to “Planman command line” on page 73.

Note: You can install the Tivoli Workload Scheduler Command Line Client feature on fault-tolerant agents and systems outside the Tivoli Workload Scheduler network to issue from those systems the **planman showinfo** command.

The output of this command shows:
- The installation path.
- The start time of the production plan.
- The end time of the production plan.
- The duration of the production plan, after the last plan extension, if extended.
- The date and time of the last plan update, done either using **JnextPlan** or **planman**.
- The end time of the preproduction plan.
- The start time of the first not completed job stream instance.
- The *run number*, that is the total number of times the plan was generated.

- The *confirm run number*, that is the number of times the plan was successfully generated.

The start and end times of the production and preproduction plans are displayed using the format specified in the ***date format*** variable set in the `localopts` file and the time zone of the local machine.

A sample output of this command is the following:

```
# planman showinfo
Tivoli Workload Scheduler (UNIX)/PLANMAN 8.6 (20100715)
Licensed Materials - Property of IBM*
5698-WSH
(C) Copyright IBM Corp. 1998, 2011 All rights reserved.
* Trademark of International Business Machines
Installed for user "aix61usr".
Locale LANG set to the following: "en"
Plan creation start time: 07/21/2010 06:00 TZ CEST
Production plan start time of last extension: 07/21/2010 06:00 TZ CEST
Production plan end time: 07/22/2010 05:59 TZ CEST
Production plan time extention: 024:00
Plan last update: 07/21/2010 10:05 TZ CEST
Preproduction plan end time: 08/05/2010 06:00 TZ CEST
Start time of first not complete preproduction plan job stream instance:
   07/21/2010 10:30 TZ CEST
Run number: 1
Confirm run number: 1
```

Creating a trial plan

The following syntax is used to create a trial plan:

planman [*connection_parameters*] **crttrial** *file_name*

[**-from** *mm/dd/*[*yy*]*yy* [*hhmm* [**tz** | **timezone** *tzname*]]]

{**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* |

-for *[h]hhmm* [-**days** *n*] |

-**days** *n*}

where:

connection_parameters
: Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to "Planman command line" on page 73.

file_name
: Assigns a name to the file to be created under the directory `TWS_home/schedTrial` and that contains the trial plan. The file name of the file containing the trial plan is **T*filename***. This means that if the value assigned to *file_name* is `myfile` then the file name that contains the generated trial plan is `Tmyfile`.

-from
: Sets the start time of the trial plan.

  If the **-from** argument is omitted, then:
  - The default date is *today*.

- The default hour is the value set in the ***startOfDay*** global option using **optman** on the master domain manager.

-to
Sets the end time of the trial plan. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for
Sets the length of the trial plan. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with the **-to** argument.

-days *n*
Sets the number of days you want the trial plan to last for. The **-days** argument is mutually exclusive with the **-to** argument.

Note: The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no **-to**, **-for**, or **-days** arguments are specified then the default trial plan length is one day.

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Forecast→Generate Trial Plan**
2. Choose an engine name and write the requested information in the Generate Trial Plan page
3. Click **Generate Plan**.

Creating a trial plan of a production plan extension

The following syntax is used to create a trial plan with the extension of the current production plan:

planman [*connection_parameters*] **exttrial** *file_name*

{**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* |

-for *[h]hhmm* [-**days** *n*] |

-**days** *n*}

where:

connection_parameters
Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to "Planman command line" on page 73.

file_name
Assigns a name to the file to be created under the directory *TWS_home*/`schedTrial` and that contains the trial plan. The file name of the file containing the trial plan is **T*filename***. This means that if the value assigned to *file_name* is `myfile` then the file name that contains the generated trial plan is `Tmyfile`.

-to
Sets the end time of the trial plan containing the production plan extension. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for Sets the length of the trial plan containing the production plan extension. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with the **-to** argument.

-days *n*
Sets the number of days you want the trial plan containing the production plan extension to last for. The **-days** argument is mutually exclusive with the **-to** argument.

Note: The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no **-to**, **-for**, or **-days** arguments are specified then the default production plan extension contained in the trial plan is one day.

Creating a forecast plan

The following syntax is used to create a forecast plan:

planman [*connection_parameters*] **crtfc** *file_name*

[**-from** *mm/dd/*[*yy*]*yy* [*hhmm* [**tz** | **timezone** *tzname*]]]

{**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* |

-for *[h]hhmm* [-**days** *n*] |

-**days** *n*}

where:

connection_parameters
Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to "Planman command line" on page 73.

file_name
Assigns a name to the file to be created under the directory `TWS_home/schedForecast` and that contains the forecast plan. The name of the file containing the forecast plan is **F*filename***. This means that if the value assigned to *file_name* is `myfile` then the file name that contains the generated forecast plan is `Fmyfile`.

The maximum length of *file_name* can be 148 characters.

-from Sets the start time of the forecast plan.

If the **-from** argument is omitted, then:
- The default date is *today*.
- The default hour is the value set in the ***startOfDay*** global option using **optman** on the master domain manager.

-to Sets the end time of the forecast plan. The **-to** argument is mutually exclusive with the **-for** and **-days** arguments.

-for Sets the length of the forecast plan. The format is *hhhmm*, where *hhh* are the hours and *mm* are the minutes. The **-for** argument is mutually exclusive with the **-to** argument.

-days *n*

Sets the number of days you want the forecast plan to last for. The **-days** argument is mutually exclusive with the **-to** argument.

Note: The format used for the date depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

If no **-to**, **-for**, or **-days** arguments are specified then the default forecast plan length is one day.

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Forecast→Generate Forecast Plan**
2. Choose an engine name and write the requested information in the Generate Forecast Plan page
3. Click **Generate Plan**.

Deploying rules

The `planman deploy` command is used in event management. You can use it to manually deploy all rules that are not in draft state (the `isDraft` property is set to `NO` in their definition). The command operates as follows:

1. Selects all event rule definitions not in draft state from the Tivoli Workload Scheduler database.
2. Builds event rule configuration files.
3. Deploys the configuration files to the monitoring engines running on the Tivoli Workload Scheduler agents.

The new configuration files update the event rules running on each monitoring engine in terms of:

- New rules
- Changed rules
- Rules deleted or set back to `draft` state

You can use this command in addition to, or in replacement of, the `deploymentFrequency` (`df`) `optman` configuration option, which periodically checks event rule definitions for changes to deploy (see the *Administration Guide* for details on this option).

The changes applied to the event rule definitions in the database become effective only after deployment has taken place.

The command syntax is:

planman [*connection_parameters*] **deploy** [**-scratch**]

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to "Planman command line" on page 73.

-scratch

Without this option, the command affects only the rules that have been added, changed, deleted, or set back to `draft` state .

With this option, the command deploys all the non-draft rules existing in the database, including the ones that are already in deployment and have not changed.

Note that the deployment time increases proportionally with the number of active rules to deploy. If you need to deploy a large number of new or changed rules, run `planman deploy` with this option to reduce the deployment time.

Use of this option results in a complete reset of the event processor and should be used with caution. The command may cause the loss of any rule instances in progress at the time you issue it. The typical case is a sequential rule that has been triggered and is waiting for additional events to take place: if you use the option at this time, the event rule environment is reset and the tracked events are lost.

To run this command, you need **build** access on the **prodsked** file.

Unlocking the production plan

When Tivoli Workload Scheduler starts to create the production plan, it locks the definitions of scheduling objects in the database and then unlocks them either when the creation of the production plan is finished or if an error condition occurs. The lock is applied to prevent object definitions from being modified when the production plan in generated or extended. If the processing ends abnormally the database entries might remain locked. Only users with **build** access on the **prodsked** file object type specified in the security file on the master domain manager are allowed to unlock the database. The command used to perform this action is:

planman [*connection_parameters*] **unlock**

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to “Planman command line” on page 73.

Note: You can install the Tivoli Workload Scheduler Command Line Client feature on fault-tolerant agents and systems outside the Tivoli Workload Scheduler network to issue from those systems the **planman unlock** command.

Resetting the production plan

The following script is used to either reset or scratch the production plan:

ResetPlan [*connection_parameters*] [**-scratch**]

where:

connection_parameters

Defines the settings to use when establishing the connection using HTTP or HTTPS through WebSphere Application Server to the master domain manager. For more information refer to “Planman command line” on page 73.

The difference between resetting and scratching the production plan is the following:

- If you *reset* the production plan, the preproduction plan is kept, it is updated with job statistics, and it is used later to generate a new production plan. This means that when you create a new production plan, it will contain all job stream instances which were not in COMPLETE state when you run the **ResetPlan**. The steps performed by the product when resetting the production plan are the following:
  1. All Tivoli Workload Scheduler processes are stopped on the master domain manager.
  2. The current `Symphony` file is archived.
  3. The job statistics are updated.
- If you *scratch* the production plan, the preproduction plan is scratched too. The preproduction plan will be created again based on the modeling information stored in the database when you later generate a new production plan. This means that the new production plan will contain all job stream instances scheduled to run in the time frame covered by the plan regardless of whether or not they were already in COMPLETE state when the plan was scratched. The steps performed by the product when scratching the production plan are the following:
  1. All Tivoli Workload Scheduler processes are stopped on the master domain manager.
  2. The current `Symphony` file is archived.
  3. The job statistics are updated.
  4. The preproduction plan is scratched.

  Note: If you use the **-scratch** option, make sure you run **dbrunstats** before the **JnextPlan** script. See the *Administration Guide* for details on `dbrunstats`.

Removing the preproduction plan

The following script is used to remove the preproduction plan, while maintaining the `Symphony` file:

Planman reset -scratch

When you run this command, the preproduction plan is scratched. The preproduction plan will be created again based on the modeling information stored in the database when you later generate a new production plan. This means that the new production plan will contain all job stream instances scheduled to run in the time frame covered by the plan regardless of whether or not they were already in COMPLETE state when the plan was scratched. The steps performed by the product when scratching the production plan are the following:

1. All Tivoli Workload Scheduler processes are stopped on the master domain manager.
2. The `Symphony` file is maintained.
3. The job statistics are updated.
4. The preproduction plan is scratched.

Note: If you use the **-scratch** option, make sure you run **dbrunstats** before the **JnextPlan** script. See the *Administration Guide* for details on `dbrunstats`.

The stageman command

The **stageman** command carries forward uncompleted job streams, archives the old production plan, and installs the new production plan. A copy of `Symphony`, is sent to domain managers and agents as part of the initialization process for the new production plan. When running **JnextPlan**, **stageman** is invoked from within the **SwitchPlan** script.

You must have ***build*** access to the `Symphony` file to run **stageman**.

Syntax

stageman -V | -U

stageman
[**-carryforward**{**yes**|**no**|**all**}]
[**-log** *log_file*| **-nolog**]
[*symnew*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-carryforward
Defines how uncompleted job streams are managed when moving to a new production plan. The available settings are:

no Does not carry forward any job streams.

yes Carries forward only the uncompleted job streams whose definition contains the keyword **carryforward**.

all Carries forward all uncompleted job streams, regardless whether or not contain the keyword **carryforward** in the job stream definition.

If you omit this keyword, by default it is set to the value specified globally using **optman** for the *enCarryForward* option. Refer to "Understanding carry forward options" on page 63 to have information on the resulting carry forward setting when both the *enCarryForward* global option and the **-carryforward** keywords are set.

-log Archives the old production plan in the directory *TWS_home*/`schedlog` with file name *log_file*. The archived production plans can then be listed and selected using the commands "listsym" on page 338 and "setsym" on page 350. If neither **-log** nor **-nolog** keywords are specified, Tivoli Workload Scheduler archives the old production plan using the following naming convention:

```
Myyyymmddhhtt
```

where *yyyymmddhhtt* corresponds to the year, month, day, hour, minutes the old production plan is archived. If you generate the production plan using **JnextPlan** you can customize this naming convention in the **SwitchPlan** script.

Note: Be sure to monitor the disk space in the `schedlog` directory and remove older log files on a regular basis.

-nolog Does not archive the old production plan.

symnew
The name assigned to the intermediate production plan file created by **planman**. If not specified, **stageman** uses the file name `Symnew`.

Comments

To allow carry forward procedures to work properly in a network, the master domain manager's production plan file, `Symphony`, must be updated with the latest job stream status from its agents and subordinate domain managers. Run the following command:

```
conman "link @"
```

before running **stageman**. This command links any unlinked workstations so that messages about job processing status queued in the `Mailbox.msg` file are sent back to the master domain manager to update the `Symphony` file.

Note: In UNIX only, **stageman** also determines which executable files associated to jobs submitted using the **at** and **batch** utility commands can be deleted when Tivoli Workload Scheduler is started for the new production period. These jobs are not carried forward.

Examples

Carry forward all uncompleted job streams (regardless of the status of the Carry Forward option), log the old `Symphony` file, and create the new `Symphony` file:

```
DATE='datecalc today pic YYYYMMDDHHTT'
stageman -carryforward all -log schedlog/M$DATE
```

Carry forward uncompleted job streams as defined by the ***carryforward*** global option, do not log the old `Symphony` file, and create an intermediate production plan named `mysym`:

```
stageman -nolog mysym
```

Managing concurrent accesses to the Symphony file

This section contains two sample scenarios describing how Tivoli Workload Scheduler manages possible concurrent accesses to the `Symphony` file when running **stageman**.

Scenario 1: Access to Symphony file locked by other Tivoli Workload Scheduler processes

If Tivoli Workload Scheduler processes are still active and accessing the `Symphony` file when **stageman** is run, the following message is displayed:

```
Unable to get exclusive access to Symphony.
Shutdown batchman and mailman.
```

To continue, stop Tivoli Workload Scheduler and rerun **stageman**. If **stageman** aborts for any reason, you must rerun both **planman** and **stageman**.

Scenario 2: Access to Symphony file locked by stageman

If you try to access the plan using the command-line interface while the Symphony is being switched, you get the following message:

```
Current Symphony file is old. Switching to new Symphony.
Schedule mm/dd/yyyy (nnnn) on cpu, Symphony switched.
```

Managing follows dependencies using carry forward prompt

To retain continuity when carrying forward job streams, **stageman** generates prompts for each job stream that is carried forwarded and has a `follows` dependency on another job stream which is not carried forward. These prompts are issued after the new processing period begins, when Tivoli Workload Scheduler checks to see if the job or job stream is ready to launch, and are replied to as standard prompts. The following is an example of a *carry forward prompt*:

```
INACT 1(SYS2#SKED2[(0600 01/11/06),(0AAAAAAAAAAAAA2Y)]) follows
      SYS1#SKED1, satisfied?
```

This prompt indicates that a job stream, which is carried forward from the previous production plan, `(SYS2#SKED2[(0600 01/11/06),(0AAAAAAAAAAAAA2Y)])`, has a follows dependency from a job stream named `SYS1#SKED1` which was not carried forward. For information on the syntax used to indicate the carried forward job stream refer to "Selecting job streams in commands" on page 305.

The state of the prompt, **INACT** in this case, defines the state of the corresponding `follows` dependency. The possible states are:

INACT
: The prompt has not been issued and the dependency is not satisfied.

ASKED
: The prompt has been issued, and is awaiting a reply. The dependency is not satisfied.

NO
: Either a "no" reply was received, or it was determined before carry forward occurred that the followed job stream `SKED3` had not completed successfully. The dependency is not satisfied.

YES
: Either a "yes" reply was received, or it was determined before carry forward occurred that the followed job stream `SKED3` had completed successfully. The dependency is satisfied.

The logman command

The **logman** command logs job statistics from a production plan log file.

Syntax

logman -V|-U

logman
    [*connectionParameters*]
    {**-prod** | *symphony-file*]
    [**-smooth** *weighting*]
    [**-minmax** {*elapsed* | *cpu*}]}

Arguments

-U Displays command usage information and exits.

-V Displays the command version and exits.

connectionParameters

Represents the set of parameters that control the interaction between the product interface, **logman** running on the master domain manager in this case, and the WebSphere Application Server infrastructure using HTTP or HTTPS. Use this syntax to specify the settings for the connection parameters:

[**-host** *hostname*] [**-port** *port_number*] [**-protocol** *protocol_name*] [**-proxy** *proxy_name*] [**-proxyport** *proxy_port_number*] [**-password** *user_password*] [**-timeout** *timeout*] [**-username** *username*]

where:

hostname

Is the hostname of the master domain manager.

port_number

Is the port number used when establishing the connection with the master domain manager.

protocol_name

Is the protocol used during the communication. It can be HTTP with basic authentication, or HTTPS with certificate authentication.

proxy_name

Is the proxy hostname used in the connection.

proxy_port_number

Is the proxy port number used in the connection.

user_password

Is the password of the user that is used to run **logman**.

Note: On Windows workstations, when you specify a password that contains double quotation marks (") or other special characters, make sure that the character is escaped. For example, if your password is `tws11"tws`, write it as `"tws11\"tws"`.

timeout

Is the maximum time, expressed in seconds, the connecting command-line program can wait for the master domain manager response before considering the communication request as failed.

username

Is the username of the user running **logman**.

If any of these parameters is omitted when invoking **logman**, Tivoli Workload Scheduler searches for a value first in the `useropts` file and then in the `localopts` file. If a setting for the parameter is not found an error is displayed. Refer to “Setting up options for using the user interfaces” on page 45 for information on `useropts` and `localopts` files.

-prod Updates the preproduction plan with the information on the job streams in COMPLETE state in production. By doing so the preproduction plan is kept up-to-date with the latest processing information. This avoids the possibility of the new production plan running again, job streams already completed in the previous production period.

-minmax {*elapsed* | *cpu*}
Defines how the minimum and maximum job run times are logged and reported. The available settings are:

elapsed
Base the minimum and maximum run times on elapsed time.

cpu
Base the minimum and maximum run times on CPU time.

This setting is used when the **logman** command is run from the command line and not by the **JnextPlan** script. When the **logman** command is run by **JnextPlan**, the setting used is the one specified in the *logmanMinMaxPolicy* global option.

-smooth *weighting*
Uses a weighting factor that favors the most recent job run when calculating the normal (average) run time for a job. This is expressed as a percentage. For example, **-smooth 40** applies a weighting factor of 40% to the most recent job run, and 60% to the existing average. The default is 0%. This setting is used when the **logman** command is run from the command line and not by the **JnextPlan** script. When the **logman** command is run by **JnextPlan**, the setting used is the one specified in the *logmanSmoothPolicy* global option.

symphony-file
The name of an archived symphony file from which job statistics are extracted.

Comments

Jobs that have already been logged, cannot be logged again. Attempting to do so generates a `0 jobs logged` error message.

Examples

Log job statistics from the log file M199903170935:

```
logman schedlog/M199903170935
```

How average run times are calculated

The estimated duration of a job run is provided by logman as part of the daily planning cycle. The estimated duration of a job run is based on the average of its preceding runs. To compute the average run time for a job, logman divides the total run time for all successful runs by the number of successful runs. If a large number of runs is used to compute the average, a sudden change in a job's run time will not immediately be reflected in the average. To respond more quickly to such changes, you can use the `smooth` option so that the average can be weighted in favor of the most recent job runs. Use the `-smooth` option to enter a weighting factor, as a percentage, for current job runs. For example, the `logman -smooth 40` command will cause logman to use a weighting factor of 40 percent for the most recent runs of the job, and 60 percent for the existing average. The `logman -smooth 100` command will cause the most recent runs of the job to override the existing average.

Logman retains the statistical data of job runs in the Tivoli Workload Scheduler database. There is no limit to the number of job instances retained in the job history.

Starting production plan processing

To start a production cycle, follow these steps:

1. Log in as *TWS_user* on the master domain manager.
2. At a command prompt, run the script command **. ./***TWS_home***/tws_env.sh** in UNIX or *TWS_home***\tws_env.cmd** in Windows to set up the environment, then run the `JnextPlan` job by entering, for example on a UNIX workstation, the following command:
   ```
   JnextPlan.sh -from 05/03/06 0400 -to 06/06/06
   ```
   This creates a new production plan that starts on the third of May 2006 at 4:00 a.m. and stops on the sixth of June 2006 at 3:59 a.m. The processing day will start at the time specified on the master domain manager in the variable ***startOfDay***.
3. When the `JnextPlan` job completes, check the status of Tivoli Workload Scheduler:
   ```
   conman status
   ```
   If Tivoli Workload Scheduler started correctly, the status is
   ```
   Batchman=LIVES
   ```
   If mailman is still running a process on the remote workstation, you might see that the remote workstation does not initialize immediately. This happens because the workstation needs to complete any ongoing activities involving the mailman process before re-initializing. After the interval defined in the ***mm retry link*** parameter set in the *TWS_home*/`localopts` configuration file elapses, the domain manager tries again to initialize the workstation. As soon as the ongoing activities complete, the activities for the next day are initialized. For information about the `localopts` configuration file, refer to *IBM Tivoli Workload Scheduler Administration Guide*.
4. Increase the limit to allow jobs to run. The default job limit after installation is zero. This means no jobs will run.
   ```
   conman "limit;10"
   ```

Automating production plan processing

If you want to extend your production plan at a fixed time interval, for example every week, you have the option to automate the extension. This section explains how you can do this.

Tivoli Workload Scheduler includes a sample job stream named `final` that helps you automate plan management. A copy of this job stream is in the `Sfinal` file in the *TWS_home*/`config`/**Sfinal** directory. A copy of the job script is in the *TWS_home*/`JnextPlan` directory.

You can use either this `Sfinal` file or create and customize a new one.

Important: In any case, to be able to run this job stream succesfully, the *TWS_user* must have `Write` access to the `tmp` default temporary directory.

The `final` job stream runs the sequence of script files described in `JnextPlan` to generate the new production plan. See "Creating and extending the production plan" on page 70 for reference.

By default, the `final` job stream is set to run once a day. You can modify the time it runs by modifying two settings in the `final` job stream definition. These are the details about the two steps you need to follow to do this, for example, to make the `final` job stream run every three days:

- Modify the run cycle by setting inside the `final` job stream definition to schedule the job stream to run every three days.
- In the statement that invokes **MakePlan** inside the `final` job stream, set the production plan to last for three days by specifying **-for 72**.

Then you need to add the `final` job stream to the database by performing the following steps:

1. Log in as *TWS_user*.
2. Run the **tws_env** script to set the Tivoli Workload Scheduler environment as follows:
   - UNIX: on C shells launch `. ./`*`TWS_home`*`/tws_env.csh`
   - UNIX: on Korn shells launch `. ./`*`TWS_home`*`/tws_env.sh`
   - From a Windows command line: launch *`TWS_home`*`\tws_env.cmd`

   where *TWS_home* represents the product installation directory.
3. Run the **composer** command.
4. Add the final job stream definition to the database by running the following command:
   ```
   composer add Sfinal
   ```
   If you did not use the `Sfinal` file provided with the product but you created a new one, use its name in place of `Sfinal`.
5. Start the production cycle by running the **JnextPlan** script. In this way the `final` job stream will be included in the current production plan.

Note: Even if you decided to automate the production plan extension you can still run **JnextPlan** at any time.

Chapter 5. Using workload service assurance

Workload service assurance is an optional feature that provides the means to flag jobs as mission critical for your business and to ensure that they are processed in a timely manner. Using this function benefits your scheduling operations personnel by enhancing their ability to meet defined service levels.

When the workload service assurance feature is enabled, you can flag jobs as mission critical and ensure they have an associated completion deadline specified in their definition or at submission. Two additional threads of execution, Time Planner and Plan Monitor, that run within WebSphere Application Server, are thereafter engaged to make sure that the critical jobs are completed on time.

Defining a critical job and its deadline triggers the calculation of the start times of all the other jobs that are predecessors of the critical job. The set of predecessors of a critical job make up its *critical network*. This might include jobs from other job streams. Starting from the critical job's deadline and duration, Time Planner calculates its *critical start time*, which is the latest starting time for the job to keep up with its deadline. Moving backwards from the critical start time it calculates the latest time at which each predecessor within the critical network can start so that the critical job at the end of the chain can complete on time.

While the plan runs, Plan Monitor constantly checks the critical network to ensure that the deadline of the critical job can be met. When changes that have an impact on timings are made to the critical network, for example addition or removal of jobs or follows dependencies, Plan Monitor requests Time Planner to recalculate the critical start times. Also, when a critical network job completes, timings of jobs that follow it are recalculated to take account of the actual duration of the job.

Within a critical network, the set of predecessors that more directly risk delaying the critical start time is called *critical path*. The critical path is dynamically updated as predecessors complete or their risk of completing late changes.

The scheduler (batchman) acts automatically to remedy delays by prioritizing jobs that are actually or potentially putting the target deadline at risk, although some conditions that cause delays might require operator intervention. A series of specialized critical job views, available on the Tivoli Dynamic Workload Console, allows operators to browse critical jobs, display their predecessors and the critical paths associated with them, identify jobs that are causing problems, and drill down to identify and remedy problems.

For detailed information, see:
- "Enabling and configuring workload service assurance" on page 92
- "Planning critical jobs" on page 95
- "Processing and monitoring critical jobs" on page 97
- "Workload service assurance scenario" on page 99

For information about troubleshooting and common problems with the workload service assurance, see the Workload Service Assurance chapter in *Tivoli Workload Scheduler: Troubleshooting*.

Enabling and configuring workload service assurance

A number of global and local options control the management of critical jobs. The Tivoli Workload Scheduler security file also must authorize users with proper access to all jobs, job streams, and workstations associated with critical jobs.

Global options

The workload service assurance feature is enabled and disabled by the global option `enWorkloadServiceAssurance`. It is enabled by default. Other global and local options are used to control different aspects of the processing of critical jobs and their predecessors.

Table 10 shows the global options that are used by workload service assurance. If you want to customize the values, modify global options on the master domain manager using the **optman** command line. In most cases, the changes take effect after the next JnextPlan is run.

Table 10. Workload service assurance global options

Option	Description
`enWorkloadServiceAssurance | wa`	Enables or disables privileged processing of mission-critical jobs and their predecessors. The default value is YES. Specify NO to disable.
`promotionOffset | po`	Workload service assurance calculates a critical start time for the critical job itself and each of its predecessors. This is the latest time that the job can start without putting the timely completion of the critical job at risk. When the critical start time of a job is approaching and the job has not started, the promotion mechanism is used. A promoted job is assigned additional operating system resources and its submission is prioritized. The `promotionoffset` global option determines the length of time before the critical start time that a job becomes eligible for promotion. The default setting is 120 seconds.

Table 10. Workload service assurance global options (continued)

Option	Description
longDurationThreshold | ld	The calculation of critical start times for the jobs that form a critical network is based on the deadline defined for the critical job and the estimated durations of the critical job and each of its predecessors. If a job takes much longer than expected to complete, it could cause jobs that follow it to miss their critical start times and so put the timely completion of the critical job at risk. The `longDurationThreshold` global option is a percentage value. The default is 150. Using the default value, if the actual duration of a job is 150% of the estimated duration or longer, the job is considered to have a long duration and is added to the hot list that can be viewed on the Tivoli Dynamic Workload Console.
approachingLateOffset | al	The critical start time of a job in the critical network is the latest time that the job can start without causing the critical job to end after the deadline. In most cases, a job will start well before the critical start time so that if the job runs longer than its estimated duration, the situation does not immediately become critical. Therefore, if a job has not started and the critical start time is only a few minutes away, the timely completion of the critical job is considered to be potentially at risk. The `approachingLateOffset` option allows you to determine the length of time before the critical start time of a job in the critical network at which you are to alerted to this potential risk. If a job has still not started the specified number of seconds before the critical start time, the job is added to a hot list that can be viewed on the Dynamic Workload Console. The default is 120 seconds. **Note:** The value of this parameter is checked regularly. JnextPlan does not need to be run for changes to take effect.

Table 10. Workload service assurance global options (continued)

Option	Description
deadlineOffset | do	In general, a deadline should be specified for a job flagged as `critical`. If it is not, the scheduler uses the deadline defined for the job stream. The `deadlineOffset` option provides an offset used to calculate the critical start time in case the deadline is missing for both a critical job and its job stream. The plan end plus this offset is assumed as the critical job's deadline. The offset is expressed in minutes. The default is 2 minutes. **Important:** When the plan is extended, the start time of critical jobs whose deadline is calculated with this mechanism is automatically changed as a consequence of the fact that it must now match the new plan finishing time.

For more information about global options, see *IBM Tivoli Workload Scheduler Administration Guide*.

Local options

Workload service assurance uses local options to control the priority allocation of system resources to jobs in the critical network that must be promoted to maintain the critical deadline. Table 11 shows the local options used by the workload service assurance feature. To set local options, edit the `*twshome*\localopts` file on each workstation where critical jobs will be running. Run JnextPlan or restart the agent for changes to the local options to take effect.

Table 11. Workload service assurance local options

Option	Description
jm promoted nice	Sets the `nice` value to be assigned to critical jobs or critical job predecessors that need to be promoted on UNIX and Linux operating systems, so that they are assigned more resources and processed ahead of other jobs. Specific values vary for the different platforms, but in general, the setting must be a negative integer. The default is -1 and lower numbers represent higher priorities. If you specify a positive integer, the default value is used. The **jm nice** local option has a similar role in prioritizing jobs that have been submitted by the root user. A critical job that has been submitted by the root user could be eligible for both prioritization mechanisms. In such a case, values would be added together. For example, if **jm promoted nice** is set to -4 and **jm nice** to -2, the critical job submitted by user root would have a priority of -6.

Table 11. Workload service assurance local options (continued)

Option	Description
`jm promoted priority`	Sets the priority value for critical jobs or critical job predecessors that need to be promoted so that Windows operating systems assign them more resources and process them ahead of other jobs. The possible values are: • High • AboveNormal • Normal • BelowNormal • Low or Idle The default is AboveNormal. Note that if you set a lower priority value than the one non-critical jobs might be assigned, no warning is given and no mechanism such as the one available for jm promoted nice sets it back to the default.

Security file requirements

It is mandatory that the users who own the Tivoli Workload Scheduler instances running critical jobs are authorized to work with all jobs, job streams, and workstations associated with these jobs. These users must therefore have DISPLAY, MODIFY, and LIST rights in the security file for all the JOB, SCHEDULE and CPU associated objects.

See the Tivoli Workload Scheduler *Administration Guide* for details about the security file.

Planning critical jobs

Workload service assurance provides the means to identify critical jobs, define deadlines, and calculate timings for all jobs that must precede the critical job.

If it is critical that a job must be completed before a specific time, you can flag it as critical when you add it to a job stream using the Workload Designer functions on the Dynamic Workload Console. You can define the deadline either at job or job stream level.

Jobs can also be flagged as critical by including the **critical** keyword in the job statement when you create or modify a job stream using the **composer** command line.

When the **JnextPlan** command is run to include the new job in the production plan, all jobs that are direct or indirect predecessors of the critical job are identified. These jobs, together with the critical job itself, form a `critical network`.

Because timing of jobs in the critical network must be tightly controlled, Time Planner calculates the following timing benchmarks for each critical network job:

Critical start

It applies to distributed systems only and represents the latest time at which the job can start without causing the critical job to miss its deadline.

Critical start times are calculated starting with the deadline set for the critical job and working backwards using the estimated duration of each job to determine its critical start time. For example, if the critical job deadline is 19:00 and the estimated duration of the critical job is 30 minutes, the critical job will not finish by the deadline unless it has started by 18:30. If the immediate predecessor of the critical job has an estimated duration of 20 minutes, it must start at latest by 18.10.

Note: Only the deadline of the critical job is considered when calculating critical start times for jobs in the critical network. If other jobs have deadlines defined, their critical start times might be later than their deadlines.

Earliest start

Represents the earliest time at which a job in the critical network could start, taking into consideration all dependencies and resource requirements.

Estimated start and end times

Estimated start times are calculated starting with the earliest time at which the first job or jobs in the critical network could start and working forward using the estimated duration of each job to estimate the start time of the job that follows it.

Planned start and end times

For the initial calculations, these values are set to the estimated start and end times. They are subsequently recalculated to take into consideration any changes or delays in the plan.

Estimated duration

The estimated duration of a job is based on the statistics collected from previous runs of the job. If the job has never run before, a default value of one minute is used. Take this into account when considering the accuracy of calculated timings for the critical job networks that include jobs running for the first time. In the case of a shadow job, the estimated duration is always set to the default value of one minute. This applies to shadow jobs running for the first time, as well as any subsequent runs of the shadow job.

The timings for each job in the critical network are added to the `Symphony` file that includes all the plan information and that is distributed to all workstations on which jobs are to be run.

As the plan is run, Plan Monitor monitors all critical networks: subsequent changes to the critical network that affect the timing of jobs will trigger the recalculation of the critical and estimated start times. Changes might include manual changes, for example releasing dependencies or rerunning jobs, and changes made automatically by the system in response to a potential or actual risk to the timely completion of the critical job.

Specific views for critical jobs and their predecessors, available from the Dynamic Workload Console, allow you to keep track of the processing of the critical network. The views can immediately identify problems in your planning of the critical job. For example, if the estimated start time of a job in the critical network is later than the critical start time, this is immediately signalled as a potential risk to the critical job.

Processing and monitoring critical jobs

Workload service assurance provides automatic tracking and prioritizing of critical network jobs and online functions that you use to monitor and intervene in the processing of critical network jobs.

Automatic tracking and prioritizing

To ensure that critical deadlines can be met, workload service assurance provides the following automated services to critical jobs and the predecessor jobs that form their critical networks:

Promotion

When the critical start time of a job is approaching and the job has not started, the promotion mechanism is used. A promoted job is assigned additional operating system resources and its submission is prioritized.

The timing of promotions is controlled by the global option `promotionoffset`. Promoted jobs are selected for submission after jobs that have priorities of "high" and "go", but before all other jobs. Prioritizing of operating system resources is controlled by the local options `jm promoted nice` (UNIX and Linux) and `jm promoted priority` (Windows).

Calculation of the critical path

The critical path is the chain of dependencies, leading to the critical job, that is most at risk of causing the deadline to be missed at any given time. The critical path is calculated using the estimated end times of the critical job predecessors. Working back from the critical job, the path is constructed by selecting the predecessor with the latest estimated end time. If the actual end time differs substantially from the estimated end time, the critical path is automatically recalculated.

Figure 21 on page 98 shows the critical path through a critical network at a specific moment during the processing of the plan.

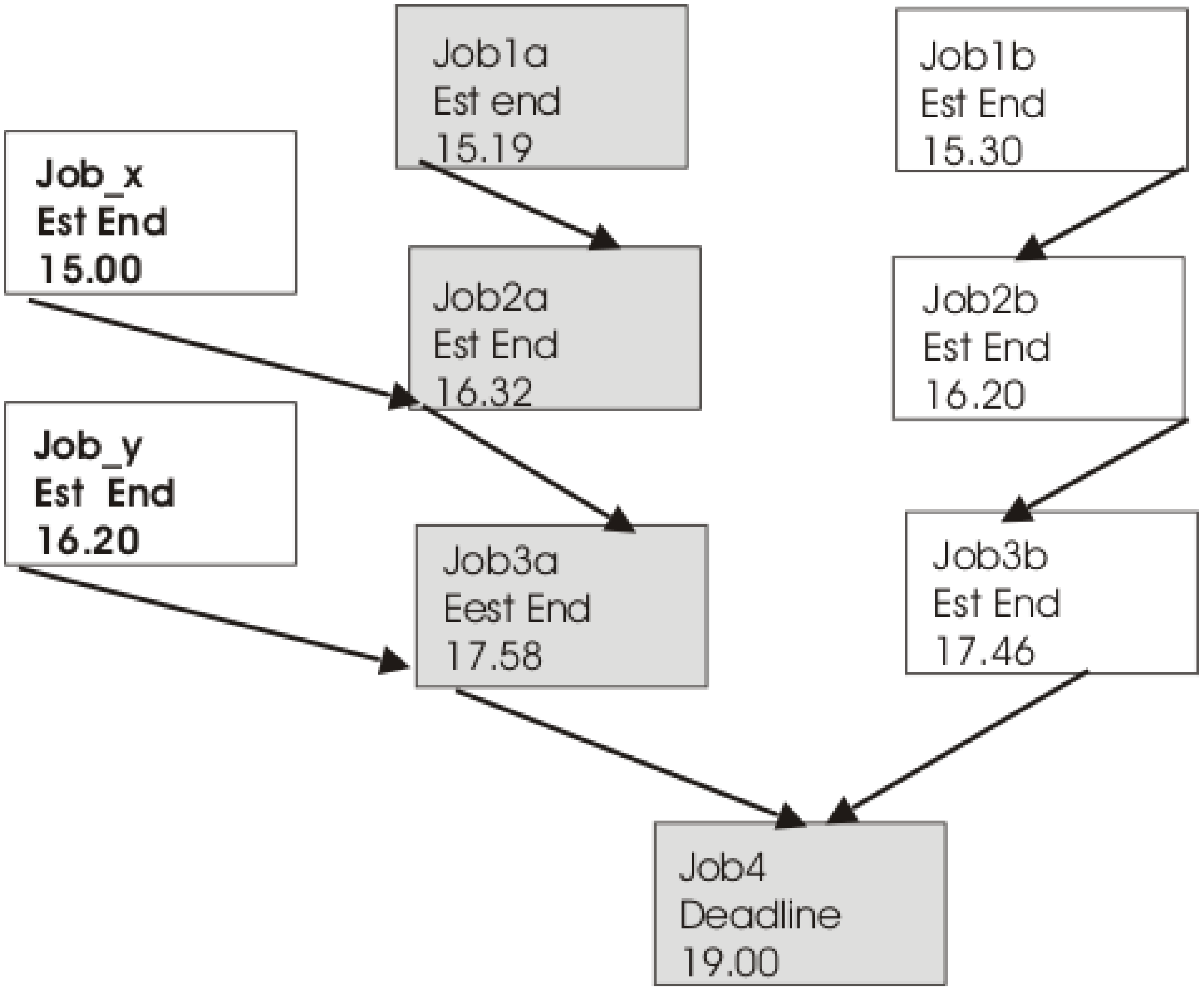


Figure 21. Critical path

At this time, the critical path includes Job3a, Job2a, and Job1a. Job3a and Job3b are the immediate predecessors of the critical job, Job4, and Job3a has the later estimated end date. Job3a has two immediate predecessors, Job2a and Job_y. Job2a has the later estimated end time, and so on.

Addition of jobs to the hot list

Jobs that are part of the critical network are added to a hot list that is associated to the critical job itself. The hot list includes any critical network jobs that have a real or potential impact on the timely completion of the critical job. Jobs are added to the hot list for the one or more of the reasons listed next. Note that only the jobs beginning the current critical network, for which there is no predecessor, can be included in the hot list.

- The job has stopped with an error. The length of time before the critical start time is determined by the `approachingLateOffset` global option.
- The job has been running longer than estimated by a factor defined in the `longDurationThreshold` global option.
- The job has still not started, though all its follows dependencies have either been resolved or released, and at least one of the following conditions is true:
  - The critical start time has nearly been reached.

- The job is scheduled to run on a workstation where the limit is set to zero.
- The job belongs to a job stream for which the limit is set to zero.
- The job or its job stream has been suppressed.
- The job or its job stream currently has a priority that is lower than the fence or is set to zero.

Setting a high or potential risk status for the critical job
A risk status can be set for the critical job, as follows:

High risk
Calculated timings show that the critical job will finish after its deadline.

Potential risk
Critical predecessor jobs have been added to the hot list.

Online tracking of critical jobs

The Tivoli Dynamic Workload Console provides specialized views for tracking the progress of critical jobs and their predecessors. You can access the views from the Dashboard or by creating Monitor Critical Jobs tasks.

The initial view lists all critical jobs for the engine, showing the status: normal, potential risk, or high risk. From this view, you can navigate to see:
- The hot list of jobs that put the critical deadline at risk.
- The critical path.
- Details of all critical predecessors.
- Details of completed critical predecessors.
- Job logs of jobs that have already run.

Using the views, you can monitor the progress of the critical network, find out about current and potential problems, release dependencies, and rerun jobs.

Workload service assurance scenario

This scenario illustrates the use of workload service assurance in ensuring that important production deadlines can be met.

Fine Cola uses Tivoli Workload Scheduler to manage the timing and interdependencies of its production and supply process.

Fine Cola has service level agreements with many customers that support "just-in-time" restocking. This means that late starts on any of the delivery routes will almost certainly result in Fine Cola's products not being on the shelves.

The job that produces the loading orders for trucks must be completed at latest by 5.30 a.m. This job is dependent on the successful completion of other jobs. For example, although orders are processed ahead of time, last-minute changes often arrive when trucks return after completing a delivery route. Fine Cola also provides invoices with delivery notes, so changes to orders must also be reflected in the pricing and might trigger special-offer adjustments to prices.

Planning the critical job

Using the Workload Designer on the Dynamic Workload Console, the Fine Cola scheduler flags the loading order job as critical and sets the deadline for 5 a.m.

When **JnextPlan** is run, the critical start dates for this job and all the jobs that are identified as predecessors of the critical job are calculated.

Tracking the critical job

1. The Tivoli Workload Scheduler operator checks the dashboards and sees that there are critical jobs scheduled on one of the engines.
2. He sees that there is a critical job in potential risk. He clicks the potential risk link to get the list of critical jobs in this state.

   The loading orders job shows a status of "potential risk".
3. He selects the job and clicks **Hot List** to see the job or jobs that are putting the critical job at risk.

   The orders adjustment job is listed as being in error.
4. He selects the job and clicks **Job log**.

   The log shows that the job failed because of incorrect credentials for the orders database.
5. After discovering that the database password was changed that day, he changes the job definition in the symphony file and reruns the job.
6. When he returns to the dashboard, he sees that there are no longer any jobs in potential risk. Also, the critical jobs list that was opened when clicking on the potential risk link no longer shows the critical job after the job is rerun.
7. The job is now running and has been automatically promoted to give it higher priority for submission and system resources.
8. No further problems need fixing and the critical job finally completes at 4.45 a.m.

Chapter 6. Customizing your workload using variable tables

This chapter introduces the concept of variable tables to group global parameters, from now on called variables, to customize your workload.

Starting from Tivoli Workload Scheduler version 8.5, what were called global parameters in previous versions, are now called variables. Variable definitions are contained in variable tables. A variable table is an object that groups together multiple variables. Using variable tables you can assign different values to the same variable for use in job and job stream definitions in JCL, log on, prompts dependencies, file dependencies, and recovery prompts. This is particularly useful when a job definition is used as a template for a job that belongs to more than one job stream. For example, you can assign different values to the same variable and reuse the same job definition in different job streams saving therefore time and money.

When you define a variable, you assign it to a variable table because the same variable can be defined in different variable tables with different values. Or a better approach is to create one or more variable tables, specifying a list of variable names and values for each table. While doing this, you can add the same variable name with different values in different tables. When you request a list of variables you get *variabletable.variablename* pairs to easily identify to which variable table the variable belongs.

For example, the VAR1 variable defined in the REP1_TABLE1 variable table is shown as:

```
REP1_TABLE1.VAR1
```

You can assign variable tables to run cycles, job streams, and workstations.

Using variable tables you change the behavior of the workload according to when, why, and where you want to run your schedule giving you more flexibility in customizing your workload and meet your service level agreements. In detail:

When To change the behavior of jobs and job streams based on when they are scheduled to run, that is, on which days they run. Using variable tables with run cycles.

Why To change the behavior of jobs and job streams based on why they are scheduled to run, for example to create a job that runs different commands. Using variable tables with job streams.

Where To change the behavior of jobs and job streams based on where they run, for example on different workstations. Using variable tables with workstations.

Migrating global parameters from previous versions

Considerations when migrating global parameters from previous versions

When you upgrade from versions earlier than 8.5, the global parameter definitions, now called variable definitions, that you have in the database, are automatically migrated to the default variable table called MAIN_TABLE. After the upgrade:

- All variables are preceded by the default table name. For example, after you migrate, the *REP_PATH* variable acquires the following name:

  ```
  MAIN_TABLE.REP_PATH
  ```

  When you request a list of variables you get *variabletable.variablename* pairs to easily identify to which variable table the variable belongs.
- Your workload is resolved in the same way as before the migration because any Tivoli Workload SchedulerTivoli Workload Scheduler object containing variables refers to the MAIN_TABLE for variable resolution.
- For every user section that includes the `parameter` keyword, the following row is added in the security file:

  ```
  vartable name=@ access=add,delete,display,modify,list,use,unlock
  ```

For details about the upgrade process, see Tivoli Workload Scheduler*Tivoli Workload Scheduler Planning and Installation*.

When upgrading from version 8.3 or later, do not modify variables until you migrate the master domain manager and all its backup masters because in this transition phase you have two different versions of the database. If you must add or modify variables during this transition phase, make sure you make the change in both version 8.3 or 8.4 and version 8.5 of the master domain managers.

Local parameters that were created and managed with the **parms** utility command in the local parameter database on the workstations, work in the same way as before.

In V8.3 and V8.4 the parameters are stored in the MDL.VAR_VARIABLES table of the database. After you upgrade to V8.5 or later, the same parameters are stored in the MAIN_TABLE contained in the MDL.VAR_VARIABLES2 database table. If you started to migrate your environment, but did not yet complete the migration, and the master domain manager is V8.3 or V8.4 and the backup master domain manager is V8.5 or later, or viceversa, the V8.3 or V8.4 master domain manager does not recognize the table MDL.VAR_VARIABLES2 so if you change a parameter in the V8.3 or V8.4 master domain manager this change is stored in the old table (MDL.VAR_VARIABLES). If you change the parameter in the V8.5 master domain manager the change is stored in the MDL.VAR_VARIABLES2 table.

The default variable table

This topic describes the default variable table and how it works.

The default variable table is the table that contains all the variables that you defined without specifying any variable table name. The default name of the default variable table is **MAIN_TABLE**. You can modify this name at any time or set another variable table as the default variable table. You cannot delete the default variable table. When you set another variable table as the default, the original default variable table is no longer marked as default. You can work with the default variable table in the same way as any other variable table. You can easily identify the default variable table on the user interface because it is marked with a **Y** in the **default** field.

Example

This example shows a list of variable tables

```
Variable Table Name                     Default  Updated On  Locked By
---------------------------------------- -------  ----------  ----------------
MAIN_TABLE                                 Y      05/07/2008  -
VT_1                                              05/07/2008  -
VT_2                                              05/07/2008  -
VARTABLE3                                         05/07/2008  -
V4                                                05/07/2008  -
VT5                                               05/07/2008  -

AWSBIA291I Total objects: 6
```

Data integrity for variable tables

This topic explains how data integrity is guaranteed using variable tables.

In the same way as for other objects, Tivoli Workload Scheduler maintains variable table data integrity whenever you run commands that create, modify, rename, or delete the definition of a variable table.

When referencing a variable table from any run cycle, job stream, or workstation Tivoli Workload Scheduler checks that the variable table exists and preserves the link between it and the run cycle, job stream, and workstation. This means that you cannot delete a variable table definition while a reference from a run cycle, a job stream, or a workstation still exists.

Referential integrity is guaranteed at variable table level and not at variable level, that is, when a run cycle, a job stream, and a workstation references a variable table, Tivoli Workload Scheduler checks that the variable table exists, not that the referenced variable exists.

Locking mechanism for variable tables

This topic describes how the locking mechanism works for variable tables.

Locking is set at variable table level to ensure that definitions in the database are not overwritten by different users concurrently accessing the same variable table. This means that both when you lock a variable table and when you lock a variable you gain exclusive permissions for all the variables in that variable table. That is, you can perform any commands on the locked variable table and on all the variables in it. Any other user only has read-only access to this variable table and to the variables in it.

This prevents any other user from changing the same variable table that you are changing. If another user tries to lock a variable table or a variable that you have already locked, an error message is returned.

Variable table security

This topic describes how to define security settings for variable tables.

User access to variable tables must be authorized in the Tivoli Workload Scheduler security file. As for other objects, the connector verifies the existence of proper authorization before executing an action that requires access to a variable table. The following new keyword is available in the security file for this purpose:

```
vartable name=@ access=add,delete,display,modify,list,use,unlock
```

You need `use` access to be able to reference a variable table from other objects (job streams, run cycles and workstations). Security filters are based on the `name` attribute only, but your Tivoli Workload Scheduler administrator has the option to use the **$default** keyword to specify security permissions on the default table, regardless of its name.

Permission to work on a variable is no longer based on the individual variable but on the table enclosing it. Access to a variable is granted only if the corresponding action on the enclosing variable table is permitted. The following table shows the corresponding permissions for variables and variable tables:

Table 12. The relationship between variable tables and their enclosed variables in the Tivoli Workload Scheduler security file

Defined access to vartable	Allowed action on enclosed variables
modify	add
	delete
	modify
display	display
unlock	unlock

Starting with version 8.5, the `parameter` keyword in the security file applies to local parameters only.

See the *Tivoli Workload Scheduler Administration Guide* for details about the security file.

Variable resolution

This topic describes how variables are resolved both when you generate a plan and when you submit a job or a job stream.

When you generate a plan, Tivoli Workload Scheduler analyzes the variable tables in the order shown below for variable resolution:
1. In the run cycle.
2. In the job stream.
3. In the workstation. See “Workstation considered for variable resolution” on page 105.
4. In the default variable table.

At plan resolution time each level is analyzed in the order described above. If you specify a variable that is not contained in any variable table, including the default, a warning message containing the name of the unresolved variable is written in the `*TWS_home*\eWAS\profiles\TIPProfile\logs\twaserver\SystemOut.log` log file and the variable name is left in the plan.

When you submit a job stream, Tivoli Workload Scheduler resolves variables by analyzing the variable tables in the order shown below:
1. Specified during the submit operation.
2. In the job stream.
3. In the workstation. See “Workstation considered for variable resolution” on page 105.
4. In the default variable table.

When you submit a job, Tivoli Workload Scheduler resolves variables by analyzing the variable tables in the order shown below:
1. Specified during the submit operation.
2. In the workstation. See "Workstation considered for variable resolution."
3. In the default variable table.

Workstation considered for variable resolution

When the variable is resolved by the variable table specified in the workstation, the workstation taken into consideration is:

For variable in file dependency
The workstation where the file resides.

For variable in job
The workstation where the job is defined.

For variable in prompt dependency

Global prompt
No workstation is taken into consideration. Variables in global prompts are resolved always using the default variable table. This is because global prompt are used by all jobs and job streams so just one value must be used for variable resolution.

Ad hoc prompt
The workstation where the job or the job stream that depends on the prompt dependency is defined.

Chapter 7. Running event-driven workload automation

Event-driven workload automation adds the capability to perform on-demand workload automation in addition to plan-based job scheduling. It provides the capability to define rules that can trigger on-demand workload automation.

The object of event-driven workload automation in Tivoli Workload Scheduler is to carry out a predefined set of actions in response to events that occur on nodes running Tivoli Workload Scheduler (but also on non-Tivoli Workload Scheduler ones, when you use the `sendevt` command line). This implies the capability to submit workload and run commands on the fly, notify users via email, or send messages to Tivoli Enterprise Console.

The main tasks of event-driven workload automation are:
- Trigger the execution of batch jobs and job streams based on the reception or combination of real time events.
- Reply to prompts
- Notify users when anomalous conditions occur in the Tivoli Workload Scheduler scheduling environment or batch scheduling activity.
- Invoke an external product when a particular event condition occurs.

Event-driven workload automation is based upon the concept of event rule. In Tivoli Workload Scheduler an event rule is a scheduling object that includes the following items:
- Events
- Event-correlating conditions
- Actions

When you define an event rule, you specify one or more events, a correlation rule, and the one or more actions that are triggered by those events. Moreover, you can specify validity dates, a daily time interval of activity, and a common time zone for all the time restrictions that are set.

The events that Tivoli Workload Scheduler can detect for action triggering can be:

Internal events
: They are events involving Tivoli Workload Scheduler internal application status and changes in the status of Tivoli Workload Scheduler objects. Events of this category can be job or job stream status changes, critical jobs or job streams being late or canceled, and workstation status changes.

External events
: They are events not directly involving Tivoli Workload Scheduler that may nonetheless impact workload submission. Events of this category can be messages written in log files, events sent by third party applications, or a file being created, updated, or deleted.

Within a rule two or more events can be correlated through correlation attributes such as a common workstation or job. The correlation attributes provide a way to direct the rule to create a separate rule (or copy of itself) for each group of events that share common characteristics. Typically, each active rule has one copy that is running in the event processing server. However, sometimes the same rule is needed for different groups of events, which are often related to different groups of resources. Using one or more correlation attributes is a method for directing a rule to create a separate rule copy for each group of events with common characteristics.

The actions that Tivoli Workload Scheduler can run when it detects any of these events can be:

Operational actions
They are actions that cause the change in the status of scheduling objects. Actions of this category are submitting a job, job stream, or command, or replying to a prompt.

Notification actions
They are actions that have no impact on the status of scheduling objects. Actions belonging to this category are sending an email, logging the event in an internal auditing database, forwarding the event to Tivoli Enterprise Console, or running a non-Tivoli Workload Scheduler command.

This classification of events and actions is conceptual. It has no impact on how they are handled by the event-driven mechanism.

Simple event rule scenarios

This section lists some simple scenarios involving the use of event rules. The corresponding XML coding is shown in “Event rule examples” on page 116.

Scenario 1: Send email notification

1. The administrator defines the following event rule:
   - When any of the `job123` jobs terminates in error and yields the following error message:
     ```
     AWSBHT001E The job "MYWORKSTATION#JOBS.JOB1234" in file "ls" has
     failed with the error: AWSBDW009E The following operating system
     error occurred retrieving the password structure for either the
     logon user...
     ```
     send an email to operator `john.smith@mycorp.com`. The subject of the email includes the names of the job instance and of the associated workstation.

     The event rule is valid from December 1st to December 31st in the 12:00-16:00 EST time window.
2. The administrator saves the rule as non-draft in the database and it is readily deployed by Tivoli Workload Scheduler.
3. The scheduler starts monitoring the jobs and every time one of them ends in error, John Smith is sent an email so that he can check the job and take corrective action.

Scenario 2: Monitor that workstation links back

1. The administrator defines the following event rule:
   - If workstation `CPU1` becomes unlinked and does not link back within 10 minutes, send a notification email to `chuck.derry@mycorp.com`.
2. The administrator saves the rule as non-draft in the database and it is readily deployed by Tivoli Workload Scheduler.
3. The scheduler starts monitoring `CPU1`.

If the workstation status becomes unlinked, Tivoli Workload Scheduler starts the 10 minute timeout. If the `CPU1 linked` event is not received within 10 minutes, the scheduler sends the notification email to Chuck Derry.

4. Chuck Derry receives the email, queries the actions/rules that were triggered in the last 10 minutes, and from there navigates to the `CPU1` instance and runs a first problem analysis.

Scenario 3: Submit job stream when FTP has completed

1. The administrator defines the following event rule:
   - When file `daytransac*` is created in the `SFoperation` directory in workstation `system1`, and modifications to the file have terminated, submit the `calmonthlyrev` job stream.

     The event rule is valid year-round in the 18:00-22:00 EST time window.
2. The administrator saves the rule as non-draft in the database and it is readily deployed by Tivoli Workload Scheduler.
3. The scheduler starts monitoring the `SFoperation` directory. As soon as file `daytransac*` is created and is no longer in use, it submits job stream `calmonthlyrev`.
4. The operator can check the logs to find if the event rule or the job stream were run.

Scenario 4: Start long duration jobs based on timeout

1. The administrator defines the following event rule:
   - When the `job-x=exec` event and the `job-x=succ/abend` event are received in 5 minutes, the scheduler should reply `Yes` to `prompt-1` and start the `jobstream-z` job stream, otherwise it should send an email to `twsoper@mycompany.com` alerting that the job is late.
2. The administrator saves the event rule in draft status. After a few days he edits the rule, changes the email recipient and saves it as non-draft. The rule is deployed.
3. Every time the status of `job-x` becomes `exec`, Tivoli Workload Scheduler starts the 5 minutes timeout.

   If the internal state of `job-x` does not change to `succ` or `abend` within 5 minutes and the corresponding event is not received, Tivoli Workload Scheduler sends the email, otherwise it replies `Yes` to the prompt and submits `jobstream-z`.

Scenario 5: Monitoring process status and running a batch script

The administrator creates a rule to monitor the status of Tivoli Workload Scheduler processes and run a batch script.

Scenario 6: Integration with SAP R/3 (in combination with Tivoli Workload Scheduler for Applications)

1. The administrator defines the following event rule:
   - When an event called `ID3965` is generated on SAP R/3 server `Billing`, Tivoli Workload Scheduler must:
     a. Run the command:
        ```
        "/usr/apps/helpDesk –openTicket –text
         'Processing error $parameter
          on SAP system $wsname'"
        ```

        to open a service desk ticket

   b. Send an event to Tivoli Enterprise Console.
2. The administrator saves the rule as non-draft and it is readily deployed.
3. Tivoli Workload Scheduler starts monitoring the status of SAP R/3 events activated on the `Billing` system.

   When the `ID3965` event is detected, Tivoli Workload Scheduler runs the specified help desk command and sends an event to TEC.
4. After some time, the administrator sets the event rule in draft status. The rule is automatically deactivated. It can be deployed again when needed.

Scenario 7: Monitoring the Symphony file status and logging the occurrence of a corrupt record
The administrator creates a rule to monitor the status of the Symphony file in the Tivoli Workload Scheduler instance and logs the occurrence of a corrupt Symphony dependency record in the internal auditing database.

The event rule management process

Event-driven workload automation is an ongoing process and can be reduced to the following steps:

1. An event rule definition is created or modified with the Tivoli Dynamic Workload Console or with the composer command line and saved in the objects database. Rule definitions can be saved as draft or non-draft.
2. All new and modified non-draft rules saved in the database are periodically (by default every five minutes) found, built, and deployed by an internal process named `rule builder`. At this time they become active. Meanwhile, an `event processing server`, which is normally located in the master domain manager, receives all events from the agents and processes them.
3. The updated monitoring configurations are downloaded to the Tivoli Workload Scheduler agents and activated. Each Tivoli Workload Scheduler agent runs a component named `monman` that manages two services named `monitoring engine` and `ssmagent` that are to catch the events occurring on the agent and perform a preliminary filtering action on them.
4. Each `monman` detects and sends its events to the event processing server.
5. The event processing server receives the events and checks if they match any deployed event rule.
6. If an event rule is matched, the event processing server calls an actions helper to carry out the actions.
7. The action helper creates an event rule instance and logs the outcome of the action in the database.
8. The administrator or the operator reviews the status of event rule instances and actions in the database and logs.

The event-driven workload automation feature is automatically installed with the product. You can at any time change the value of the `enEventDrivenWorkloadAutomation` global option if you do not want to use it in your Tivoli Workload Scheduler network.

Event-driven workload automation is based on a number of services, subsystems, and internal mechanisms. The following ones are significant because they can be managed:

monman

Is installed on every Tivoli Workload Scheduler agent where it checks for all local events. All detected events are forwarded to the event processing server. The following `conman` commands are available to manage `monman`:

Table 13. conman commands for managing monitoring engines

Command	Purpose
deployconf	Updates the monitoring configuration file for the event monitoring engine on an agent. It is an optional command since the configuration is normally deployed automatically.
showcpus getmon	Returns the list of event rules defined for the monitor running on an agent. This command can be used remotely to get the information of the configuration file in another agent of the network
startmon	Starts `monman` on an agent. Can be issued from a different agent.
stopmon	Stops `monman` on an agent. Can be issued from a different agent.

`monman` starts up automatically each time a new Symphony is activated. This is determined by the `autostart monman` local option that is set to `yes` by default (and that you can disable if you do not want to monitor events on a particular agent).

Following each rule deployment cycle, updated monitoring configurations are automatically distributed to the agents hosting rules that have been changed since the last deployment. Note that there might be some transitory situations while deployment is under course. For example, if a rule is pending deactivation, the agents might be sending events in the time fraction that the new configuration files have not been deployed yet, but the event processor already discards them.

If an agent is unable to send events to the `event processing server` for a specified period of time, the monitoring status of the agent is automatically turned off. The period of time can be customized (in seconds) with the `edwa connection timeout` parameter in the `localopts` file. By default, it is set to 300 seconds (5 minutes).

The following events can be configured in the `BMEvents.conf` file to post the monitoring status of an agent:

- TWS_Stop_Monitoring (261) : sent when the monitoring status of an agent is set to off (for stopmon command or because the agent is unable to send events to the `event processing server`).
- TWS_Start_Monitoring (262): sent when the monitoring status of an agent is set to on (for startmon command or because the agent has restarted to send events to the `event processing server`).

These events have the following positional fields:

1. Event number
2. Affected workstation
3. Reserved, currently always set to 1

Event processing server

Can be installed on the master domain manager, the backup master, or on any fault-tolerant agent installed as a backup master. It runs in the embedded application server. It can be active on only one node in the network. It builds the rules, creates configuration files for the agents, and notifies the agents to download the new configurations. It receives and correlates the events sent by the monitoring engines and runs the actions. The following `conman` commands are available to manage the event processing server:

Table 14. conman commands for managing the event processing server

Command	Purpose
starteventprocessor	Starts the event processing server
stopeventprocessor	Stops the event processing server
switcheventprocessor	Switches the event processing server from the master domain manager to the backup master or fault-tolerant agent installed as a backup master, or vice versa

The event processing server starts up automatically with the master domain manager. Only one event processor may run in the network at any time. If you want to run the event processor installed on a workstation other than the master (that is, on the backup master or on any fault-tolerant agent installed as backup master), you must first use the switcheventprocessor command to make it the active event processing server.

Note: If you set the **`ignore`** keyword on the workstation definition of the agent (installed as backup master) that at the time hosts the active event processor, the first following **`JnextPlan`** occurrence acknowledges that this particular agent is out of the plan. As a consequence, it cannot restart the event processor hosted there. For this reason, the scheduler yields a warning message and starts the event processor hosted by the master domain manager.

Using the involved interfaces and commands

Running and managing event-driven workload automation calls for the following tasks:
- Edit configuration settings
- Model event rules
- Manually deploy or undeploy event rules
- Manage monitoring and event processing devices
- Monitor and manage event rule instances

You must be ready to utilize several Tivoli Workload Scheduler interfaces and commands to do them. Table 15 on page 113 summarizes the ones you need:

Table 15. Interfaces and commands for managing event-driven workload automation

Interface or command	Use to...
optman	Change the default values of global options associated with event management. Global options are used to configure: • The frequency with which rule definitions are checked for updates (`deploymentFrequency`). Modified definitions are deployed in the Tivoli Workload Scheduler domain • The EIF port number where the event processing server receives events (`eventProcessorEIFPort`, or `eventProcessorEIFSSLPort` when SSL-protected). • Management of the cleanup policies of rule instance, action run, and message log data (`logCleanupFrequency`). • SMTP server properties if you deploy rules implementing actions that send emails via an SMTP server (`smtpServerName`, `smtpServerPort`, `smtpUseAuthentication`, `smtpUserName`, `smtpUserPassword`, `smtpUseSSL`, `smtpUseTLS`). • Tivoli Enterprise Console server properties if you deploy rules implementing actions that forward events to TEC (`TECServerName`, `TECServerPort`). • The possibility to disable the event rule management mechanism (`enEventDrivenWorkloadAutomation`) which is installed by default with the product. See the *Administration Guide* for a list of global options.
composer	Run modeling and management tasks of event rule definitions like add, create, delete, display, extract, list, lock, modify, new, print, unlock, validate. Event rules are defined in XML. Query the Tivoli Workload Scheduler relational database for: • event rule definitions filtered by: – rule, event, and action properties – jobs and job streams involved with the rule action • event rule instances, actions run, and message log records See "Event rule definition" on page 222 to learn how to define event rules. See Chapter 9, "Managing objects in the database - composer," on page 235 for command reference.

Table 15. Interfaces and commands for managing event-driven workload automation (continued)

Interface or command	Use to...
Dynamic Workload Console	Have a graphical user interface to: • Model and manage event rule definitions (browse, create, delete, modify, query, unlock) • Query the Tivoli Workload Scheduler relational database for: – event rule definitions filtered by: - rule, event, and action properties - jobs and job streams involved with the rule action – event rule instances, actions run, and message log records • Manage the event processing server and monitoring engines, as described in tables Table 13 on page 111 and Table 14 on page 112 See the Dynamic Workload Console online documentation.
conman	Manage the monitoring devices, namely the event processing server and monitoring engines, as described in tables Table 13 on page 111 and Table 14 on page 112. See Chapter 10, “Managing objects in the plan - conman,” on page 291 for command reference.
utility commands	Create custom event definitions and manually send custom events to the event processing server. See “evtdef” on page 445 and “sendevent” on page 461 for details on these commands.
planman	Manually deploy new and changed rules. See “Deploying rules” on page 80 for details.
Security file	Set security authorizations to manage event rules, events, actions, and their instances. See the *Tivoli Workload Scheduler Administration Guide* for reference about configuring the Tivoli Workload Scheduler security file.

Important: If you use a security firewall, make sure that the ports defined in global option `eventProcessorEIFPort` and in the `nm port` local option on each agent are open for incoming and outgoing connections.

Defining event rules

When you define an event rule, you specify one or more events, a correlation rule, and one or more actions. To define event rules you can use:
- The `composer` command line
- The Dynamic Workload Console
- A set of APIs described in a separate document

In `composer`, you edit the rules with an XML editor of your choice (preferable but not mandatory) since event rule definitions are made using XML syntax.

The explanation of how you use `composer` to define event rules is in "Event rule definition" on page 222, while the explanation of how you use the Dynamic Workload Console can be found in its online help.

Event rule definitions are saved in the Tivoli Workload Scheduler database like all other scheduling objects. You can save them as:

Draft The rule is saved in the database but is not ready yet to be deployed and activated.

This state is determined by the `isDraft=yes` attribute.

Not draft
This rule is being deployed or is ready to be deployed in the scheduling environment.

This state is determined by the `isDraft=no` attribute.

Non-draft rules are ready to be activated. The rule builder calculates the status of each rule. The status can be:
- active
- not active
- update pending
- update error
- activation pending
- activation error
- deactivation pending
- deactivation error

The scheduler periodically (every five minutes or in accordance with a time set in the `deploymentFrequency` global configuration option) scans the database for non-draft rules and builds rule configuration files for deployment. The new monitoring configurations are downloaded to the agents (each agent gets its own configuration file containing strictly the rules it is to run) only if there have been changes since the previous configuration files.

As an additional feature, a `planman deploy` command is available to deploy rules manually at any time.

| The time required for deployment increases proportionally with the number of
| active rules to deploy. If you need to manually deploy a large number of new or
| changed rules, and you are concerned with keeping the deployment time low, run
| `planman deploy -scratch` to reduce the deployment time.

You can deploy and undeploy rules as you need by setting the `isDraft` attribute to `no` or to `yes` with `composer` or with the Dynamic Workload Console.

Based on their characteristics, rules are classified as:

filter The rule is activated upon the detection of a single specific event.

sequence
The rule is activated when an ordered group of events is detected or fails to complete within a specific time interval.

set The rule is activated when an unordered group of events is detected or fails to complete within a specific time interval.

Filter rules are based on the detection of one event such as a job being late, a Tivoli Workload Scheduler workstation going down, a file being modified, a job stream completing its run successfully, and so on.

Set and sequence rules are based on the detection of more events. Optionally, they can be based on a time out condition. A rule times out when the first event(s) in a sequence or part of the events in a set are received, but not all the events are received within a specified time interval counted from when the first event was received.

Rule definitions may include attributes that specify a validity period and an activity time window within each day of validity. If you do not specify these attributes, the rule is active perpetually at all times once it is deployed and until it is changed back to draft status or deleted from the database.

You can use variable substitution. This means that when you define action parameters, you can use attributes of occurrences of events that trigger the event rule in either of these two forms:

- `${event.property}`

  Replaces the value as is. This is useful to pass the information to an action that works programmatically with that information, for example the `schedTime` of a job stream.
- `%{event.property}`

  Replaces the value formatted in human readable format. This is useful to pass the information to an action that forwards that information to a user, for example to format the `schedTime` of a job stream in the body of an email.

Where:

event Is the name you set for the triggering `eventCondition`.

property
Is the `attributeFilter` name in the filtering predicate of the triggering event condition. The value taken by the attribute filter when the rule is triggered is replaced as a parameter value in the action definition before it is performed.

Note that you can use variable substitution also if no `attributeFilter` was specified for an attribute and also if the attribute is read-only.

You can see the use of variable substitution in some of the following sample definitions, where attribute filter values are replaced in email subject and body matters.

Event rule examples

The following are examples of event rule definitions that apply to the scenarios described in "Simple event rule scenarios" on page 108.

Event rule definition for scenario #1

When any of the `job123` jobs terminates in error and yields the following error message:

```
AWSBHT001E The job "MYWORKSTATION#JOBS.JOB1234" in file "ls" has failed with
the error: AWSBDW009E The following operating system error occurred retrieving
the password structure for either the logon user...
```

send an email to operator `john.smith@mycorp.com`. The subject of the email includes the names of the job instance and of the associated workstation.

The event rule is valid from December 1st to December 31st in the 12:00-16:00 EST time window.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="scenario1_rule" ruleType="filter" isDraft="no">
      <description>This is the definition for scenario1</description>
      <timeZone>America/Indianapolis</timeZone>
      <validity from="2010-12-01" to="2010-12-31" />
      <activeTime start="12:00:00" end="16:00:00" />
      <eventCondition name="event1"
                      eventProvider="TWSObjectsMonitor"
                      eventType="JobStatusChanged">
         <filteringPredicate>
            <attributeFilter name="JobStreamWorkstation" operator="eq">
               <value>*</value>
            </attributeFilter>
            <attributeFilter name="JobStreamName" operator="eq">
               <value>*</value>
            </attributeFilter>
            <attributeFilter name="JobName" operator="eq">
               <value>job123*</value>
            </attributeFilter>
            <attributeFilter name="Status" operator="eq">
               <value>Error</value>
            </attributeFilter>
            <attributeFilter name="ErrorMessage" operator="eq">
               <value>*AWSBDW009E*</value>
            </attributeFilter>
         </filteringPredicate>
      </eventCondition>
      <action actionProvider="MailSender"
              actionType="SendMail"
              responseType="onDetection">
         <description>Send email to John Smith including names of job
                      and associated workstation</description>
         <parameter name="To">
            <value>john.smith@mycorp.com</value>
         </parameter>
         <parameter name="Subject">
            <value>Job %{event1.JobName} on agent %{event1.Workstation}
                   ended in error</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

Important: The error message that explains why a job terminates in error can be found in the TWSMERGE log file. In this scenario, the TWSMERGE log file contains the following statement:

```
BATCHMAN:+
BATCHMAN:+ AWSBHT001E The job "MYWORKSTATION#JOBS.JOB1234" in file "ls"
has failed with the error: AWSBDW009E The following operating system
error occurred retrieving the password structure for either the logon
user, or the user who owns a file or external dependency
BATCHMAN:+
```

where the error message is everything that follows the string:

```
has failed with the error:
```

Event rule definition for scenario #2

If workstation CPU1 becomes unlinked and does not link back within 1 hour, send a notification email to chuck.derry@mycorp.com.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="scenario2_rule" ruleType="filter" isDraft="no">
      <description>This is the definition for scenario2</description>
      <timeZone>America/Anchorage</timeZone>
      <timeInterval amount="1" unit="hours" />
      <eventCondition name="WSevent"
                      eventProvider="TWSObjectsMonitor"
                      eventType="ChildWorkstationLinkChanged">
         <filteringPredicate>
            <attributeFilter name="Workstation" operator="eq">
               <value>CPU1</value>
            </attributeFilter>
            <attributeFilter name="LinkStatus" operator="eq">
               <value>Unlinked</value>
            </attributeFilter>
         </filteringPredicate>
      </eventCondition>
      <action actionProvider="MailSender"
              actionType="SendMail"
              responseType="onDetection">
         <description>Send email to Chuck Derry with name of
                      unlinked workstation</description>
         <parameter name="To">
             <value>chuck.derry@mycorp.com</value>
         </parameter>
         <parameter name="Subject">
             <value>Agent CPU1 has been unlinked for at least 10 minutes</value>
         </parameter>
         <parameter name="Body">
             <value>The cause seems to be: %{WSevent.UnlinkReason}</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

Event rule definition for scenario #3

When file daytransac is created in the SFoperation directory in workstation system1, and modifications to the file have terminated, submit the calmonthlyrev job stream.

The event rule is valid year-round in the 18:00-22:00 EST time window.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="scenario3_rule"
              ruleType="filter"
              isDraft="no">
      <description>This is the definition for scenario3</description>
      <timeZone>America/Louisville</timeZone>
      <validity from="2007-01-01" to="2007-12-31" />
      <activeTime start="18:00:00" end="22:00:00" />
      <eventCondition eventProvider="FileMonitor"
                      eventType="ModificationCompleted">
```

```
            <filteringPredicate>
               <attributeFilter name="FileName" operator="eq">
                  <value>daytransac</value>
               </attributeFilter>
               <attributeFilter name="Workstation" operator="eq">
                  <value>EVIAN1</value>
               </attributeFilter>
            </filteringPredicate>
         </eventCondition>
         <action actionProvider="TWSAction"
                 actionType="sbs"
                 responseType="onDetection">
            <description>Submit the calmonthlyrev job stream.</description>
            <parameter name="JobStreamName">
               <value>calmonthlyrev</value>
            </parameter>
            <parameter name="JobStreamWorkstationName">
               <value>act5cpu</value>
            </parameter>
         </action>
      </eventRule>
   </eventRuleSet>
```

Event rule definition for scenario #4

When the `job-x=exec` event and the `job-x=succ/abend` event are received in 500 seconds, the scheduler should reply `Yes` to `prompt-1` and start the `jobstream-z` job stream, otherwise it should send an email to `twsoper@mycompany.com` alerting that the job is late.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="scenario4_rule"
              ruleType="sequence"
              isDraft="yes">
      <description>This is the definition for scenario4</description>
      <timeZone>America/Buenos_Aires</timeZone>
      <timeInterval amount="500" unit="seconds" />
      <eventCondition eventProvider="TWSObjectsMonitor"
                      eventType="JobStatusChanged">
         <filteringPredicate>
            <attributeFilter name="JobName" operator="eq">
               <value>job-x</value>
            </attributeFilter>
            <attributeFilter name="InternalStatus" operator="eq">
               <value>EXEC</value>
            </attributeFilter>
         </filteringPredicate>
      </eventCondition>
      <eventCondition eventProvider="TWSObjectsMonitor"
                      eventType="JobStatusChanged">
         <filteringPredicate>
            <attributeFilter name="JobName" operator="eq">
                <value>job-x</value>
            </attributeFilter>
            <attributeFilter name="InternalStatus" operator="eq">
                <value>ABEND</value>
                <value>SUCC</value>
            </attributeFilter>
         </filteringPredicate>
      </eventCondition>
      <action actionProvider="MailSender"
              actionType="SendMail"
              responseType="onTimeOut">
         <description>Send email to operator saying that job-x
                      is late</description>
         <parameter name="To">
            <value>twsoper@mycorp.com</value>
         </parameter>
         <parameter name="Subject">
            <value>Job-x is late by at least 5 minutes</value>
         </parameter>
      </action>
      <action actionProvider="TWSAction"
              actionType="Reply"
              responseType="onDetection">
         <description>Reply Yes to prompt-1</description>
         <parameter name="PromptName">
            <value>prompt-1</value>
         </parameter>
         <parameter name="PromptAnswer">
            <value>Yes</value>
         </parameter>
      </action>
      <action actionProvider="TWSAction"
              actionType="sbs"
              responseType="onDetection">
         <description>Submit jobstream-z</description>
         <parameter name="JobStreamName">
            <value>jobstream-z</value>
         </parameter>
         <parameter name="JobStreamWorkstationName">
            <value>act23cpu</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

Event rule definition for scenario #5

Monitor the status of Tivoli Workload Scheduler processes listed in ProcessName and run the RUNCMDFM.BAT batch script located in E:\production\eventRules.

The TWSPATH keyword indicates the fully qualified path where the monitored Tivoli Workload Scheduler instance is installed, including the /TWS suffix.

On Windows operating systems, the event rule is triggered every time the agent is stopped using the ShutDownLwa command and every time the agent is stopped manually. On UNIX operating systems, the event rule is triggered when you stop the process manually, while it is not triggered by the ShutDownLwa command.

If you specify ProcessName=agent, the agent component is monitored, while the TWS JobManager process is not monitored.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                        event-management/rules/EventRules.xsd">
    <eventRule name="scenario5rule" ruleType="filter" isDraft="no">
        <eventCondition name="twsProcMonEvt1"
        eventProvider="TWSApplicationMonitor"
        eventType="TWSProcessMonitor">
            <scope>
                AGENT, NETMAN DOWN ON WIN86MAS
            </scope>
            <filteringPredicate>
                <attributeFilter name="ProcessName" operator="eq">
                 <value>agent</value>
                 <value>appservman</value>
                 <value>batchman</value>
                 <value>jobman</value>
                 <value>mailman</value>
                 <value>monman</value>
                 <value>netman</value>
       </attributeFilter>
                <attributeFilter name="TWSPath" operator="eq">
                    <value>E:\Program Files\IBM\TWA\TWS</value>
                </attributeFilter>
                <attributeFilter name="Workstation" operator="eq">
                    <value>win86mas</value>
                </attributeFilter>
                <attributeFilter name="SampleInterval" operator="eq">
                    <value>5</value>
                </attributeFilter>

            </filteringPredicate>
        </eventCondition>
        <action actionProvider="GenericActionPlugin" actionType="RunCommand"
    responseType="onDetection">
            <scope>
                RUNCMDFM.BAT
            </scope>
            <parameter name="Command">
                <value>runCmdFM.bat</value>
            </parameter>
            <parameter name="WorkingDir">
                <value>E:\production\eventRules</value>
            </parameter>
        </action>
    </eventRule>
</eventRuleSet>
```

Event rule definition for scenario #7

Monitor the Symphony file status in the *IT041866-9088* workstation and logs the occurrence of a corrupt Symphony record in the internal auditing log database.

In each workstation, the Batchman process monitors the Symphony file. When it detects a corrupt record, it send the corruption event to the Monman process message queue and then to Event Processor in the Master workstation.

The event rule is triggered every time the Batchman process finds a corrupt dependency record in the workstation specified in the event rule definition.

If you set the workstation value to IT041866-9088, the Symphony file on this workstation is monitored, and the event rule is triggered when the Batchman process on this workstation detects a corrupt record in the Symphony file.

The occurrence of the corrupt record is written to the Messagge logger audit file.

```
?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules
http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules/EventRules.xsd">
<eventRule name="TEST1" ruleType="filter" isDraft="no">
<eventCondition name="productAlertEvt1" eventProvider="TWSObjectsMonitor"
eventType="ProductAlert">
<scope>
IT041866-9088
</scope>
```

```
<filteringPredicate>
<attributeFilter name="Workstation" operator="eq">
<value>IT041866-9088</value>
</attributeFilter>

</filteringPredicate>
</eventCondition>
<action actionProvider="MessageLogger" actionType="MSGLOG"
responseType="onDetection">
<scope>
OBJECT=%{PRODUCTALERTEVT1.WORKSTATION} MESSAGE=%{PRODUCTALERTEVT1.WORKSTATION}
corruption
</scope>
<parameter name="ObjectKey">
<value>%{productAlertEvt1.Workstation}</value>
</parameter>
<parameter name="Message">
<value>%{productAlertEvt1.Workstation} corruption</value>
</parameter>
<parameter name="Severity">
<value>Info</value>
</parameter>
</action>
</eventRule>
</eventRuleSet>
```

Rule operation notes

The following contains critical information on event rule behavior that might help you understand the reason for unexpected results:

Notes on rule status

- Depending on its `from` and `to` validity dates, the status of any rule changes as follows upon deployment:
  - If you create a rule with already expired `from` and `to` validity dates, the rule is saved in `activation pending` state. When the rule is deployed, it remains in `activation pending` status.
  - If you set the `to` validity field to a future date, the rule is deployed in the `active` state. If you reset it to a past date, the rule is redeployed in the `no active` state.
- Rule activity times (`start` and `end`) do not affect rule status. As long as a rule is within the right validity dates, the rule remains in the `active` state regardless whether it is within its defined activity times. If the scheduler receives a rule 's defined events outside its activity time, the events are discarded but the rule stays in the `active` status.

Lack of persistence in rule instance status

If the event processor is stopped or crashes, the status of rule instances that are only partially matched is lost.

Repeating `set` and `sequence` rules

Typically, each active rule has one, and only one, copy that runs in the event processing server.`Set` and `sequence` rules use the mechanism explained in the following example:

1. You define a `sequence` rule with two events, A and B.
2. When the first event that matches the sequence occurs (event A), it activates the rule and waits for the second event (event B).

   Once the rule is active, any additional event A events that may arrive are ignored. No additional rules are created for any newly detected event A events as long as the rule is waiting for event B.

3. Once event B occurs, the rule is completed and resets, waiting for event A to occur again.

The mechanism of `set` and `sequence` rules is such that any additional occurrences of an already detected event are ignored and discarded if the other defined events have not arrived.

You can prevent this problem by using correlation attributes. Using one or more correlation attributes is a method for directing a rule to create a separate rule copy when another event A arrives.

`Set` and `sequence` rule types with shorter than 24 hours activity time windows

Occurrences of `set` or `sequence` rules that were defined to be active for only a few hours of every day, are not purged when the activity period within each day expires if only part of the events arrive. They remain in an idle state, waiting to receive the remaining events in the following days.

For example, you define a `set` rule that includes two events. The rule is valid from January 1 to January 10, and is active daily from 6 to 10 AM.

If on January 1 the first event is received at 8 AM, the rule waits for the second event to take place, and stays *alive* beyond 10 AM if the event is not detected.

If the second event is received at 11 AM (which is out of the activity time window), it is discarded, but the rule keeps on staying *alive*. If the second event is received again at 7 AM of January 2, the rule is triggered and the actions are implemented.

If you don not want the rule to *carry over* to the next day, you should purge it.

Triggered rule elements

Every time the event conditions listed in a deployed event rule are matched, or time out, an event rule instance is created. An event rule instance includes information like event rule name, date and time when it was matched, and the list of action instances, and their status, that were run as a result of the matching event conditions. The `RuleInstance` object is used to collect information about triggered rules in a history log of rule instances.

Actions carried out by a triggered rule are collected in a history log of action runs. The provided information includes action runs that have been completed with errors or warnings, as opposed to successful ones. If at least one action ends in error, then the whole rule instance will be reported in error. As part of the information tracked in action runs, rule fields are also maintained, and queries can be executed to look for action runs based on these fields (the rule instance identifier is also available).

Defining custom events

In addition to the already defined event types and event classes (known as providers) listed in detail in “Event providers and definitions” on page 575, Tivoli Workload Scheduler supplies the template of a generic event provider named `GenericEventPlugIn` that programmers with specific application and XML programming skills can modify to define custom event types that might be of use to the organization.

The tools supplied to define custom event types are:
- The `GenericEventPlugIn` event provider in XML
- The evtdef utility command with which a programmer can download the `GenericEventPlugIn` event provider as a local file to define the custom events
- The XML schema definition (XSD) files necessary to validate the modified generic event provider. They also contain online guidelines to aid in the programming task.
- The sendevent utility command with which the custom events can be sent to the event processing server to trigger rules from any agent or any workstation running simply the Tivoli Workload Scheduler remote command line client.

This is the flow for defining and using custom events:
1. With the evtdef command, the programmer:
   a. Downloads the generic event provider as a local file.
   b. Follows the schema definitions to add custom event types and to define their properties and attributes in the file with an XML editor.
   c. Uploads the local file as the modified generic event provider containing the new custom event type definitions. The modified generic event provider is saved in an XML file on the master domain manager.
2. The rule builder, or the administrator, defines, with either composer or the Dynamic Workload Console, the event rules that are to be triggered by these custom events, specifying:
   - The generic event provider as the event provider
   - The custom event types as the event types
   - The custom event type properties (or attributes) defined for the custom events in the generic event provider with the particular values that will trigger the rules.
3. Deploy the rules.
4. When the occurrence of a custom event takes place, it can be sent to the event processing server in one of the following ways:
   - By the sendevent command, run from a script or from the command line
   - By another application, such as Tivoli Enterprise Console or Tivoli Monitoring

   As soon as the event is received by the event processing server, it triggers the rule.

Chapter 8. Defining objects in the database

Tivoli Workload Scheduler is based on a set of object definitions that are used to map your scheduling environment into the database. Scheduling objects are:

- calendars
- domains
- event rules
- jobs
- job streams
- prompts
- parameters
- resources
- variable tables
- workstations
- workstation classes
- users

This chapter contains the following section:

- "Defining scheduling objects"

Defining scheduling objects

Scheduling objects are managed with the **composer** command-line program and are stored in the Tivoli Workload Scheduler database. The **composer** command-line program can be installed and used on any machine connected through TCP/IP to the system where the master domain manager is installed. It does not require the installation of a Tivoli Workload Scheduler workstation as a prerequisite. It communicates through HTTP/HTTPS with the master domain manager where the DB2 database is installed. The HTTP/HTTPS communication setup and the authentication check are managed by the WebSphere Application Server infrastructure. The **composer** program uses edit files to update the scheduling database. The format of the edit file used to define a specific object is described later in this chapter. For example, to create a new object, enter its definition in an edit file, and then use **composer** to add it to the database by specifying the edit file containing the definition. The **composer** command-line program checks for correct syntax inside the edit file, and, if correct, transforms the object definition into XML language and then sends the request through HTTP/HTTPS to the master domain manager.

On the master domain manager the XML definition is parsed, semantic and integrity checks are performed, and then the update is stored in the database.

With this version of the product all entries are managed individually. Another feature introduced with this new version is the object locking mechanism. Scheduling objects defined in the database are locked while accessed by a user to prevent concurrent accesses. This means that only the user locking the object has `write` permission to that object, and other users have `read only` access to that object. For additional information refer to "lock" on page 270 and "unlock" on page 284.

You can use short and long keywords when issuing commands from **composer**, as Table 16 on page 126 shows. The first two columns on the left list the long and short keyword formats currently supported by Tivoli Workload Scheduler. The rightmost column lists the backward compatible formats you want to use if your network includes pre-version 8.3 installations.

Table 16. List of supported scheduling object keywords

Long keywords	Short keywords	Backwards compatible keywords with pre-version 8.3 installations
calendar	cal	calendars
domain	dom	cpu
eventrule	erule | er	—
jobdefinition	jd	jobs
jobstream	js	sched
parameter	parm	parms
prompt	prom	prompts
resource	res	resources
user	user	users
variabletable	vt	—
workstation	ws	cpu
workstationclass	wscl	cpu

Note: The **cpu** keyword is maintained to represent domains, workstations, and workstation classes for backward compatibility.

The **composer** program does not issue specific warnings if scheduling language keywords are used as names of scheduling objects. However, the use of such keywords can result in errors, so avoid using the keywords listed in Table 17 when defining jobs and job streams:

Table 17. List of reserved words when defining jobs and job streams

abendprompt	after	as	at	autodocoff
autodocon	canc	carryforward	confirmed	continue
dateval	day(s)	day_of_week	deadline	description
docommand	draft	end	every	everyday
except	extraneous	fdignore	fdnext	fdprev
filename	follows	freedays	from	go
hi	i18n_id	i18n_priority	interactive	isdefault
isuserjob	jobfilename	jobs	keyjob	keysched
limit	matching	members	needs	nextjob
notempty	number	on	onuntil	op
opens	order	previous	priority	prompt
qualifier	rccondsucc	recovery	relative	request
rerun	runcycle	sa	sameday	schedtime
schedule	scriptname	stop	streamlogon	su
tasktype	timezone	to	token_in	until
validfrom	validto	vartable	vt	weekday(s)
workday(s)				

Avoid using the keywords listed in Table 18 on page 127 when defining workstations, workstation classes, and domains:

Table 18. List of reserved words when defining workstations

access	AIX	agent_type	autolink	behindfirewall
command	cpuclass	cpuname	description	domain
enabled	end	extraneous	for	force
fta	fullstatus	host	hpux	ibm i
ignore	isdefault	linkto	maestro	manager
master	members	mpeix	mpev	mpexl
mpix	node	number	off	on
os	other	parent	posix	server
secureaddr	securitylevel	tcpaddr	timezone	type
tz	tzid	UNIX	using	vartable
wnt				

Avoid using the keywords listed in Table 19 when defining users:

Table 19. List of reserved words when defining users

username	password	end

Using object definition templates

Scheduling object definition templates are available for your use in the *TWS_home*\templates directory. You can use the templates as a starting point when you define scheduling objects.

Note that the dates in the templates are in the format expressed in the `date format` local option.

Workstation definition

In a Tivoli Workload Scheduler network, a workstation is a scheduling object that runs jobs. You create and manage the scheduling object workstation in the Tivoli Workload Scheduler database by means of a workstation definition. A workstation definition is required for every object that runs jobs. Typically, a workstation definition is used to represent a physical workstation but, in the case of extended agents for example, it represents a logical definition that must be hosted by a physical workstation.

You can include multiple workstation definitions in the same text file, together with workstation class definitions and domain definitions. A workstation definition has the following syntax:

Syntax

cpuname *workstation* [**description** "*description*"]
[**vartable** *table_name*]
os *os-type*
[**node** *hostname*] [**tcpaddr** *port*]
[**secureaddr** *port*] [**timezone|tz** *tzname*]
[**domain** *domainname*]
[**for maestro** [**host** *host-workstation* [**access** *method* | **agentID** *agentID*]]
    [**type fta** | **s-agent** | **x-agent** | **manager** | **broker** | **agent** | **rem-eng** | **pool** | **d-pool**]
    [**ignore**]
    [**autolink on** | **off**]

[**behindfirewall on** | **off**]
[**securitylevel enabled** | **on** | **force**]
[**fullstatus on** | **off**]
[**server** *serverid*]]
[**protocol** *http* | *htpps*]
[**members** [*workstation*] [...]]
[**requirements** *jsdl_definition*]]
end

[**cpuname** ...]

[**cpuclass** ...]

[**domain** ...]

Arguments

You can include multiple workstation definitions in the same text file, together with workstation class definitions and domain definitions. A workstation definition has the following syntax:

Table 20. Attribute settings for management workstation types

Attributes	Master domain manager	Domain manager	Backup domain manager
cpuname	The name of the workstation.		
description	A description for the workstation enclosed within double quotes. This attribute is optional.		
vartable	The name of a variable table associated with the workstation. Variables used with the workstation are defined in this table. This attribute is optional.		

os	The operating system installed on the system. Specify one of the following values:
	
	`UNIX`
	`WNT`
	`OTHER`
	`IBM_i`		

node	The system host name or IP address.		
tcpaddr	The value assigned to *nm port* in the `localopts` file. For multiple workstations on a system, enter an unused port number. The default value is 31111.		
secureaddr	The value assigned to *nm ssl port* in the `localopts` file. Specify it if **securitylevel** is set to `on`, `force` or `enabled`.		
timezone | **tz**	The time zone in which the system is located. It is recommended that the value matches the value set on the operating system.		
domain	`MASTERDM`	The name of the managed domain.	
host	Not applicable		
access	Not applicable		
type	`manager`		`fta`
ignore	Use this attribute if you do not want this workstation to appear in the next production plan.		

Table 20. Attribute settings for management workstation types (continued)

Attributes	Master domain manager	Domain manager	Backup domain manager

autolink	It indicates if a link between workstations is automatically opened at startup. Specify one of the following values:
	
	`ON`
	`OFF`
	
	This is an optional attribute. The default value is `ON`.		

behindfirewall	This setting is ignored.	It indicates if there is a firewall between the workstation and the master domain manager. Specify one of the following values:
		
		`ON`
		`OFF`
		
		The default value is `OFF`.	

securitylevel	The type of SSL authentication to use:
	
	`enabled`
	`on`
	`force`		

fullstatus	`ON`		
server	Not applicable		This setting is ignored.
protocol	Not applicable		
members	Not applicable		
requirements	Not applicable		

Table 21 describes the values you set for each attribute for target workstation types. Following the table you find additional details about each attribute.

Table 21. Attribute settings for target workstation types

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool
cpuname	The name of the workstation.						
description	A description for the workstation enclosed within double quotes. This attribute is optional.						
vartable	The name of a variable table associated with the workstation. Variables used with the workstation are defined in this table. This attribute is optional.						

os	The operating system installed on the system. Specify one of the following values:
	
	`UNIX`
	`WNT`
	`OTHER`
	`IBM_i`
	
	Specify `OTHER` for IBM isystems running as limited fault-tolerant agents.
		`OTHER`	The operating system installed on the machine. Specify one of the following values:
			
			`UNIX`
			`WNT`
			`OTHER`
			`IBM_i`
				This value setting is discovered on the system.	The operating system installed on the machine. Specify one of the following values:
					
					`UNIX`
					`WNT`
					`ZOS`
						The operating system installed on the machine. Specify one of the following values:
						
						`UNIX`
						`WNT`
						`OTHER`
						`IBM_i`	

Table 21. Attribute settings for target workstation types (continued)

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool
node	The system host name or IP address.		The system host name or IP address. Specify `NULL` when **host** is set to `$MASTER`, or when defining an extended agent for PeopleSoft, SAP or Oracle.	Agent host name or IP address.	Remote engine host name or IP address.	Not applicable	
tcpaddr	The value assigned to *nm port* in the `localopts` file. When defining multiple workstations on a system, enter an unused port number. The default value is 31111.	The value assigned to *nm port* in the `localopts` file. When defining multiple workstations on a system, enter an unused port number. The default value is 41114.	See the selected access method specifications.	The port number to communicate with the agent when the protocol is `http`.	The port number to communicate with the remote engine when the protocol is `http`.	Not applicable	
secureaddr	The value assigned to *nm ssl port* in the `localopts` file. Specify it if **securitylevel** is set to `on`, `force` or `enabled`.	Not Applicable	Not Applicable	The port number to communicate with the agent when the protocol is `https`.	The port number to communicate with the remote engine when the protocol is `https`.	Not applicable	
timezone | tz	The time zone in which the system is located. It is recommended that the value matches the value set on the operating system.		The time zone set on the workstation specified in the **host** attribute.	The time zone set on the agent.	The time zone set on the remote engine.	The time zone set on the pool agents.	The time zone set on the dynamic pool agents.
domain	Specify an existing domain. The default value for fault-tolerant agents is `MASTERDM`. This setting is mandatory for standard agents.	Specify an existing domain. This setting is mandatory.	This setting is needed only if the value assigned to **host** is `$MANAGER`.	Not applicable			
host	Not Applicable		The host workstation. It can be set to `$MASTER` or `$MANAGER`.	The broker workstation.			
access	Not Applicable			Select the appropriate access method file name.	Not Applicable		
agentID				The unique identifier of the dynamic agent			

Table 21. Attribute settings for target workstation types (continued)

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool

type	`fta`
	`s-agent`
	
	The default value is `fta`.
	
	Specify `fta` for IBM isystems running as limited fault-tolerant agents.
		`broker`	`x-agent`	`agent`	`rem-eng`	`pool`	`d-pool`

ignore	Use this attribute if you do not want this workstation to appear in the next production plan.						

autolink	It indicates if a link between workstations is automatically opened at startup. Specify one of the following values:
	
	`ON`
	`OFF`
	
	This is an optional attribute. The default value is `ON`.
			`OFF`	Not applicable			

behindfirewall	It indicates if there is a firewall between the workstation and the master domain manager. Specify one of the following values:
	
	`ON`
	`OFF`
	
	The default value is `OFF`.
			`OFF`	Not applicable			

securitylevel	The type of SSL authentication to use:
	
	`enabled`
	`on`
	`force`
	
	Not applicable for IBM isystems running as limited fault-tolerant agents.
		Not Applicable					

fullstatus	It indicates if the workstation is updated about job processing status in its domain and subdomains. Specify one of the following values:
	
	`ON`
	`OFF`
	
	Specify `OFF` for standard agents.
		`OFF`		Not applicable			

Table 21. Attribute settings for target workstation types (continued)

Attribute	Fault-tolerant agent and standard agent	Workload broker workstation	Extended agent	Agent	Remote engine workstation	Pool	Dynamic pool
server	0-9, A-Z. When specified, it requires the creation of a dedicated mailman processes on the parent workstation.		Not Applicable				

protocol	Not applicable			Specify one of the following values:
				`http`
				`https`
				This attribute is optional. When not specified, it is automatically determined from the settings specified for **tcpaddr** and **secureaddr**.
						Not applicable	

members	Not applicable					Required value	Not applicable
requirements	Not applicable						Required value

Following you find additional details workstation definition attributes:

cpuname *workstation*
: Specifies the name of the workstation. Workstation names must be unique and cannot be the same as workstation class names.

  The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

  Do not use in this field any of the reserved words specified in Table 17 on page 126

description "*description*"
: Provides a description of the workstation. The description can contain up to 120 alphanumeric characters. The text must be enclosed within double quotes.

vartable *table_name*
: Specifies the name of the variable table you want to associate with the workstation. Variables used with the workstation are defined in this table.

  The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 80 characters.

os *os_type*
: Specifies the operating system of the workstation. When used in remote engine workstation definitions it represents the operating system of the Tivoli Workload Scheduler remote engine.

  Valid values are:

  UNIX For supported operating systems running on UNIX-based systems, including LINUX systems.

  WNT For supported Windows operating systems.

  OTHER
  Mandatory value for: dynamic workload broker workstations, and

IBM i systems running as limited fault-tolerant agents. Optional value for other types of workstations.

ZOS Used with remote engine workstations that are defined to communicate with Tivoli Workload Scheduler for z/OS remote engine.

IBM_i For supported IBM i operating systems.

Note: See the *IBM Tivoli Workload Scheduler System Requirements document* for an up-to-date list of supported operating systems.

node *hostname*

Specify the host name or the TCP/IP address of the workstation. Fully-qualified domain names are accepted.

For host names, valid characters are alphanumeric, including dash (-). The maximum length is 51 characters.

Specify `NULL` when:

- defining an extended agent for PeopleSoft, SAP or Oracle.
- **host** is set to `$MASTER`

If you are defining a remote engine workstation, specify the host name of the system where the remote engine is installed.

tcpaddr *port*

Specifies the **netman** TCP/IP port number that Tivoli Workload Scheduler uses for communicating between workstations.

For workload broker workstations

Specify the value of the **TWS.Agent.Port** property of the `TWSAgentConfig.properties` file.

For remote engine workstations using HTTP protocol to communicate with the remote engine

Specify the HTTP port number of the remote engine.

For other types of workstations

Specify the value assigned in the `localopts` file for variable *nm port*.

The default value for this field is `31111`. Specify a value in the range from `1` to `65535`.

secureaddr

Defines the port used to listen for incoming SSL connections. This value is read when the **securitylevel** attribute is set.

For workload broker workstations

Ignore this attribute.

For remote engine workstations using HTTPS protocol to communicate with the remote engine

Specify the HTTPS port number of the remote engine.

For other types of workstations

Specify the value assigned in the `localopts` file for variable *nm ssl port*. The value must be different value from the value assigned to *nm port* variable in the `localopts` file.

If **securitylevel** is specified, but this attribute is missing, the default value for this field is `31113`. Specify a value in the range from `1` to `65535`.See the

IBM Tivoli Workload Scheduler Administration Guide for information about SSL authentication and local options set in the `TWS_home/localopts` configuration file.

timezone|tz *tzname*
Specifies the time zone of the workstation. To ensure the accuracy of scheduling times, this time zone must be the same as the computer operating system time zone.

When used in remote engine workstation definitions it represents the time zone set on the Tivoli Workload Scheduler remote engine.

See Chapter 16, "Managing time zones," on page 531 for valid time zone names.

domain *domainname*
Specifies the name of Tivoli Workload Scheduler domain the workstation belongs to. The default value for fault-tolerant workstation is `MASTERDM`.

Tivoli Workload Scheduler ignores domain setting when defined for extended agents, except when the **host** attribute is set to `$MASTER`.

This setting is mandatory for standard agent and dynamic workload broker workstations.

host *host-workstation*
This attribute is mandatory for extended agents and remote engine workstations and specifies:

For remote engine workstations, agents, pools and dynamic pools:
The broker workstation hosting the workstation. This field cannot be updated after the remote engine workstation creation.

For extended agents
The workstation with which the extended agent communicates and where its access method is installed. The hosting workstation cannot be another extended agent.

If the hosting workstation is a domain manager for which you have defined a backup, you can specify one of the following values to ensure the extended agent is not isolated if the hosting workstation becomes unavailable:

$MASTER
To indicate that the host workstation for the extended agent is the master domain manager.

$MANAGER
To indicate that the host workstation for the extended agent is the domain manager. In this case you must specify the domain where the agent is located.

In this case make sure that the extended agent methods are installed also on the backup workstation. You can enable and disable the automatic resolution of the `$MASTER` key using the *mm resolve master* option in the `localopts` file.

For more information about the options available in the `localopts` file, see *IBM Tivoli Workload Scheduler Administration Guide*.

access *method*

Specifies an access method for extended and network agents. It corresponds to the name of a file that is located in the `TWS_home/methods` directory on the hosting workstation.

Specify `NULL` when defining an extended agent for PeopleSoft, SAP, or Oracle.

agentID *agentID*

The unique identifier of the dynamic agent.

type Specifies the type of the workstation.

Note: If you plan to change the workstation types, consider the following rules:

- you can change fault-tolerant agent, standard agent, extended agent, domain manager and dynamic workload broker workstations to any workstation type, with the exception of dynamic agent, pool, dynamic pool, and remote engine.
- you cannot change the type of dynamic agent, pool, dynamic pool, and remote engine.

Enter one of the following values:

fta If you define a *fault-tolerant agent*, that is an agent workstation that launches jobs and resolves local dependencies without a domain manager. This is the default value for this attribute.

You must specify `fta` if you want to assign the workstation the role of backup domain manager or backup master domain manager.

Specify `fta` for IBM isystems running as limited fault-tolerant agents.

s-agent

If you define a *standard agent*, that is an agent workstation that launches jobs only under the direction of its domain manager.

x-agent

If you define an *extended agent*, that is an agent workstation that launches jobs only under the direction of its hosting workstation. Extended agents can be used to interface Tivoli Workload Scheduler with non-Tivoli systems and applications.

For more information, see *IBM Tivoli Workload Scheduler for Applications*.

manager

If you define a *domain manager*, that is a workstation that manages a domain. When defining this type of workstation, specify:

Server NULL

Domain

The name of the domain the workstation manages, if different from `MASTERDM` domain.

You specify that a workstation is a manager also in the *manager* field of the “Domain definition” on page 144. Tivoli Workload Scheduler automatically checks that the values specified in these fields are consistent.

broker

If you define a *dynamic workload broker* workstation, that is a workstation that runs both existing job types and job types with advanced options. It is the broker server installed with the master domain manager and the dynamic domain manager. It hosts the following workstations:

- extended agent
- remote engine
- pool
- dynamic pool
- agent. This definition includes the following agents:
  - dynamic agent
  - Tivoli Workload Scheduler for z/OS agent
  - agent for z/OS

  For more information about the dynamic agent and Tivoli Workload Scheduler for z/OS agent, see *Scheduling Workload Dynamically*. For more information about the agent for z/OS, see *Scheduling with the agent for z/OS*.

agent If you define a *dynamic agent*, that is a workstation that manages a wide variety of job types, for example, specific database or file transfer jobs, in addition to traditional job types. It is hosted by the workload broker workstation. The workstation definition is automatically created and registered when you install the dynamic agent component. In its definition you can edit only the following attribute:

- **description**
- **vartable**

rem-eng

If you define a *remote engine workstation*, that is a workstation used to communicate with a remote engine when binding a locally defined job, named *shadow job*, to a specific job running on the remote engine, named *remote job*. When the two jobs are bound, the shadow job status transition maps the remote job status transition. This mapping is useful also to define and monitor dependencies of local jobs on jobs running on the remote engine; such dependencies are called *cross dependencies*.

For more information on shadow jobs and cross dependencies, see Chapter 19, “Defining and managing cross dependencies,” on page 559.

When defining this type of workstation, specify:

os The operating system of the remote engine.

host The name of the hosting broker workstation.

node The hostname or the IP address of the remote engine.

When specifying the port number to use to communicate with the remote engine, use **secureaddr** if the protocol used is HTTPS, **tcpaddr** if the protocol used is HTTP. It is recommended that you specify in the **timezone** field the time zone set on the remote engine.

pool If you define a *pool*, that is a set of dynamic agents with similar hardware or software characteristics to submit jobs to. This workstation is hosted by the workload broker workstation. In its definition you can edit only the following attributes:

- **description**
- **vartable**
- **members**

d-pool If you define a *dynamic pool*, that is a set of dynamic agents which is dynamically defined based on the requirements listed in the JSDL file specified in the **resources** attribute. This workstation is hosted by the workload broker workstation. In its definition you can edit only the following attributes:

- **description**
- **vartable**
- **requirements**

ignore Specifies that the workstation definition must not be added to the production plan. If you specify this setting the jobs and job streams scheduled to run on this workstations are not added to the production plan.

autolink

Specifies whether to open the link between workstations at startup. Depending on the type of the workstation, when you set its value to `on`:

On a fault-tolerant agent or on a standard agents

It means that the domain manager open the link to the agent when the domain manager is started.

On a domain manager

It means that its agents open links to the domain manager when they are started.

This setting is particularly useful when a new production plan is created on the master domain manager: As part of the production plan generation all workstations are stopped and then restarted. For each agent with **autolink** turned on, the domain manager automatically sends a copy of the new production plan and then starts the agent. If **autolink** is turned on also for the domain manager, the agent opens a link back to the domain manager.

If the value of **autolink** is `off` for an agent, you can open the link from its domain manager by running the **conman link** command on the agent's domain manager or the master domain manager.

behindfirewall

If set to `on`, it means there is a firewall between the workstation and the master domain manager. In this case only a direct connection between the workstation and its domain manager is allowed and the **start** *workstation*, **stop** *workstation*, and **showjobs** commands are sent following the domain hierarchy, instead of making the master domain manager or the domain manager open a direct connection with the workstation.

Set this attribute to `off` if you are defining a workstation with type `broker`.

fullstatus

Specify this setting when defining a fault-tolerant agent workstation. For domain managers this keyword is automatically set to `on`. Specify:

on If you want the fault-tolerant agent workstation to operate in *full status* mode, meaning that the workstation is updated with the status of jobs and job streams running on all other workstations in its domain and subordinate domains, but not on peer or parent domains. The copy of the production plan on the agent is kept at the same level of detail as the copy of the production plan on its domain manager.

off If you want the fault-tolerant agent workstation to be informed about the status of jobs and job streams only on other workstations that affect its own jobs and job streams. Specifying "on" can improve performance by reducing network activity.

securitylevel

Specifies the type of SSL authentication for the workstation. Do not specify this attribute for a workstation with type `broker`. It can have one of the following values:

enabled
The workstation uses SSL authentication only if its domain manager workstation or another fault-tolerant agent below it in the domain hierarchy requires it.

on The workstation uses SSL authentication when it connects with its domain manager. The domain manager uses SSL authentication when it connects to its parent domain manager. The fault-tolerant agent refuses any incoming connection from its domain manager if it is not an SSL connection.

force The workstation uses SSL authentication for all of its connections and accepts connections from both parent and subordinate domain managers. The workstation refuses any incoming connection that is not an SSL connection.

If this attribute is omitted, the workstation is not configured for SSL connections and any value for **secureaddr** is ignored. Make sure, in this case, that the *nm ssl port* local option is set to 0 to ensure that **netman** process does not try to open the port specified in **secureaddr**. See the *IBM Tivoli Workload Scheduler Administration Guide* for information about SSL authentication.

The following table describes the type of communication used for each type of **securitylevel** setting.

Table 22. Type of communication depending on the security level value

Value set on the Fault-tolerant Agent (or the Domain Manager)	Value set on its Domain Manager (or on its Parent Domain Manager)	Type of connection established
Not specified	Not specified	TCP/IP
Enabled	Not specified	TCP/IP
On	Not specified	No connection
Force	Not specified	No connection
Not specified	On	TCP/IP
Enabled	On	TCP/IP
On	On	SSL
Force	On	SSL
Not specified	Enabled	TCP/IP

Table 22. Type of communication depending on the security level value (continued)

Value set on the Fault-tolerant Agent (or the Domain Manager)	Value set on its Domain Manager (or on its Parent Domain Manager)	Type of connection established
Enabled	Enabled	TCP/IP
On	Enabled	SSL
Force	Enabled	SSL
Not specified	Force	No connection
Enabled	Force	SSL
On	Force	SSL
Force	Force	SSL

The values for **securitylevel** is not specified for dynamic workload broker workstations or for workstation with a Tivoli Workload Scheduler agent version 8.2 or prior.

server *ServerID*
Use the **server** attribute in the fault-tolerant agent workstation definition to reduce the time required to initialize agents and to improve the timeliness of messages. By default, communications with the fault-tolerant agents are handled by a **mailman** process running on the domain manager. The **server** attribute allows you to start a **mailman** process on the domain manager to handle communications with this fault-tolerant agent workstation only.

If you are defining a fault-tolerant agent that can work as backup domain manager, the *ServerID* is used only when the workstation works as afault-tolerant agent; the setting is ignored when the workstation works as a backup domain manager.

Within the *ServerID*, the ID is a single letter or a number (A-Z and 0-9). The IDs are unique to each domain manager, so you can use the same IDs in other domains without conflict. A specific *ServerID* can be dedicated to more than one fault-tolerant agent workstation.

As best practices:

- If more than 36 server IDs are required in a domain, consider the possibility to split the domain into two or more domains.
- If the same ID is used for multiple agents, a single server is created to handle their communications. Define extra servers to prevent a single server from handling more than eight agents.

If a *ServerID* is not specified, communications with the agent are handled by the main **mailman** process on the domain manager.

protocol *http | https*
Specifies the protocol to use to communicate with:

The broker workstation
If the workstation is an agent workstation.

The remote engine
If the workstation is a remote engine workstation.

members *[workstation] [...]*
Use this value for a pool workstation to specify the dynamic agents that you want to add to the pool.

requirements *jsdl_definition*

Use this value for a dynamic pool workstation to specify the requirements, in .JSDL format, the agents must satisfy to automatically belong to the dynamic pool. You use the following syntax:

```
jsdl_definition:
<jsdl:resources>
<jsdl:logicalResource subType="MyResourceType"/>
</jsdl:resources>
```

For more information about JSDL syntax, see *Scheduling Workload Dynamically*.

Note: You can add workstation definitions to the database at any time, but you must run **JnextPlan -for 0000** again to be able to run jobs on newly created workstations. Every time you run **JnextPlan** all workstations are stopped and restarted.

Examples

The following example creates a master domain manager named `hdq-1`, and a fault-tolerant agent named `hdq-2` in the master domain. Note that a **domain** argument is optional in this example, because the master domain defaults to **masterdm**.

```
cpuname hdq-1 description "Headquarters master DM"
     os unix
     tz America/Los_Angeles
     node sultan.ibm.com
     domain masterdm
     for maestro type manager
          autolink on
          fullstatus on
end

cpuname hdq-2
     os wnt
     tz America/Los_Angeles
     node opera.ibm.com
     domain masterdm
     for maestro type fta
          autolink on
end
```

The following example creates a domain named `distr-a` with a domain manager named `distr-a1` and a standard agent named `distr-a2`:

```
domain distr-a
     manager distr-a1
     parent masterdm
end

cpuname distr-a1 description "District A domain mgr"
     os wnt
     tz America/New_York
     node pewter.ibm.com
     domain distr-a
     for maestro type manager
          autolink on
          fullstatus on
end

cpuname distr-a2
     os wnt
     node quatro.ibm.com
     tz America/New_York
     domain distr-a
     for maestro type s-agent
end
```

The following example creates a fault-tolerant workstation with SSL authentication. The **securitylevel** security definition specifies that the connection between the workstation and its domain manager can be only of the SSL type. Port 32222 is reserved for listening for incoming SSL connections.

```
cpuname ENNETI3
     os WNT
     node apollo
     tcpaddr 30112
     secureaddr 32222
     for maestro
          autolink off
          fullstatus on
          securitylevel on
end
```

The following example creates a broker workstation. This workstation manages the lifecycle of Tivoli Workload Scheduler workload broker type jobs in dynamic workload broker.

```
cpuname ITDWBAGENT
       vartable TABLE1
       os OTHER
      node itdwbtst11.ibm.com TCPADDR 41114
      timezone Europe/Rome
      domain MASTERDM
      for MAESTRO
          type  BROKER
          autolink  OFF
          behindfirewall OFF
          fullstatus OFF
end
```

The following example creates a remote engine workstation to use to manage cross dependencies and communicate with a remote engine installed on a system with hostname London-hdq using the default HTTPS port 31116. The remote engine workstation is hosted by the broker workstation ITDWBAGENT

```
cpuname REW_London
     description "Remote engine workstation to communicate with London-hdq"
     os WNT
     node London-hdq secureaddr 31116
     timezone Europe/London
     for maestro host ITDWBAGENT
        type rem-eng
        protocol HTTPS
end
```

The following example shows how to create a dynamic pool of dynamic agents. All dynamic agents in the dynamic pool must have the HP-UX or Linux operating systems installed:

```
CPUNAME DPOOLUNIX
  DESCRIPTION "Sample Dynamic Pool Workstation"
  VARTABLE table1
  OS OTHER
  TIMEZONE Europe/Rome
  FOR MAESTRO HOST MAS86MAS_DWB
    TYPE D-POOL
  REQUIREMENTS
    <?xml version="1.0" encoding="UTF-8"?>
```

```
<jsdl:resourceRequirements
      xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl">
  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="HPUX"/>
      <jsdl:operatingSystem type="LINUX"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
</jsdl:resourceRequirements>
END
```

The following example shows how to create a dynamic pool of dynamic agents. All dynamic agents in the dynamic pool must have the Windows 2000 operating system installed:

```
CPUNAME DPOOLWIN
  DESCRIPTION "Sample Dynamic Pool Workstation"
  OS WNT
  TIMEZONE Europe/Rome
  FOR MAESTRO HOST MAS86MAS_DWB
    TYPE D-POOL
  REQUIREMENTS
    <?xml version="1.0" encoding="UTF-8"?>
<jsdl:resourceRequirements
      xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl">
  <jsdl:resources>
    <jsdl:candidateOperatingSystems>
      <jsdl:operatingSystem type="Windows 2000"/>
    </jsdl:candidateOperatingSystems>
  </jsdl:resources>
</jsdl:resourceRequirements>
END
```

The following example shows how to create a pool of dynamic agents with name POOLUNIX and containing two dynamic agents: NC121105 and NC117248:

```
CPUNAME POOLUNIX
  DESCRIPTION "Sample Pool Workstation"
  OS OTHER
  TIMEZONE Europe/Rome
  FOR MAESTRO HOST MAS86MAS_DWB
    TYPE POOL
  MEMBERS
    NC121105
    NC117248
END
```

See also

To create a workstation definition in the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Scheduling Environment**→**Design**→**Create Workstations**
2. Select an engine name and click **Go**
3. Specify your choices in the Workstation properties panel.

Workstation class definition

A workstation class is a group of workstations for which common jobs and job streams can be written. You can include multiple workstation class definitions in the same text file, along with workstation definitions and domain definitions.

When defining workstation classes, ensure that the workstations in the class support the job types you plan to run on them. The following rules apply:

- Job types with advanced options run only on dynamic agents, pools, and dynamic pools.
- Shadow jobs run only on remote engines.

Each workstation class definition has the following format and arguments:

Syntax

cpuclass *workstationclass*
  [**description** "*description*"]
  [**ignore**]
  members [*workstation* | @] [...]
  end

[**cpuname** ...]

[**cpuclass** ...]

[**domain** ...]

Arguments

cpuclass *workstationclass*
: Specifies the name of the workstation class. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

  Note: You cannot use the same names for workstations, workstation classes, and domains.

description *"description"*
: Provides a description of the workstation class. The description can contain up to 120 alphanumeric characters. The text must be enclosed within double quotes.

ignore
: Specifies that Tivoli Workload Scheduler must ignore all workstations included in this workstation class when generating the production plan.

members *workstation*
: Specifies a list of workstation names, separated by spaces, that are members of the class. The @ wildcard character means that the workstation class includes all workstations.

Examples

The following example defines a workstation class named `backup`:

```
cpuclass backup
     members
          main
          site1
          site2
end
```

The following example defines a workstation class named `allcpus` that contains every workstation:

```
cpuclass allcpus
     members
          @
end
```

See also

To create a workstation class definition in the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **Go**
3. In the Working List toolbar of the pop-up window that opens, click **New→Workstation Class**
4. Specify your choices in the Properties - Workstation Class panel.

Domain definition

A domain is a group of workstations consisting of one or more agents and a domain manager. The domain manager acts as the management hub for the agents in the domain. You can include multiple domain definitions in the same text file, along with workstation definitions and workstation class definitions. Each domain definition has the following format and arguments:

Syntax

domain *domainname*[**description** "*description*"]
    * **manager** *workstation*
    [**parent** *domainname* | **ismaster**]
end

[**cpuname** ...]

[**cpuclass** ...]

[**domain** ...]

Arguments

domain *domainname*
: The name of the domain. It must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters. You cannot use the same names for workstations, workstation classes, and domains.

description "*description***"**
: Provides a description of the domain. The text must be enclosed within double quotes.

* **manager** *workstation*
: This is a commented field used only to show, when displaying the domain definition, the name of the workstation that has the role of domain manager for that domain. Make sure this field remains commented. It is kept for backward compatibility. With Tivoli Workload Scheduler version 8.3 the information about whether a workstation is a domain manager is set in the **type** field in the "Workstation definition" on page 127.

parent *domainname*
: The name of the parent domain to which the domain manager is linked. The default is the master domain, which does not require a domain definition. The master domain is defined by the global options **master** and **master domain**.

ismaster

If specified, indicates that the domain is the top domain of the Tivoli Workload Scheduler network. If set it cannot be removed later.

Examples

The following example defines a domain named `east`, with the master domain as its parent, and two subordinate domains named `northeast` and `southeast`:

```
domain east
     description "The Eastern domain"
     * manager cyclops
end
domain northeast
     description "The Northeastern domain"
     * manager boxcar
     parent east
end
domain southeast
     description "The Southeastern domain"
     * manager sedan
     parent east
end
```

See also

In the Dynamic Workload Console, to create a domain definition you have to go through the workstation definition process as follows:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Design→Create Workstations**
2. Select an engine name and click **Go**
3. In the Workstation properties panel, click **Assign to domain**.
4. In the Select domain panel click **New**.
5. Specify your choices in the Domain properties panel.

Job definition

A job is an executable file, program, or command that is scheduled and launched by Tivoli Workload Scheduler. You can write job definitions in edit files and then add them to the Tivoli Workload Scheduler database with the composer program. You can include multiple job definitions in a single edit file.

Each job definition has the following format and arguments:

Syntax

$jobs
[*workstation*#]*jobname*
  {**scriptname** *filename* **streamlogon** *username* |
   docommand "*command*" **streamlogon** *username* |
   task *job_definition* [**streamlogon** *username*]}
  [**description** "*description*"]
  [**tasktype** *tasktype*]
  [**interactive**]
  [**rccondsucc** "*Success Condition*"]
  [**recovery**

{**stop** | **continue** | **rerun**}
[**after** [*workstation#*]*jobname*]
[**abendprompt** "*text*"]]

A job itself has no settings for dependencies, these must be added to the job when it is included in a job stream definition.

You can add or modify job definitions from within job stream definitions. Modifications to jobs definitions made in job streams definitions are reflected in the job definitions stored in the database. This means that if you modify the job definition of `job1` in job stream definition `js1` and `job1` is used also in job stream `js2`, also the definition of `job1` in `js2` definition is modified accordingly.

Note: Wrongly typed keywords used in job definitions lead to truncated job definitions stored in the database. In fact the wrong keyword is considered extraneous to the job definition and so it is interpreted as the job name of an additional job definition. Usually this misinterpretation causes also a syntax error or an inexistent job definition error for the additional job definition.

Refer to section "Job stream definition" on page 183 for information on how to write job streams definitions.

Arguments

*workstation***#**

Specifies the name of the workstation or workstation class on which the job runs. The default is the workstation specified for *defaultws* when starting the **composer** session.

For more information on how to start a composer session refer to "Running the composer program" on page 237. The pound sign (**#**) is a required delimiter. If you specify a workstation class, it must match the workstation class of any job stream in which the job is included.

If you are defining a job that manages a workload broker job, specify the name of the workstation where the workload broker workstation is installed. Using the workload broker workstation, Tivoli Workload Scheduler can submit job in the dynamic workload broker environment using the dynamic job submission.

jobname

Specifies the name of the job. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 40 characters.

scriptname *filename*

Specifies the name of the file the job runs. Use **scriptname** for UNIX and Windows jobs. For an executable file, enter the file name and any options and arguments. The length of *filename* plus the length of *Success Condition* (of the **rccondsucc** keyword) must not exceed 4095 characters. You can also use Tivoli Workload Scheduler parameters.

See "Using variables and parameters in job definitions" on page 170 for more information.

For Windows jobs, include the file extensions. Universal Naming Convention (UNC) names are permitted. Do not specify files on mapped drives.

If you are defining a job that manages a workload broker job specify the name of the workload broker job. Additionally you can specify variables and the type of affinity that exists between the Tivoli Workload Scheduler job and the workload broker job using the syntax outlined in the list below. To identify an affine job using the:

Tivoli Workload Scheduler job name
: jobName [-var var1Name=var1Value,...,varNName=varNValue] [-twsAffinity jobname=twsJobName]

dynamic workload broker job ID
: jobName [-var var1Name=var1Value,...,varNName=varNValue] [-affinity jobid=jobid]

dynamic workload broker job alias
: jobName [-var var1Name=var1Value,...,varNName=varNValue] [-affinity alias=alias]

Refer to the *IBM Tivoli Workload Scheduler: Scheduling Workload Dynamically* for detailed information.

If the file path or the file name of the **scriptname** argument contains spaces, the entire string must be enclosed between "\" and \" " as shown below:

```
scriptname "\"C:\Program Files\tws\myscript.cmd\""
```

If special characters are included, other than slashes (/) and backslashes (\), the entire string must be enclosed in quotes (").

The job `fails` if the script specified in the **scriptname** option is not found or does not have execute permission. It `abends` if the script that is not found or does not have execute permission includes parameters.

docommand *command*
: Specifies a command that the job runs. Enter a valid command and any options and arguments enclosed in double quotes ("). The length of *command* plus the length of *Success Condition* (of the **rccondsucc** keyword) must not exceed 4095 characters. You can also enter Tivoli Workload Scheduler parameters.

The job `abends` if the file specified with the **docommand** option is not found or does not have execute permission.

See “Using variables and parameters in job definitions” on page 170 for more information.

task *job_definition*
: Specifies the XML syntax for job types with advanced options and shadow jobs. To define existing job types, use the **docommand** keyword. This keyword applies only to workstations of the following types:

- agent
- pool
- d-pool
- rem-eng

The syntax of the job depends on the job type you define.

You can define the following types of jobs:

- Shadow jobs that bind to other jobs in remote Tivoli Workload Scheduler for z/OS or Tivoli Workload Scheduler networks. For more information, see “Job definition - Shadow jobs” on page 154.

- Jobs that run web services commands. For more information, see "Job definition - Web services jobs" on page 155.
- Jobs that perform file transfer tasks. For more information, see "Job definition - File transfer jobs" on page 157.
- Jobs that use J2EE to send and receive messages among Java applications in the same network. For more information, see "Job definition - J2EE jobs" on page 162.
- Jobs that perform database operations. For more information, see "Job definition - Database jobs" on page 164.
- Jobs that perform Java operations. For more information, see "Job definition - Java jobs" on page 167.
- Jobs that run scripts or commands with advanced options. For more information, see "Job definition - Executable jobs" on page 167.
- Jobs that extend the capabilities of Tivoli Workload Scheduler to other applications through access methods, such SAP or PeopleSoft. For more information, see "Job definition - Access method jobs" on page 168.
- Jobs that run commands on IBM i systems. For more information, see "Job definition - IBM i jobs" on page 169.

streamlogon *username*

The user name under which the job runs. This attribute is mandatory when **scriptname** or **docommand** are specified. The name can contain up to 47 characters. If the name contains special characters it must be enclosed in double quotes ("). Specify a user that can log on to the workstation on which the job runs. You can also enter Tivoli Workload Scheduler parameters.

See "Using variables and parameters in job definitions" on page 170 for more information.

For Windows jobs, the user must also have a user definition.

See "User definition" on page 170 for user requirements.

If you are defining a job that manages a dynamic workload broker job, specify the name of the user you used to install dynamic workload broker.

The job `fails` if the user specified in the **streamologon** option does not exist.

description "*description***"**
Provides a description of the job. The text must be enclosed in double quotes.The maximum number of characters allowed is 120.

tasktype *tasktype*

Specifies the job type. It can have one of the following values:

UNIX For jobs that run on UNIX platforms.

WINDOWS
For jobs that run on Windows operating systems.

OTHER
For jobs that run on extended agents. Refer to *IBM Tivoli Workload Scheduler for Applications: User's Guide* for information about customized task types for supported vendor acquired applications.

BROKER
For jobs that manage the lifecycle of a dynamic workload broker

job. Refer to *IBM Tivoli Workload Scheduler: Scheduling Workload Dynamically* for information about how to use dynamic workload broker.

interactive

If you are defining a job that manages a dynamic workload broker job ignore this argument. Specifies that the job runs interactively on your desktop. This feature is available only on Windows environments.

rccondsucc "*Success Condition***"**

An expression which determines the return code (RC) required to consider a job successful. The success condition can be a maximum of 256 characters. This expression can be one of the following:

COMPLETE_IF_BIND_FAILS

This setting applies to shadow jobs only. When specified, the shadow job status is automatically set to SUCC if the bind with the remote job fails.

Comparison expression

Specifies the job return codes. The syntax is:

```
(RC operator operand)
```

RC The RC keyword.

operator

Comparison operator. It can have the following values:

Table 23. Comparison operators

Example	Operator	Description
RC<a	<	Less than
RC<=a	<=	Less than or equal to
RC>a	>	Greater than
RC>=a	>=	Greater than or equal to
RC=a	=	Equal to
RC!=a	!=	Not equal to
RC<>a	<>	Not equal to

operand

An integer between -2147483647 and 2147483647.

For example, you can define a successful job as a job that ends with a return code less than or equal to 3 as follows:

```
rccondsucc "(RC <= 3)"
```

Boolean expression

Specifies a logical combination of comparison expressions. The syntax is:

```
comparison_expression operator comparison_expression
```

comparison_expression

The expression is evaluated from left to right. You can use parentheses to assign a priority to the expression evaluation.

operator

Logical operator. It can have the following values:

Table 24. Logical operators

Example	Operator	Result
expr_a and expr_b	And	TRUE if both expr_a and expr_b are TRUE.
expr_a or expr_b	Or	TRUE if either expr_a or expr_b is TRUE.
Not expr_a	Not	TRUE if expr_a is not TRUE.

For example, you can define a successful job as a job that ends with a return code less than or equal to 3 or with a return code not equal to 5, and less than 10 as follows:

```
rccondsucc "(RC≤3) OR ((RC≠5) AND (RC<10))"
```

recovery

Recovery options for the job. The default is **stop** with no recovery job and no recovery prompt. Enter one of the recovery options, **stop**, **continue**, or **rerun**. This can be followed by a recovery job, a recovery prompt, or both.

stop If the job ends abnormally, do not continue with the next job.

continue
If the job ends abnormally, continue with the next job. The job is not listed as abended in the properties of the job stream. If no other problems occur, the job stream completes successfully.

rerun If the job ends abnormally, rerun the job.

after [*workstation***#]***jobname*

Specifies the name of a recovery job to run if the parent job ends abnormally. Recovery jobs are run only once for each abended instance of the parent job.

You can specify the recovery job's workstation if it is different from the parent job's workstation. The default is the parent job's workstation. Not all jobs are eligible to have recovery jobs run on a different workstation. Follow these guidelines:

- If either workstation is an extended agent, it must be hosted by a domain manager or a fault-tolerant agent with a value of **on** for **fullstatus**.
- The recovery job workstation can be in the same domain as the parent job workstation or in a higher domain.
- If the recovery job workstation is a fault-tolerant agent, it must have a value of **on** for **fullstatus**.

abendprompt "*text***"**
Specifies the text of a recovery prompt, enclosed in double quotes, to be displayed if the job ends abnormally. The text can contain up to 64 characters. If the text begins with a colon (:), the prompt is displayed, but no reply is required to continue processing. If the text begins with an exclamation mark (!), the prompt is displayed, but it is not recorded in the log file. You can also use Tivoli Workload Scheduler parameters.

See "Using variables and parameters in job definitions" on page 170 for more information.

Table 25 summarizes all possible combinations of recovery options and actions.

The table is based on the following criteria from a job stream called `sked1`:

- Job stream `sked1` has two jobs, `job1` and `job2`.
- If selected for `job1`, the recovery job is `jobr`.
- `job2` is dependent on `job1` and does not start until `job1` has completed.

Table 25. Recovery options and actions

	Stop	**Continue**	**Rerun**
Recovery prompt: No **Recovery job**: No	Intervention is required.	Run `job2`.	Rerun `job1`. If `job1` ends abnormally, issue a prompt. If reply is *yes*, repeat above. If `job1` is successful, run `job2`.
Recovery prompt: Yes **Recovery job**: No	Issue recovery prompt. Intervention is required.	Issue recovery prompt. If reply is *yes*, run `job2`.	Issue recovery prompt. If reply is *yes*, rerun `job1`. If `job1` ends abnormally, repeat above. If `job1` is successful, run `job2`.
Recovery prompt: No **Recovery job**: Yes	Run `jobr`. If it ends abnormally, intervention is required. If it is successful, run `job2`.	Run `jobr`. Run `job2`.	Run `jobr`. If `jobr` ends abnormally, intervention is required. If `jobr` is successful, rerun `job1`. If `job1` ends abnormally, issue a prompt. If reply is *yes*, repeat above. If `job1` is successful, run `job2`.
Recovery prompt: Yes **Recovery job**: Yes	Issue recovery prompt. If reply is *yes*, run `jobr`. If it ends abnormally, intervention is required. If it is successful, run `job2`.	Issue recovery prompt. If reply is *yes*, run `jobr`. Run `job2`.	Issue recovery prompt. If reply is *yes*, run `jobr`. If `jobr` ends abnormally, intervention is required. If `jobr` is successful, rerun `job1`. If `job1` ends abnormally, repeat above. If `job1` is successful, run `job2`.

Notes:

1. "Intervention is required" means that `job2` is not released from its dependency on `job1`, and therefore must be released by the operator.
2. The **continue** recovery option overrides the ends abnormally state, which might cause the job stream containing the ended abnormally job to be marked as successful. This prevents the job stream from being carried forward to the next production plan.
3. If you select the **rerun** option without supplying a recovery prompt, Tivoli Workload Scheduler generates its own prompt.
4. To reference a recovery job in **conman**, use the name of the original job (`job1` in the scenario above, not `jobr`). Only one recovery job is run for each abnormal end.

Examples

The following is an example of a file containing two job definitions:

```
$jobs
cpu1#gl1
     scriptname "/usr/acct/scripts/gl1"
     streamlogon acct
     description "general ledger job1"
bkup
     scriptname "/usr/mis/scripts/bkup"
     streamlogon "^mis^"
     recovery continue after recjob1
```

The following example shows how to define the Tivoli Workload Scheduler TWSJOB job that manages the workload broker broker_1 job that runs on the same workload broker agent where the TWSJOB2 ran:

```
ITDWBAGENT#TWSJOB
SCRIPTNAME "broker_1 -var var1=name,var2=address
           -twsaffinity jobname=TWSJOB2"
STREAMLOGON brkuser
DESCRIPTION "Added by composer."
TASKTYPE BROKER
RECOVERY STOP
```

The following example shows how to define a job which is assigned to a dynamic pool of UNIX dynamic agents and runs the `df` script:

```
DPOOLUNIX#JOBDEF7
 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">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>df</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

The following example shows how to define a job which is assigned to a dynamic pool of Windows dynamic agents and runs the `dir` script:

```
DPOOLWIN#JOBDEF6
 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">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>dir</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

The following example shows how to define a job which is assigned to the NC115084 dynamic agent and runs the `dir` script:

```
NC115084#JOBDEF3
 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">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>dir</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

The following example shows how to define a job which is assigned to a pool of UNIX dynamic agents and runs the script defined in the `script` tag:

```
POOLUNIX#JOBDEF5
 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">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>#!/bin/sh
sleep 60
dir</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

The following example shows how to define a job which is assigned to a pool of Windows dynamic agents and runs the script defined in the `script` tag:

```
POOLWIN#JOBDEF4
 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">
    <jsdl:application name="executable">
    <jsdle:executable interactive="false">
    <jsdle:script>ping -n 120 localhost</jsdle:script>
    </jsdle:executable>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Added by composer."
 RECOVERY STOP
```

See also

You can write job definitions using the Dynamic Workload Console, which creates the job with the appropriate syntax:

1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **Go**
3. In the Working List toolbar of the pop-up window that opens, click **New→Job Definition**
4. Click on a type of job definition and specify your choices in the ensuing properties panel.

Job definition - Shadow jobs

Shadow jobs are defined using XML syntax. The key attributes to identify the remote job instance and the matching criteria depend on the type of remote engine where the remote job instance is defined. Fields highlighted in bold are those used to identify the remote job instance.

z/OS Because z/OS engines support only closest preceding matching criteria the XML template to define a z/OS shadow job is the following:

```
$JOBS
WORKSTATION#ZSHADOW_CLOS_PRES
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition
       xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
       xmlns:zshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/zshadow">
    <jsdl:application name="zShadowJob">
    <zshadow:ZShadowJob>
    <zshadow:JobStream>JobStream</zshadow:JobStream>
    <zshadow:JobNumber>JobNumber</zshadow:JobNumber>
    <zshadow:matching>
    <zshadow:previous/>
    </zshadow:matching>
    </zshadow:ZShadowJob>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Sample Job Definition"
 RECOVERY STOP
```

Note: Make sure that you enter valid settings in the **JobStream** and **JobNumber** fields.

Distributed Distributed shadow jobs, instead, support the four matching criteria available for external follows dependencies. The following shows the XML templates you can use to define distributed shadow jobs:

Matching criteria: Closest preceding

XML sample:

```
$JOBS
WORKSTATION#DSHADOW_CLOS_PRES
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition
       xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
       xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
    <jsdl:application name="distributedShadowJob">
    <dshadow:DistributedShadowJob>
    <dshadow:JobStream>JobStream</dshadow:JobStream>
    <dshadow:Workstation>Workstation</dshadow:Workstation>
    <dshadow:Job>Job</dshadow:Job>
    <dshadow:matching>
    <dshadow:previous/>
    </dshadow:matching>
    </dshadow:DistributedShadowJob>
    </jsdl:application>
    </jsdl:jobDefinition>
 DESCRIPTION "Sample Job Definition"
 RECOVERY STOP
```

Matching criteria: Within an absolute interval

XML sample:

```
$JOBS
WORKSTATION#DSHADOW_ABSOLUTE
 TASK
    <?xml version="1.0" encoding="UTF-8"?>
    <jsdl:jobDefinition
          xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
          xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
    <jsdl:application name="distributedShadowJob">
```

```
      <dshadow:DistributedShadowJob>
      <dshadow:JobStream>JobStream</dshadow:JobStream>
      <dshadow:Workstation>Workstation</dshadow:Workstation>
      <dshadow:Job>Job</dshadow:Job>
      <dshadow:matching>
      <dshadow:absolute from="0600 -4" to="1100 +3"/>
      </dshadow:matching>
      </dshadow:DistributedShadowJob>
      </jsdl:application>
      </jsdl:jobDefinition>
   DESCRIPTION "Sample Job Definition"
   RECOVERY STOP
```

Matching criteria: Within a relative interval

```
  $JOBS
  WORKSTATION#DSHADOW_RELATIVE
   TASK
      <?xml version="1.0" encoding="UTF-8"?>
      <jsdl:jobDefinition
            xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
            xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
      <jsdl:application name="distributedShadowJob">
      <dshadow:DistributedShadowJob>
      <dshadow:JobStream>JobStream</dshadow:JobStream>
      <dshadow:Workstation>Workstation</dshadow:Workstation>
      <dshadow:Job>Job</dshadow:Job>
      <dshadow:matching>
      <dshadow:relative from="-400" to="+500" />
      </dshadow:matching>
      </dshadow:DistributedShadowJob>
      </jsdl:application>
      </jsdl:jobDefinition>
   DESCRIPTION "Sample Job Definition"
   RECOVERY STOP
```

Matching criteria: Same scheduled date

XML sample:

```
  $JOBS
  WORKSTATION#DSHADOW_SAMEDAY
   TASK
      <?xml version="1.0" encoding="UTF-8"?>
      <jsdl:jobDefinition
            xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
            xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
      <jsdl:application name="distributedShadowJob">
      <dshadow:DistributedShadowJob>
      <dshadow:JobStream>JobStream</dshadow:JobStream>
      <dshadow:Workstation>Workstation</dshadow:Workstation>
      <dshadow:Job>Job</dshadow:Job>
      <dshadow:matching>
      <dshadow:sameDay/>
      </dshadow:matching>
      </dshadow:DistributedShadowJob>
      </jsdl:application>
      </jsdl:jobDefinition>
   DESCRIPTION "Sample Job Definition"
   RECOVERY STOP
```

For more information about the matching criteria, see "Managing external follows dependencies for jobs and job streams" on page 52.

Job definition - Web services jobs

This section describes the required and optional attributes for web services jobs. Each job definition has the following format and arguments:

Table 26. Required and optional attributes for the definition of a web services job

Attribute	Description/value	Required
application name	ws	✓
operation	The name of the web service command you are invoking.	✓

Table 26. Required and optional attributes for the definition of a web services job (continued)

Attribute	Description/value	Required
wsdlURL	The name of the web service URL.	✓
arguments	The parameters required by the web service command you are invoking (the number of values depends on the command).	

credentials	The name and password of the user running this job. As an alternative to hard-coding actual values, you can parametrize in one of the following ways:
	• Enter a *username* specified in the database with the user definition (it is applicable to all operating systems on this job type) and key the statement:
	`<jsdl:password>${password:username}</jsdl:password>`
	The password is retrieved from the *username* user definition in the database and resolved at runtime. See "Utilizing user definitions on job types with advanced options" on page 173 for further details.
	You can also specify the user of a different workstation and use the following syntax for the password:
	`<jsdl:password>${password:workstation#username}`
	`</jsdl:password>`
	• Enter a user and password defined with the `param` utility command locally on the dynamic agent that will run the job (if the job is to be submitted to a pool or to a dynamic pool, the definition must be present on all the agents of the pool). Provided you defined the user name with the variable *user* and a password, the corresponding credential statements would be:
	`<jsdl:userName>${agent:user}</jsdl:userName>`
	`<jsdl:password>${agent:password.user}</jsdl:password>`
	The user and password variables are resolved on the agent at runtime. See "Defining variables and passwords for local resolution on dynamic agents" on page 424 for further details.
	If you use an HTTPS connection, ensure that the security certificates are configured for Job Manager on the workstation where the job is to run.	

The command output is recorded in the job log.

The following example shows a task that runs the `getSum` web services command. The task definition provides in the `arguments` section the two values that must be added.

```
$JOBS
AGENT#WEB_SERVICE
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlws="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlws" description="Calls a web
service to perform a sum of two numbers" name="SumNumber">
<jsdl:annotation>annotation</jsdl:annotation>
<jsdl:variables>
<jsdl:stringVariable description="URL of Web Service"
name="wsdlURL">http://np515516.cyber.com:9080/
Sum/services/Sum/wsdl/Sum.wsdl</jsdl:stringVariable>
<jsdl:stringVariable description="Operation to Invoke"
name="Operation">getSum</jsdl:stringVariable>
</jsdl:variables>
```

```
<jsdl:application name="ws">
<jsdlws:ws>
<jsdlws:wsToInvoke operation="${Operation}" wsdlURL="${wsdlURL}">
<jsdlws:arguments>
<jsdlws:value>1</jsdlws:value>
<jsdlws:value>2</jsdlws:value>
</jsdlws:arguments>
<jsdlws:credentials>
<jsdl:userName>administrator</jsdl:userName>
<jsdl:password>password</jsdl:password>
</jsdlws:credentials>
</jsdlws:wsToInvoke>
</jsdlws:ws>
</jsdl:application>
<jsdl:resources>
<jsdl:candidateHosts>
<jsdl:hostName>${host}</jsdl:hostName>
</jsdl:candidateHosts>
</jsdl:resources>
</jsdl:jobDefinition>
```

The following example applies if you use an HTTPS connection with the agent running the web services task. It shows how you configure the `JVMOptions` key in the `jobManager.ini` file of the agent to point to the security certificates.

```
JVMOptions=-Djavax.net.ssl.keyStore=/images/ITAuser/TWA/TWS/JavaExt/cfg/agentKeystore.jks
-Djavax.net.ssl.keyStorePassword=tdwb8nxt
-Djavax.net.ssl.trustStore=/images/ITAuser/TWA/TWS/JavaExt/cfg/agentKeystore.jks
-Djavax.net.ssl.trustStorePassword=tdwb8nxt
-Djavax.net.ssl.trustStoreType=JKS
```

| By default, the timeout period for web services jobs is 90 seconds. If the web
| service has not completed successfully within the timeout period, the Tivoli
| Workload Scheduler job associated to it ends in error. To prevent jobs from failing
| when web services take longer than 90 seconds to complete, you can customize the
| timeout.

Timeout period is specified in the `WSJobExecutor.properties` file that is located in `TWA_HOME/TWS/JavaExt/cfg` directory and is structured as follows:

```
# Web Service timeout
# Specify the timeout in seconds
# Default Value is 90 seconds
TIMEOUT=90
```

For example, for a web service that takes 5 minutes to complete, customize the configuration file specifying: `TIMEOUT=300`.

Job definition - File transfer jobs

This section describes the required and optional attributes for file transfer jobs. Each job definition has the following format and arguments:

Table 27. Required and optional attributes for the definition of a file transfer job

Attribute	Description/value	Required
application name	filetransfer	✓
File transfer type (upload or download)	Enclose all the file transfer attributes between `jsdlfiletransfer:uploadInfo` or `jsdlfiletransfer:downloadInfo` tags as shown in the example.	✓
server	The address of the file transfer server (and the port number, if other than the standard port, when you choose FTP as protocol).	✓
localfile and remotefile	• If uploading, `localfile` is the fully qualified path and name of the file to be uploaded, while `remotefile` is the fully qualified path and name of the file to be created on the remote target. • If downloading, `localfile` is the fully qualified path and name of the file to be created on the local target, while `remotefile` is the fully qualified path and name of the file to be downloaded.	✓

Table 27. Required and optional attributes for the definition of a file transfer job (continued)

Attribute	Description/value	Required

localCredentials and remoteCredentials	The names and passwords of the authorized users on the local and remote systems. As an alternative to hard-coding actual values, you can parametrize in one of the following ways:
	• Enter a *username* specified in the database with the user definition (it is applicable to all operating systems on this job type) and key the statement:
	`<jsdl:password>${password:`*`username`*`}</jsdl:password>`
	The password is retrieved from the *username* user definition in the database and resolved at runtime. See "Utilizing user definitions on job types with advanced options" on page 173 for further details.
	You can also specify the user of a different workstation and use the following syntax for the password:
	`<jsdl:password>${password:`*`workstation#username`*`}</jsdl:password>`
	• Enter a user and password defined with the `param` utility command locally on the dynamic agent that will run the job (if the job is to be submitted to a pool or to a dynamic pool, the definition must be present on all the agents of the pool). Provided you defined the user name with the variable *user* and a password, the corresponding credential statements would be:
	`<jsdl:userName>${agent:`*`user`*`}</jsdl:userName>`
	`<jsdl:password>${agent:password.`*`user`*`}</jsdl:password>`
	The user and password variables are resolved on the agent at runtime. See "Defining variables and passwords for local resolution on dynamic agents" on page 424 for further details.
		✓

protocol	Can be: **WINDOWS** The Microsoft file sharing protocol. If you do not specify a protocol, and SSH does not work, WINDOWS is assumed. Specify the shared directory in the `remotefile` keyword, without specifying any paths where the shared directory is nested. Specify the address of the workstation hosting the shared directory in the `server` keyword. The default working directory is: `C:\Program Files\IBM\TWA\TWS\ITA`. **SSH** A network protocol that provides file access, file transfer, and file management functions over any reliable data stream. If you do not specify a protocol, SSH is assumed as default. **FTP** A standard network protocol used to exchange files over a TCP/IP-based network, such as the Internet. **FTPS** An extension to the File Transfer Protocol (FTP) that adds support for the Tranport Layer Security (TLS) cryptographic protocol. Specifically, the file transfer is performed using implicitly the TLS security protocol for the FTP sessions, providing a private security level for the data connection. TLS protocol version 1 is supported. The SSL session reuse configuration is not supported. **FTPES** An extension to the File Transfer Protocol (FTP) that adds support for the Tranport Layer Security (TLS) cryptographic protocol. Specifically, the file transfer is performed using explicitly the TLS security protocol for the FTP sessions, providing a private security level for the data connection. TLS protocol version 1 is supported. The SSL session reuse configuration is not supported.	
transferMode	Can be `binary` (the default) or `ascii`.	

Table 27. Required and optional attributes for the definition of a file transfer job (continued)

Attribute	Description/value	Required

remoteCodepage	The codepage used on the remote workstation. If you want to use a custom code page, define the `remoteCodepage` parameter as follows:
	`<jsdlfiletransfer:remoteCodepage>USER:`*`CUSTOM_CP`*
	`</jsdlfiletransfer:remoteCodepage>`
	
	where *CUSTOM_CP* is the code page defined by the user.
	
	For example, to use the `tcpip.ftpd.ftpxlbin.frence3` custom code page, define the `remoteCodepage` parameter as follows:
	`<jsdlfiletransfer:remoteCodepage>USER:tcpip.ftpd.ftpxlbin.frence3`
	`</jsdlfiletransfer:remoteCodepage>`
		Required if you specify localCodepage

localCodepage	The codepage used on the local workstation.	Required if you specify remoteCodepage
Timeout	Specifies the number of seconds to be used for the file transfer operation. The default value is 60 seconds.	

portsRange	When the active mode is enabled, the `portsRange` section restricts the port numbers sent by the FTP PORT command. This option accommodates highly restrictive firewall rules. The `portsRange` section defines the port range to use on the client side of TCP data connections. If you do not specify the `portsRange` section, the operating system determines the port numbers to be used.
	
	The `portsRange` parameter requires the following parameters:
	min *(min_port)*
	The minimum port value to use on the client side of TCP data connections. The values allowed range from 0 to 65535. For example, if you set this value to 1035, Tivoli Workload Scheduler restricts the port numbers to be greater than or equal to port 1035
	max *(max_port)*
	The maximum port value to use on the client side of TCP data connections. The values allowed range from 0 to 65535. For example, if you set this value to 1038, Tivoli Workload Scheduler restricts the port number to be less than or equal to port 1038.
	
	For example, to limit the port range to use on the client side between port 1035 and port 1040, specify the following:
	`<jsdlfiletransfer:portsRange>`
	`<jsdlfiletransfer:min>1035</jsdlfiletransfer:min>`
	`<jsdlfiletransfer:max>1040</jsdlfiletransfer:max>`
	`</jsdlfiletransfer:portsRange>`	

passiveMode	Specifies whether the server is passive or active in establishing connections for data transfers. If you set this option to `NO`, the server establishes the data connection with the client (active mode). If you set this option to `YES`, the client establishes the data connection with the server (passive mode). The default value is `NO`.	

The following xml file shows the JSDL "application" section of a sample job definition for a file transfer job type:

```
<jsdl:application name="filetransfer">
<jsdlfiletransfer:filetransfer>
<jsdlfiletransfer:downloadInfo>
<jsdlfiletransfer:server>FTP_SERVER</jsdlfiletransfer:server>
<jsdlfiletransfer:localfile>LOCAL_FILE</jsdlfiletransfer:localfile>
<jsdlfiletransfer:remotefile>REMOTE_FILE</jsdlfiletransfer:remotefile>
<jsdlfiletransfer:remoteCredentials>
<jsdl:userName>USERNAME</jsdl:userName>
<jsdl:password>PASSWORD</jsdl:password>
</jsdlfiletransfer:remoteCredentials>
<jsdlfiletransfer:protocol>PROTOCOL</jsdlfiletransfer:protocol>
<jsdlfiletransfer:transferMode>ASCII_BINARY</jsdlfiletransfer:transferMode>
<jsdlfiletransfer:codepageConversion>
<jsdlfiletransfer:remoteCodepage>RM_CP</jsdlfiletransfer:remoteCodepage>
```

```
<jsdlfiletransfer:localCodepage>LC_CP</jsdlfiletransfer:localCodepage>
</jsdlfiletransfer:codepageConversion>
<jsdlfiletransfer:timeout>CONNECTION_TIMEOUT</jsdlfiletransfer:timeout>
<jsdlfiletransfer:portsRange>
<jsdlfiletransfer:min>MIN_PORT</jsdlfiletransfer:min>
<jsdlfiletransfer:max>MAX_PORT</jsdlfiletransfer:max>
</jsdlfiletransfer:portsRange>
<jsdlfiletransfer:passiveMode>YES_NO</jsdlfiletransfer:passiveMode>
</jsdlfiletransfer:downloadInfo>
</jsdlfiletransfer:filetransfer>
</jsdl:application>
```

The following example shows a generalized task that downloads a file from a remote workstation with address 10.0.0.8:

```
$JOBS
AGENT#FILE_TRANSFER
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlns:jsdlfiletransfer="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlfiletransfer"
name="FILETRANSFER">
<jsdl:application name="filetransfer">
<jsdlfiletransfer:filetransfer>
<jsdlfiletransfer:downloadInfo>
<jsdlfiletransfer:server>10.0.0.8</jsdlfiletransfer:server>
<jsdlfiletransfer:localfile>c:\MyTextFile.txt</jsdlfiletransfer:localfile>
<jsdlfiletransfer:remotefile>./MyRemoteFile.txt</jsdlfiletransfer:remotefile>
<jsdlfiletransfer:localCredentials>
<jsdl:userName>${agent:locluser}</jsdl:userName>
<jsdl:password>${agent:password.${agent:locluser}}</jsdl:password>1
</jsdlfiletransfer:localCredentials>
<jsdlfiletransfer:remoteCredentials>
<jsdl:userName>remuser</jsdl:userName>
<jsdl:password>${password:remuser}</jsdl:password>2
</jsdlfiletransfer:remoteCredentials>
<jsdlfiletransfer:protocol>FTP</jsdlfiletransfer:protocol>
<jsdlfiletransfer:transferMode>ascii</jsdlfiletransfer:transferMode>
<jsdlfiletransfer:codepageConversion>
<jsdlfiletransfer:remoteCodepage>IBM-280</jsdlfiletransfer:remoteCodepage>
<jsdlfiletransfer:localCodepage>ISO8859-1</jsdlfiletransfer:localCodepage>
</jsdlfiletransfer:codepageConversion>
</jsdlfiletransfer:downloadInfo>
</jsdlfiletransfer:filetransfer>
</jsdl:application>
</jsdl:jobDefinition>
```

Note: in the credentials sections,

1. The local user name was defined on the agent that runs the job with a variable named `locluser` through the `param` utility command. So, the value defined for `locluser` will be retrieved at runtime from the variables file located in the agent. Likewise, the password for the value represented by `locluser` was defined on the agent with the `param` command and will be resolved at runtime from the local variables file.
2. The remote user name was defined with the `composer user` command and is stored in the database together with its password as username `remuser`. The password for `remuser` will be retrieved from the database at runtime.

The following example shows a job definition to be used to download a text file using the FTPES protocol with an active mode, a timeout of 10 seconds and a port range to use between 1035 and 1038:

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlfiletransfer="http://www.ibm.com/xmlns/prod/scheduling/1.0/
jsdlfiletransfer" name="FTPES_DOWNLOAD_TEXT">
<jsdl:application name="filetransfer">
<jsdlfiletransfer:filetransfer>
```

```
<jsdlfiletransfer:downloadInfo>
<jsdlfiletransfer:server>myServerFtp</jsdlfiletransfer:server>
<jsdlfiletransfer:localfile>d:\MyLocalFile.txt</jsdlfiletransfer:localfile>
<jsdlfiletransfer:remotefile>/tmp/MyRemoteFile.txt</jsdlfiletransfer:remotefile>
<jsdlfiletransfer:remoteCredentials>
<jsdl:userName>myUser</jsdl:userName>
<jsdl:password>myPassword</jsdl:password>
</jsdlfiletransfer:remoteCredentials>
<jsdlfiletransfer:protocol>FTPES</jsdlfiletransfer:protocol>
<jsdlfiletransfer:transferMode>ascii</jsdlfiletransfer:transferMode>
<jsdlfiletransfer:timeout>10</jsdlfiletransfer:timeout>
<jsdlfiletransfer:portsRange>
<jsdlfiletransfer:min>1035</jsdlfiletransfer:min>
<jsdlfiletransfer:max>1038</jsdlfiletransfer:max>
</jsdlfiletransfer:portsRange>
</jsdlfiletransfer:downloadInfo>
</jsdlfiletransfer:filetransfer>
</jsdl:application>
</jsdl:jobDefinition>
```

The following example shows a job definition to be used to transfer a text file using the SSH protocol:

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlfiletransfer="http://www.ibm.com/xmlns/prod/scheduling/1.0/
jsdlfiletransfer" name="SSH_UPLOAD">
<jsdl:application name="filetransfer">
<jsdlfiletransfer:filetransfer>
<jsdlfiletransfer:uploadInfo>
<jsdlfiletransfer:server>myServer</jsdlfiletransfer:server>
<jsdlfiletransfer:localfile>d:\MyLocalFile.txt</jsdlfiletransfer:localfile>
<jsdlfiletransfer:remotefile>/tmp/MyRemoteFile.txt</jsdlfiletransfer:remotefile>
<jsdlfiletransfer:remoteCredentials>
<jsdl:userName>myUser</jsdl:userName>
<jsdl:password>myPassword</jsdl:password>
</jsdlfiletransfer:remoteCredentials>
<jsdlfiletransfer:protocol>SSH</jsdlfiletransfer:protocol>
<jsdlfiletransfer:transferMode>ascii</jsdlfiletransfer:transferMode>
</jsdlfiletransfer:uploadInfo>
</jsdlfiletransfer:filetransfer>
</jsdl:application>
</jsdl:jobDefinition>
```

The following example shows a job definition to be used to transfer a text file using the WINDOWS protocol:

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdlfiletransfer="http://www.ibm.com/xmlns/prod/scheduling/1.0/
jsdlfiletransfer" name="WINDOWS_DOWNLOAD">
<jsdl:application name="filetransfer">
<jsdlfiletransfer:filetransfer>
<jsdlfiletransfer:downloadInfo>
<jsdlfiletransfer:server>myServer</jsdlfiletransfer:server>
<jsdlfiletransfer:localfile>d:\MyLocalFile.txt</jsdlfiletransfer:localfile>
<jsdlfiletransfer:remotefile>mySharedFolder\MyRemoteFile.txt
</jsdlfiletransfer:remotefile>
<jsdlfiletransfer:remoteCredentials>
<jsdl:userName>myUser</jsdl:userName>
<jsdl:password>myPassword</jsdl:password>
</jsdlfiletransfer:remoteCredentials>
<jsdlfiletransfer:protocol>WINDOWS</jsdlfiletransfer:protocol>
<jsdlfiletransfer:transferMode>ascii</jsdlfiletransfer:transferMode>
```

```
</jsdlfiletransfer: downloadInfo >
</jsdlfiletransfer:filetransfer>
</jsdl:application>
</jsdl:jobDefinition>
```

The file transfer job type provides the following return codes. Return codes are available only for completed jobs.

RC=0 File transfer completed successfully.

RC=-1 File transfer not performed. The job failed with the following error code:

```
AWKFTE007E
```

Explanation: An error occurred during the file transfer operation.

Possible reasons: Remote file not found or permission denied.

RC=-2 File transfer not performed. The job failed with the following error code:

```
AWKFTE020E
```

Explanation: Only for SSH or WINDOWS protocols. An error was returned while attempting to convert the code page.

Possible reasons: For SSH or WINDOWS protocols, the code page is automatically detected and converted. In this case, there is an error in the code page of the file to be transferred, which is not compliant with the code page of the local system.

RC=-3 File transfer not performed. The job failed with the following error code:

```
WKFTE015E
```

Explanation: An error occurred during the file transfer operation.

Possible reasons: Local file not found.

RC=-4 File transfer performed with the default code page. The job failed with the following error code:

```
AWKFTE023E
```

Explanation: The specified code page conversion was not performed. The file transfer was performed with default code pages.

Possible reason: The specified code page is unavailable.

Job definition - J2EE jobs

This section describes the required and optional attributes for web services jobs. Each job definition has the following format and arguments:

Table 28. Required and optional attributes for the definition of a J2EE job.

Attribute	Description/value	Required
application name	j2ee	✓
jms operation	The operation to be performed. Supported values are: • **send**. This is the default value. • **receive**. If you specify **receive**, you can optionally define a value for the **timeout** attribute.	
timeout	The timeout, expressed in seconds, within which the task must complete. If you do not specify a timeout or set it to 0, the task continues indefinitely.	
connectionURL	The URL of the WebSphere Application Server.	

Table 28. Required and optional attributes for the definition of a J2EE job. (continued)

Attribute	Description/value	Required
connFactory	An administered object that a client uses to create a connection to the JMS provider. To specify the connection factory, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string.	✓
destination	An administered object that encapsulates the identity of a message destination, which is where messages are delivered and consumed. To specify the destination, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string.	✓
message	The message to be sent.	✓ **Note:** This attribute is required only for the send task.

Credentials	Specify the user name and the password to be used when running the J2EE application. Use this field if global security is enabled on WebSphere Application Server. The user must be defined on WebSphere Application Server. To specify these credentials, you can use variable expressions that can contain one or more variable references such as ${var}, optionally in association with any character or with a simple string. In addition, you can parametrize in one of the following ways:
	• Enter a *username* specified in the database with the *username* user definition (it is applicable to all operating systems on this job type) and key the statement:
	`<jsdl:password>${password:username}</jsdl:password>`
	The password is retrieved from the *username* user definition in the database and resolved at runtime. See "Utilizing user definitions on job types with advanced options" on page 173 for further details.
	You can also specify the user of a different workstation and use the following syntax for the password:
	`<jsdl:password>${password:workstation#username}</jsdl:password>`
	• Enter a user and password defined with the `param` utility command locally on the dynamic agent that will run the job (if the job is to be submitted to a pool or to a dynamic pool, the definition must be present on all the agents of the pool). Provided you defined the user name with the variable *user* and a password, the corresponding credential statements would be:
	`<jsdl:userName>${agent:user}</jsdl:userName>`
	`<jsdl:password>${agent:password.user}</jsdl:password>`
	The user and password variables are resolved on the agent at runtime. See "Defining variables and passwords for local resolution on dynamic agents" on page 424 for further details.	

The following example shows a send task that sends a message to the queue MyQueue:

```
$JOBS
AGENT#JOB_NAME_JMS_SEND
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlns:jsdlj="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlj" name="JMS_JOB_SEND">
<jsdl:application name="j2ee">
<jsdlj:j2ee>
<jsdlj:jms operation="send">
<jsdlj:connectionURL>corbaloc:iiop:washost.mydomain.com:2809</jsdlj:connectionURL>
<jsdlj:connFactory>jms/MyCF</jsdlj:connFactory>
<jsdlj:destination>jms/MyQueue</jsdlj:destination>
<jsdlj:message>Submission of jms job: SEND MESSAGE</jsdlj:message>
</jsdlj:jms>
<jsdlj:credentials>
<jsdlj:userName>jtwoeeuser</jsdlj:userName>
<jsdlj:password>${password:jtwoeeuser}</jsdlj:password>1
</jsdlj:credentials>
</jsdlj:j2ee>
</jsdl:application>
</jsdl:jobDefinition>
```

Note: (1) User `jtwoeeuser` was defined on the Tivoli Workload Scheduler database using the *username* user definition command. The associated password, specified by the `${password:jtwoeeuser}` string in the task, will be retrieved from the database at runtime.

The following example shows a task that reads messages from the queue MyQueue:

```
$JOBS
AGENT#JOB_NAME_JMS_RECEIVE
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlns:jsdlj="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlj" name="JMS_JOB_RECEIVE">
<jsdl:application name="j2ee">
<jsdlj:j2ee>
<jsdlj:jms operation="receive" timeout="180">
<jsdlj:connFactory>jms/MyCF</jsdlj:connFactory>
<jsdlj:destination>jms/MyQueue</jsdlj:destination>
</jsdlj:jms>
<jsdlj:credentials>
<jsdlj:userName>userName</jsdlj:userName>
<jsdlj:password>password</jsdlj:password>
</jsdlj:credentials>
</jsdlj:j2ee>
</jsdl:application>
</jsdl:jobDefinition>
```

Job definition - Database jobs

This section describes the required and optional attributes for database jobs. Each job definition has the following format and arguments:

Table 29. Required and optional attributes for the definition of a database job

Attribute	Description/value	Required
application name	database	✓
dbms	The database type where you want the job to run. Supported values are: **db2** For the DB2 databases; **mssql** For the Microsoft SQL Server databases; **oracle** For the Oracle databases	✓
server	The host name of the server where the database is located.	✓
port	The port number for the database job.	✓
database	The name of the database.	✓
JDBC driver class name	The name of the JDBC driver class	Required if you specify a custom database.
JDBC connection string	The string that is used to connect to the database, containing database URL, username and password	Required if you specify a custom database.
JDBC jar class path	Path to the database client jar files. This value overrides the value specified in the DatabaseJobExecutor.properties configuration file, if any. If you select the Microsoft SQL Server database, version 4 of the JDBC drivers is required.	
dbStatement	The name of the database job to be run.	✓

Table 29. Required and optional attributes for the definition of a database job (continued)

Attribute	Description/value	Required

credentials	The user name and the password for accessing the database (domain users are not supported). As an alternative to hard-coding actual values, you can parametrize in one of the following ways:
	• Enter a *username* specified in the database with the user definition (it is applicable to all operating systems on this job type) and key the statement:
	`<jsdl:password>${password:username}</jsdl:password>`
	The password is retrieved from the *username* user definition in the database and resolved at runtime. See "Utilizing user definitions on job types with advanced options" on page 173 for further details.
	You can also specify the user of a different workstation and use the following syntax for the password:
	`<jsdl:password>${password:workstation#username}</jsdl:password>`
	• Enter a user and password defined with the `param` utility command locally on the dynamic agent that will run the job (if the job is to be submitted to a pool or to a dynamic pool, the definition must be present on all the agents of the pool). Provided you defined the user name with the variable *user* and a password, the corresponding credential statements would be:
	`<jsdl:userName>${agent:user}</jsdl:userName>`
	`<jsdl:password>${agent:password.user}</jsdl:password>`
	The user and password variables are resolved on the agent at runtime. See "Defining variables and passwords for local resolution on dynamic agents" on page 424 for further details.	

The following example shows a job that runs a query on a DB2 database:

```
$JOBS
AGENT#DATABASE
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlns:jsdldatabase="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdldatabase" name="database">
  <jsdl:application name="database">
    <jsdldatabase:database>
            <jsdldatabase:sqlActionInfo>
                <jsdldatabase:dbms>db2</jsdldatabase:dbms>
                <jsdldatabase:server>localhost</jsdldatabase:server>
                <jsdldatabase:port>50000</jsdldatabase:port>
                <jsdldatabase:database>TWS32</jsdldatabase:database>
                <jsdldatabase:statements>
                    <jsdldatabase:dbStatement>SELECT * FROM DWB.ARE_ABSTRACT_
RESOURCES</jsdldatabase:dbStatement>
                </jsdldatabase:statements>
                <jsdldatabase:credentials>
                    <jsdl:userName>${agent:dbvars..dbtwouser}</jsdl:userName>
                    <jsdl:password>${agent:password.${agent:dbvars..dbtwouser}}</jsdl:password>1
                </jsdldatabase:credentials>
            </jsdldatabase:sqlActionInfo>
        </jsdldatabase:database>
  </jsdl:application>
</jsdl:jobDefinition>
DESCRIPTION "Defined using composer."
RECOVERY STOP
```

Note: (1) the user name was defined on the agent that runs the job with a variable named `dbtwouser` through the `param` utility command. So, the value defined for `dbtwouser` will be retrieved at runtime from the `dbvars` variables file located in the agent. Likewise, the password for the value represented by `dbtwouser` was defined on the agent with the `param` command and will be resolved at runtime from the same variables file.

Job definition - MSSQL jobs

This section describes the required and optional attributes for MSSQL jobs. Each job definition has the following format and arguments:

Table 30. Required and optional attributes for the definition of an MSSQL job.

Attribute	Description/value	Required
application name	database	✓
dbms	The database type where you want the job to run. Because this job is specific for the Microsoft SQL Server database, the only supported value is mssql.	✓
JDBC jar class path	Path to the database client jar files. This value overrides the value specified in the DatabaseJobExecutor.properties configuration file, if any. Version 4 of the JDBC drivers is required.	
server	The host name of the server where the database is located.	✓
port	The port number for the database job.	✓
database	The name of the database.	✓
dbStatement	The SQL statement. To separate instructions, use an empty line.	✓

credentials	The user name and the password for accessing the database (domain users are not supported). As an alternative to hard-coding actual values, you can parametrize in one of the following ways:
	• Enter a *username* specified in the database with the *username* user definition (it is applicable to all operating systems on this job type) and key the statement:
	`<jsdl:password>${password:username}</jsdl:password>`
	The password is retrieved from the *username* user definition in the database and resolved at runtime. See "Utilizing user definitions on job types with advanced options" on page 173 for further details.
	• Enter a user and password defined with the `param` utility command locally on the dynamic agent that will run the job (if the job is to be submitted to a pool or to a dynamic pool, the definition must be present on all the agents of the pool). Provided you defined the user name with the variable *user* and a password, the corresponding credential statements would be:
	`<jsdl:userName>${agent:user}</jsdl:userName>`
	`<jsdl:password>${agent:password.user}</jsdl:password>`
	The user and password variables are resolved on the agent at runtime. See "Defining variables and passwords for local resolution on dynamic agents" on page 424 for further details.	

The following example shows a job that runs a job on an MSSQL database:

```
$JOBS
AGENT#MSSQLJOB
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlns:jsdldatabase="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdldatabase" name="database">
  <jsdl:application name="mssqljob">
    <jsdldatabase:database>
            <jsdldatabase:sqlActionInfo>
                <jsdldatabase:dbms>mssql</jsdldatabase:dbms>
                <jsdldatabase:server>localhost</jsdldatabase:server>
                <jsdldatabase:port>111</jsdldatabase:port>
                <jsdldatabase:database>MYDATABASE</jsdldatabase:database>
                <jsdldatabase:statements>
                    <jsdldatabase:dbStatement type="job">sada</jsdldatabase:dbStatement>
                </jsdldatabase:statements>
                <jsdldatabase:credentials>
                    <jsdl:userName>mssqluser</jsdl:userName>
|                   <jsdl:password>${agent:password.mssqluser}</jsdl:password>1
                </jsdldatabase:credentials>
            </jsdldatabase:sqlActionInfo>
        </jsdldatabase:database>
  </jsdl:application>
</jsdl:jobDefinition>
DESCRIPTION "Defined using composer."
RECOVERY STOP
```

| **Note:** (1) The password for user `mssqluser` was defined with the `param` utility
| command in the variables file on the agent that is to run the job. It will be
| resolved at run time with the defined value.

Job definition - Java jobs

This section describes the required and optional attributes for Java jobs. Each job definition has the following format and arguments:

Table 31. Required and optional attributes for the definition of a Java job.

Attribute	Description/value	Required
application name	java	✓
jarPath	The directory where the jar files are stored. This includes all jar files stored in the specified directory and all sub directories.	
className	The name of the class that the job must run.	✓
parameter key	The parameters to be used when running the Java class.	

For more information about developing a Java job, see *Tivoli Workload Automation: Developer's Guide: Extending Tivoli Workload Automation*.

The following example shows a job that runs a class with name `com.ibm.test.Test` and parameter `failExecution`:

```
$JOBS
AGENT#JAVA
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
xmlns:jsdljava="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdljava" name="java">
  <jsdl:application name="java">
    <jsdljava:java>
            <jsdljava:javaParms>
                <jsdljava:jarPath>C:\JavaExecutors</jsdljava:jarPath>
                <jsdljava:className>com.ibm.test.Test</jsdljava:className>
                <jsdljava:parameters>
                    <jsdljava:parameter key="input">failExecution</jsdljava:parameter>
                </jsdljava:parameters>
            </jsdljava:javaParms>
        </jsdljava:java>
  </jsdl:application>
</jsdl:jobDefinition> DESCRIPTION "Defined using composer."
 RECOVERY STOP
```

Job definition - Executable jobs

This section describes the required and optional attributes for executable jobs. Each job definition has the following format and arguments:

Table 32. Required and optional attributes for the definition of an executable job.

Attribute	Description/value	Required
application name	executable	✓
interactive	Specify if the job requires user intervention. This option applies only to jobs running on Windows operating systems.	✓
value	Specify the name and value of one or more arguments.	
script	Type a script to be run by the job. The script is created and ran when the job runs. You can specify the arguments in this tag, or you can type them in the **value** tag and call them in the script.	✓

The following example shows a job that pings two web sites. The address of the web sites is defined in the **value** tag and called in the **script** tag. This job has an affinity relationship with job `affine_test`, which means this job will run on the same workstation as `affine_test`:

```
$JOBS
AGENT#EXECUTABLE
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="executable">
  <jsdl:application name="executable">
    <jsdle:executable interactive="false" workingDirectory="c:\">
            <jsdle:arguments>
                <jsdle:value>www.mysite.com</jsdle:value>
                <jsdle:value>www.yoursite.com</jsdle:value>
            </jsdle:arguments>
            <jsdle:script>ping %1 ping %2</jsdle:script>
        </jsdle:executable>
  </jsdl:application>
</jsdl:jobDefinition>
DESCRIPTION "Defined using composer."
TWSAFFINITY "affine_test"
RECOVERY STOP
```

Job definition - Access method jobs

This section describes the required and optional attributes for access method jobs. Each job definition has the following format and arguments:

Table 33. Required and optional attributes for the definition of an access method job

Attribute	Description/value	Required
application name	xajob	✓
accessMethod	The name of the access method used to communicate with the external system to start the job and return the status of the job.	✓
target	The name of an option file.	
taskString	Command to be interpreted by the selected method. The maximum line length is 8 KB.	✓

credentials	The name and password of the user running this job. As an alternative to hard-coding actual values, you can parametrize in one of the following ways:
	• Enter a *username* specified in the database with the user definition (it is applicable to all operating systems on this job type) and key the statement:
	`<jsdl:password>${password:`*username*`}</jsdl:password>`
	The password is retrieved from the *username* user definition in the database and resolved at runtime. For further details, see "Utilizing user definitions on job types with advanced options" on page 173.
	You can also specify the user of a different workstation and use the following syntax for the password:
	`<jsdl:password>${password:`*workstation#username*`}`
	`</jsdl:password>`
	• Enter a user and password defined with the `param` utility command locally on the dynamic agent that will run the job (if the job is to be submitted to a pool or to a dynamic pool, the definition must be present on all the agents of the pool). If you defined the user name with the variable *user* and a password, the corresponding credential statements is:
	`<jsdl:userName>${agent:`*user*`}</jsdl:userName>`
	`<jsdl:password>${agent:password.`*user*`}</jsdl:password>`
	The user and password variables are resolved on the agent at runtime. For further details, see "Defining variables and passwords for local resolution on dynamic agents" on page 424.	

The following example shows a job that creates a file in the /methods folder using a default access method job:

```
$JOBS
AGENT#XA_JOB
TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl" xmlns:jsdlxa=
"http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlxa" name="xajob">
  <jsdl:application name="xajob">
    <jsdlxa:xajob accessMethod="unixlocl" target="optionFile">
```

```
      <jsdlxa:taskString>touch file</jsdlxa:taskString>
      <jsdlxa:credentials>
        <jsdlxa:userName>TestUser</jsdlxa:userName>
        <jsdlxa:password>{aes}IEr/DES8wRzQEij1ySQBfUR587QBxM0iwfQ1EWJaDds=</jsdlxa:password>
      </jsdlxa:credentials>
    </jsdlxa:xajob>
  </jsdl:application>
</jsdl:jobDefinition>
DESCRIPTION "Defined using composer."
RECOVERY STOP
```

Job definition - z/OS jobs

This section describes the required and optional attributes for z/OS jobs. A z/OS job runs the command you specify in the JCL tab on a JCL system. This job type runs only on Tivoli Workload Scheduler distributed - Agent for z/OS. Each job definition has the following format and arguments:

Table 34. Required and optional attributes for the definition of a z/OS job.

Attribute	Description/value	Required
application name	jcl	✓
byDefinition	The type of job submission. This is the only supported submission type.	
jclDefinition	The operation to be performed on the JCL system.	✓

The following example shows a job that returns the status of the JCL system:

```
ZOSAGENT#JCLDEF
 TASK
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl=="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlnss:jsdljcl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdljcl">
<jsdl:application name="jcl">
<jsdljcl:jcl>
<jsdljcl:JCLParameters>
<jsdljcl:jcl>
<jsdljcl:byRefOrByDef>
<jsdljcl:byDefinition>
<jsdljcl:jclDefinition>//NORMAL JOB ,'TWS JOB',CLASS=A,MSGCLASS=A,>
// MSGLEVEL=(1,1)
//*
//STEP1 EXEC PGM=IEFBR14</jsdljcl:jclDefinition>
</jsdljcl:byDefinition>
</jsdljcl:byRefOrByDef>
</jsdljcl:jcl>
</jsdljcl:JCLParameters>
<jsdljcl:JOBParameters>
<jsdljcl:jobStreamName>${tws.jobstream.name}jsdljcl:jobStreamName>${tws.jobstream.name}>
<jsdljcl:inputArrival>${tws.job.ia}jsdljcl:inputArrival>${tws.job.ia}>
</jsdljcl:JOBParameters>
</jsdljcl:jcl>
</jsdl:application>
</jsdl:jobDefinition>
 DESCRIPTION "Sample JCL Job Definition"
```

Job definition - IBM i jobs

This section describes the required and optional attributes for IBM i jobs. An IBM i jobs run the command you specify in the IBM i tab on an IBM i system (formerly known as AS/400 and i5 OS). Each job definition has the following format and arguments:

Table 35. Required and optional attributes for the definition of an IBM i job.

Attribute	Description/value	Required
application name	ibmi	✓
command	The command to be run on the IBM i system.	✓

The following example shows a job that returns the status of the IBM i system:

```
$JOBS
AGENT#IBM_I
TASK
   <?xml version="1.0" encoding="UTF-8"?>
<jsdl:jobDefinition xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
 xmlns:jsdlibmi="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdlibmi" name="ibmi">
  <jsdl:application name="ibmi">
    <jsdlibmi:ibmi>
            <jsdlibmi:IBMIParameters>
                <jsdlibmi:Task>
                    <jsdlibmi:command>wrksyssts</jsdlibmi:command>
                </jsdlibmi:Task>
            </jsdlibmi:IBMIParameters>
        </jsdlibmi:ibmi>
  </jsdl:application>
</jsdl:jobDefinition>
RECOVERY STOP
```

Using variables and parameters in job definitions

A variable is a scheduling object that is part of a variable table and is defined in the Tivoli Workload Scheduler database. It can be used by all the agents in the domain as long as the users have proper authorization in the security file.

A parameter is defined and used locally on an agent (with the `parms` utility command).

Variables and parameters have the following uses and limitations in job definitions:
- Variables and parameters are allowed in the values for the **streamlogon**, **scriptname**, **docommand**, and **abendprompt** keywords.
- A variable or parameter can be used as an entire string or as part of it.
- Multiple variables and parameters are permitted in a single field.
- Enclose variable names in carets (^), and enclose the entire string in quotation marks. Ensure that caret characters are not preceded by a backslash in the string. If necessary, include the backslash in the definition of the variable or parameter.
- Enclose parameter names in single quotes (') in UNIX, and enclose the entire string in quotation marks.
- Refer to “Variable and parameter definition” on page 175 for additional information and examples.

In the following example a variable named **mis** is used in the **streamlogon** value:

```
$jobs
cpu1#bkup
     scriptname "/usr/mis/scripts/bkup"
     streamlogon "^mis^"
     recovery continue after recjob1
```

For additional examples, see “Variable and parameter definition” on page 175.

User definition

| The user names used as the **streamlogon** value for Windows job definitions must
| have user definitions. This is not required for users who run jobs on other
| operating systems, but on job types with advanced options you can use these
| values regardless of the operating system. For further details, see “Utilizing user
| definitions on job types with advanced options” on page 173.

Each user definition has the following format and arguments:

Syntax

username[*workstation***#**][*domain*\\]*username*
password “*password*”**end**

[**username** ...]

Arguments

username [*workstation*#]*username*
Specifies the name of the user.

workstation
Specifies the workstation on which the user is allowed to launch jobs. The number sign is required. The default is blank, meaning all workstations.

[domain\]username

Specifies the name of the user. If you define a Windows user, you can specify the Windows domain of the user.

The user name can contain up to 31 characters. The Windows domain name can contain up to 16 characters, including the backslash.

If you define a user for **Windows operating systems**:

- User names are case-sensitive. Also, the user must be authorized to log on to the workstation on which Tivoli Workload Scheduler launches jobs, and have the permission to **Log on as batch**.
- If the user name is not unique, it is taken to mean a local user, a domain user, or a trusted domain user, in that order.

password
Specifies the user password. The password can contain up to 31 characters, and must be enclosed in quotes. To indicate a null password, use two consecutive double quotes with no blanks in between, "". When a user definition has been compiled, you cannot read the password. Users with appropriate security privileges can modify or delete a user, but password information is never displayed.

Comments

Passwords extracted with the `composer extract` command are of limited use. When you run the `composer extract` command on a user definition, the password is obfuscated with the "**********" reserved keyword. If you try running the composer `import`, `replace`, or `modify` commands on an extracted user password, the password replacement has no effect and the old password is maintained. Also, if you try running the composer `create`, `new`, or `add` commands on a user where the password equals the "**********" reserved keyword, the following error is returned:

```
AWSJCL521E The password specified for the Windows user "USER_NAME" does not comply
with password security policy requirements.
```

Note that the reserved keyword is a string of ten asterisks (*). You cannot define a sequence of ten asterisks as password, but you can do so with any other number of asterisks.

To fix this problem, make sure you run the `composer extract` with the ;password option.

Examples

The following example defines four users:

```
username joe
     password "okidoki"
end
#
username server#jane
     password "okitay"
end
#
username dom1\jane
     password "righto"
end
#
username jack
     password ""
end
```

See also

To create a user definition in the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**>**Workload**>**Design**>**Create Workload Definitions**.
2. Select an engine name and click **Go**.
3. In the Working List toolbar of the pop-up window that opens, click **New**>**User**.
4. Specify your choices in the Properties - User panel.

Using the Tivoli Workload Scheduler user and streamlogon definitions

On Windows workstations, user definitions are specified using **composer** in the form **[*workstation*#]*username***. The instance **[*workstation*#]*username*** uniquely identifies the Windows user in the Tivoli Workload Scheduler environment. The workstation name is optional; its absence indicates that the user named *username* is defined on all the Windows workstations in the Tivoli Workload Scheduler network. If the user named *username* is only defined on some Windows workstations in the Tivoli Workload Scheduler network, to avoid inconsistencies, you must create a user definition [*workstation*#]*username* for each workstation running on Windows where the user *username* is defined.

If you schedule a job on an dynamic agent, on a pool or a dynamic pool, the job runs with the user defined on the pool or dynamic pool. However, the user must exist on all workstations in the pool or dynamic pool where you plan to run the job.

When you define a job using **composer**, you must specify both a workstation and a valid user logon for the workstation. The logon is just a valid user name for Windows, without the workstation name. For example, in the following job definition:

```
$JOB
workstation#job01 docommand "dir"
streamlogon username
```

the value for `streamlogon` is *username* and not *workstation#username*.

However, when you use the **altpass** command, you must use the user definition in the format

```
workstation#username
```

For this command, you can omit the workstation name only when changing the password of the workstation from where you are running the command.

Trusted domain user

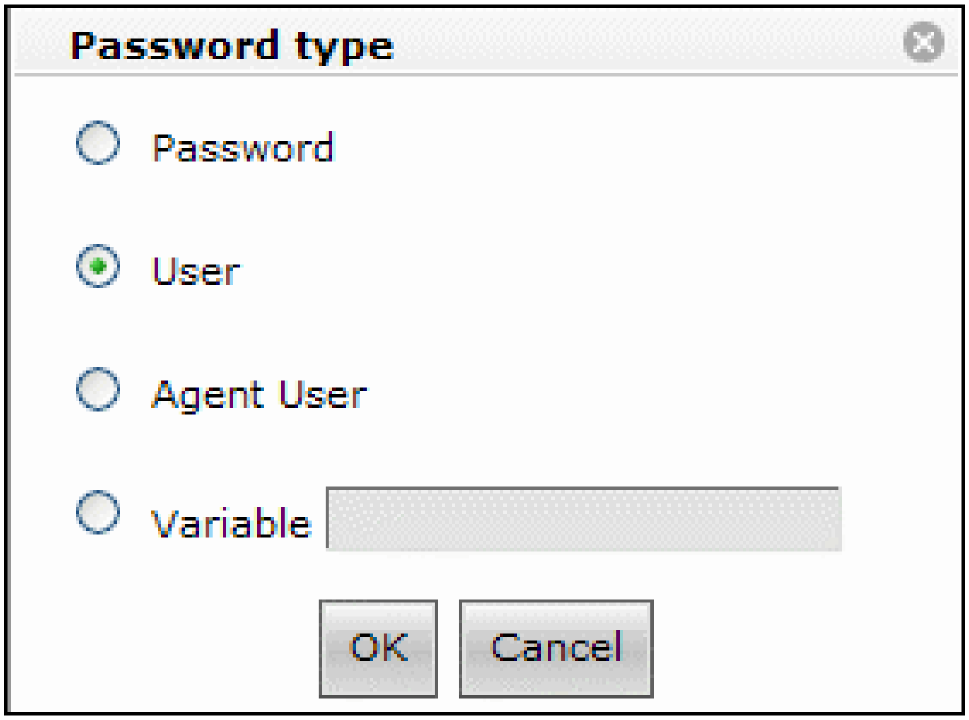
If Tivoli Workload Scheduler is to launch jobs for a trusted domain user, follow these guidelines when defining the user accounts. Assuming Tivoli Workload Scheduler is installed in `Domain1` for user account `maestro`, and user account `sue` in `Domain2` needs to launch a job, the following must be true:

- There must be mutual trust between `Domain1` and `Domain2`.
- In `Domain1` on the computers where jobs are launched, `sue` must have the right to **Log on as batch**.
- In `Domain1`, `maestro` must be a domain user.
- On the domain controllers in `Domain2`, `maestro` must have the right to **Access this computer from network**.

Utilizing user definitions on job types with advanced options

On job types with advanced options, regardless of the operating system of the dynamic agent that will run the job, you can provide the username of a user definition in the credentials section of the job and have the password field resolved at runtime with the password value stored in the database.

For example, when you define the job with the Dynamic Workload Console, you enter the username of a user definition and click the ellipsis (...) located next to the password field to display the following `Password type` widget:



where you select `User` as shown. You can likewise code this option in the task section (JSDL) of the job definition in composer. See the related sections for more information.

To be able to use this option when you define a job, you need to be authorized in the security file with the `use` access keyword for object type `userobj`, that is:

```
userobj    access=use
```

Tivoli Workload Scheduler follows this sequence when it is called to resolve the username and the password at runtime:

- If the workstation is not specified (for example, `${password:myuser}`):
  1. Searches `myuser` on the workstation running the job applying a case sensitive policy.
  2. Searches `myuser` on the workstation running the job applying a case insensitive policy.
  3. Searches `myuser` without an associated workstation applying a case sensitive policy.
  4. Searches `myuser` without an associated workstation applying a case insensitive policy.
- If the workstation is specified (for example, `${password:agent#myuser}`):
  1. Searches `myuser` on workstation `agent` applying a case sensitive policy.
  2. Searches `myuser` on workstation `agent` applying a case insensitive policy.
  3. Searches `myuser` without an associated workstation applying a case sensitive policy.
  4. Searches `myuser` without an associated workstation applying a case insensitive policy.
- If the workstation is specified but is empty (for example, `${password:#myuser}`):
  1. Searches `myuser` without an associated workstation applying a case sensitive policy.
  2. Searches `myuser` without an associated workstation applying a case insensitive policy.

Attention: User definitions lack referential integrity. This implies that, if a user definition referenced in the credentials section of a job type with advanced options is changed or deleted, no warning or error message is returned until the job is run.

Calendar definition

A calendar is a list of dates which define if and when a job stream runs. Each calendar definition has the following format and arguments:

Syntax

$calendar
calendarname [“*description*”]
    date [...]

[*calendarname* ...]

Arguments

calendarname
    Specifies the name of the calendar. The name can contain up to eight alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter.

"description"

Provides a description of the calendar. The description can contain up to 120 alphanumeric characters. It must be enclosed in double quotes. It can contain alphanumeric characters as long as it starts with a letter. It can contain the following characters: comma (,), period (.), dash (-), plus (+), single quote ('), and equal (=). It cannot contain double quotes (") other than the enclosing ones, colon (:), semi-colon (;), and ampersand (&).

date **[...]**

Specifies one or more dates, separated by spaces. The format is *mm/dd/yy*.

Examples

The following example defines three calendars named `monthend`, `paydays`, and `holidays`:

```
$calendar
monthend "Month end dates 1st half 2005"
     01/31/2005 02/28/2005 03/31/2005 04/30/2005 05/31/2005 06/30/2005
paydays
     01/15/2005 02/15/2005
     03/15/2005 04/15/2005
     05/14/2005 06/15/2005
holidays
     01/01/2005 02/15/2005 05/31/2005
```

See also

To create a calendar definition in the Tivoli Dynamic Workload Console:
1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **Go**
3. In the Working List toolbar of the pop-up window that opens, click **New→Calendar**
4. Specify your choices in the Properties - Calendar panel.

Variable and parameter definition

Variables and parameters are objects to which you assign different values.

Variables and parameters are useful when you have values that change depending on your job streams and jobs. Job stream, job, and prompt definitions that use them are updated automatically at the start of the production cycle.

Use variables and parameters as substitutes for repetitive values when defining prompts, jobs, and job streams. For example, using variables for user logon and script file names in job definitions and for file and prompt dependencies permits the use of values that can be maintained centrally in the database on the master.

While variables are scheduling objects that are defined in the Tivoli Workload Scheduler database and can be used by any authorized users in the domain, parameters are defined and used locally on individual agents.

The following sections describe variables and parameters in detail.

Variables

Variables are defined as scheduling objects in the database. Variables can be defined individually with the following command:

$parm
[*tablename.*]*variablename* "*variablevalue*"
...

where:

tablename
: Is the name of the variable table that is to contain the new variable. The variable table must be already defined. If you do not specify a variable table name, the variable is added to the default table.

variablename
: Is the name of the variable. The name can contain up to 16 alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter.

value
: Is the value assigned to the variable. The value can contain up to 72 alphanumeric characters. Do not include the names of other variables.

However, the recommended way to define variables is to use a "Variable table definition" on page 179. In any case, all variables are placed in a variable table. If you define a variable and do not specify the name of a variable table, it is included in the default variable table.

Variables can be used in job and job stream definitions to specify the **scriptname**, **docommand**, **opens**, **streamlogon**, **prompt**, and **abendprompt** keywords. They are resolved, that is replaced with their assigned value, when the production plan is generated or extended, or when you submit a job or a job stream. When you use them, specify them in the *variablename* format and enclose them between carets **^^** .

For details on variable resolution see"Variable resolution" on page 104.

In job stream and job database definitions you can set variable substitution for the following keywords:

- **abendprompt**
- **opens**
- **prompt**

The variable names specified in these definitions are first resolved against variable table definitions and then on local parameters if the variables are not found.

When you use a variable, enclose it in carets (^), and then enclose the entire string in quotation marks.

If the variable contains a portion of a path, ensure that the caret characters are not immediately preceded by a backslash (\) because, in that case, the \^ sequence could be wrongly interpreted as an escape sequence and resolved by the parser as caret character. If necessary, move the backslash into the definition of the variable between carets to avoid bad interpretation of the backslash character. For example, the following table shows the correct way for defining and using a variable named `MYDIR` in the default variable table:

Table 36. How to handle a backslash in variable substitution

Wrong way	Right way

1. Define the `MYDIR` variable as:
`$PARM`
`MYDIR "scripts"`
2. Use it in this way:
`job01 scriptname`
`"c:\operid\^MYDIR^\test.cmd"`
	1. Define the `MYDIR` variable as:
	`$PARM`
	`MYDIR "\scripts"`
	2. Use it in this way:
	`job01 scriptname`
	`"c:\operid^MYDIR^\test.cmd"`

This is true for all command line commands, graphical user interfaces, and APIs through which you use variable substitution.

Parameters

Local parameters are defined in a local database on the workstation where the jobs using them will run. To define them, you do not use this **composer** command but the “parms” on page 457 utility command.

Local parameters can be used in:
- JCL
- Log on
- Prompts dependencies
- File dependencies
- Recovery prompts

A local parameter is defined within these keywords or from within the invoked job script using the following syntax:

```
'bin\parms PARAMETERNAME'
```

Local parameters are resolved using the definitions stored in the local PARMS database as follows:
- At run time on the workstation where job processing occurs.
- At submission time on the workstation where the job or job stream is submitted from the **conman** command line. Table 37 summarizes in which **`submit`** command keyword you can use parameters.

Table 37. Keywords that can take local parameters in `submit` commands

Keyword	submit docommand (sbd command)	submit file (sbf command)	submit job (sbj command)	submit job stream (sbs command)
`abendprompt`	✓	✓	✓	
`scriptname`		✓		
`docommand`	✓			
`logon`	✓	✓		
`opens`	✓	✓	✓	✓
`prompt`	✓	✓	✓	✓

For more information on how to submit jobs and job streams in production from the **conman** command line refer to Chapter 10, “Managing objects in the plan - conman,” on page 291.

On UNIX, when you define a job or job stream in the database, you must enclose the string

```
path/parms parametername
```

between ' ' characters to ensure the parameter is solved at run time on the workstation even if a parameter with the same name is defined as a global parameter in the Tivoli Workload Scheduler database. For example, if you add to the database the following job definition:

```
$jobs
myjob
  docommand "ls ^MYDIR^"
  streamlogon "^MYUSER^"
```

and two parameters named `MYDIR` and `MYUSER` are defined in the database, then, as the production plan is created or extended, the two parameters are resolved using the definitions contained in the database and their corresponding values are carried with the `Symphony` file. If you define in the database `myjob` as follows:

```
$jobs
myjob
  docommand "ls 'bin/parms MYDIR'"
  streamlogon "'bin/parms MYUSER'"
```

then as the production plan is created or extended the only action that is performed against the two parameters in the definition of `myjob` is the removal of the ' ' characters, the parameters are carried in the `Symphony` file unresolved and then are resolved at run time locally on the target workstation using the value stored in the PARMS database.

Examples

Two parameters, `glpah` and `gllogon`, are defined as follows:

```
$parm
glpath     "/glfiles/daily"
gllogon     "gluser"
```

The `glpath` and `gllogon` parameters are used in the `gljob2` job of the`glsched` job stream:

```
schedule glsched on weekdays
:
gljob2
     scriptname "/usr/gl^glpath^"
     streamlogon "^gllogon^"
     opens "^glpath^/datafile"
     prompt ":^glpath^ started by ^gllogon^"
end
```

An example of a variable used with the **docommand** keyword is:

```
docommand "ls ^MY_HOME^"
```

Creating a variable definition using the Dynamic Workload Console

To create a variable definition in the Dynamic Workload Console, you must add it to a variable table definition:

1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **Go**

3. Open in edit mode an existing variable table from the Quick Open pane, or create a new variable table as described in “Creating a variable table definition using the Dynamic Workload Console” on page 181
4. In the Properties - Variable Table panel, click the Variables tab and add new variable definitions by clicking the "+" (Add) icon and specifying variable names and values

Variable table definition

A variable table is an object that groups multiple variables. All the global parameters (now named *variables*) that you use in workload scheduling are contained in at least one variable table. Two ways of defining variables are available:

- Define them when you define a variable table in the way described here. This is the recommended way.
- Define them individually with the **composer $parm** command in the [*tablename.*]*variablename* “*variablevalue*” format. If you do not specify a table name, the new variable is placed in the default variable table.

You are not forced to create variable tables to be able to create and use variables. You might never create a table and never use one explicitly. In any case, the scheduler provides a default table and every time you create or manage a variable without naming the table, it stores it or looks for it there.

You can define more than one variable with the same name but different value and place them in different tables. Using variable tables you assign different values to the same variable and therefore reuse the same variable in job definitions and when defining prompts and file dependencies. Variable tables can be assigned at run cycle, job stream, and workstation level.

Variable tables can be particularly useful in job definitions when a job definition is used as a template for a job that belongs to more than one job stream. For example, you can assign different values to the same variable and reuse the same job definition in different job streams.

Syntax

vartable *tablename*
[**description** “*description*”]
[**isdefault**]
members
[*variablename* “*variablevalue*”]
...
[*variablename* “*variablevalue*”]
end

Arguments

`vartable` *`tablename`*
The name of the variable table. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 80 characters.

`description` `"`*`tabledescription`*`"`
The description of the variable table. The text must be enclosed within double

quotes. The description can contain up to 120 alphanumeric characters. It cannot contain double quotes (") other than the enclosing ones, colon (:), semicolon (;), and ampersand (&).

isdefault
When specified, the table is the default table. You cannot mark more than one table as the default table. When you mark a variable table as the default variable table, the current variable table is no longer the default one. When migrating the database from a previous version, the product creates the default variable table with all the variables already defined.

| **members** *variablename* "*variablevalue*"
| The list of variables and their values separated by spaces. The name can
| contain up to 16 alphanumeric characters, including dashes (-) and underscores
| (_), and must start with a letter. The value can contain up to 72 alphanumeric
| characters. Values must be enclosed within double quotes.

Example

The following example shows a variable table and its contents.

```
VARTABLE TEST1
  MEMBERS
   DEVBATCH "DOMD\IMSBATCH\SAME"
   PARAM_01 "date"
   PARAM_02 "root"
   PARM_01 "PARM_001"
   PRPT_02 "PARM_002"
   PRPT_03 "PARM_003"
   PRPT_04 "PARM_004"
   PRPT_05 "PARM_005"
   SAME17 "test/for/variable âwith samename > variable/table"
   SLAV10 "/nfsdir/billingprod/crmb/MAESTRO_JOB/AG82STGGDWHSCART"
   SLAV11 "/nfsdir/billingprod/crmb/MAESTRO_JOB/AG82CDMGALLBCV"
   SLAV12 "/nfsdir/billingprod/crmb/MAESTRO_JOB/AG82CDMGRISCTRAF"
   SLAV13 "/opt/crm/DWH_OK/Businness_Copy_ok"
   SLAV14 "/opt/crm/DWH_OK/DW_Canc_Cust_Gior_ok_"
   TRIGGER "/usr/local/samejobtriggers"
   VFILE2 "testforvarwithsamename2.sh"
   VUSER2 "same_user2"
   WRAPPER "/usr/local/sbin/same/phi_job.ksh"
END
```

Security file considerations

From the standpoint of security file authorizations, permission to act on the variable entries contained in a variable table is dependent on the overall permission granted on the variable table, as shown in following table.

Table 38. Required access keyword on variable table in Security file (vartable object) and allowed actions.

Required security file access keyword on enclosing variable table	Allowed action on listed variable entries
Modify	Add
	Delete
	Modify
	Rename
Display	Display
Unlock	Unlock

Creating a variable table definition using the Dynamic Workload Console

To create a variable table definition in the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **Go**
3. In the Working List toolbar of the pop-up window that opens, click **New→Variable Table**
4. Specify your choices in the Properties - Variable Table panel.

Prompt definition

A prompt identifies a textual message that is displayed to the operator and halts processing of the job or job stream until an affirmative answer is replied (either manually by the operator or automatically by an event rule action). After the prompt is replied to, processing continues. You can use prompts as dependencies in jobs and job streams. You can use variables in prompts.

There are two types of prompts:

local or unnamed prompts
An unnamed prompt is a prompt defined within a job or job stream definition using the keyword **prompt**, it has no name assigned and is not defined as a scheduling object in the database therefore it cannot be used by other jobs or job streams.

global or named prompts
A global prompt is defined in the database as a scheduling object, it is identified by a unique name and it can be used by any job or job stream. Variables in global prompts are resolved always using the default variable table. This is because global prompt are used by all jobs and job streams so just one value must be used for variable resolution.

This section describes global prompts. For more information on local prompts refer to "Job" on page 636 and "Job stream definition" on page 183.

Note: Predefined or global prompt definitions are reset each time the **JnextPlan** job is run.

Syntax

$prompt
promptname "[**:** | **!**]*text*"

[*promptname* ...]

Arguments

promptname
Specifies the name of the prompt. The name can contain up to 8 alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter.

text

Provides the text of the prompt. The text of the prompt can contain up to two hundred alphanumeric characters. Based on the character preceding the text, the prompt can behave differently:

- If the text begins with a colon (:), the prompt is displayed, but no reply is required to continue processing.
- If the text begins with an exclamation mark (!), the prompt is displayed, but it is not recorded in the log file.

You can use one or more parameters as part or all of the text string for a prompt. If you use a parameter, the parameter string must be enclosed in carets (^). See "Variable and parameter definition" on page 175 for an example.

Note: Within local prompts, carets (^) not identifying a parameter, must be preceded by a backslash (\) to prevent them from causing errors in the prompt. Within global prompts, carets do not have to be preceded by a backslash.

You can include backslash n (**\n**) within the text to create a new line.

Examples

The following example defines three prompts:

```
$prompt
     prmt1 "ready for job4? (y/n)"
     prmt2 ":job4 launched"
     prmt3 "!continue?"
```

See also

To create a prompt definition in the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **Go**
3. In the Working List toolbar of the ensuing popup window, click **New→Prompt**
4. Specify your choices in the Properties - Prompt panel.

Resource definition

Resources represent physical or logical scheduling resources that can be used as dependencies for jobs and job streams.

Syntax

$resource
workstation#*resourcename* *units* [“*description*”]

[*workstation*#*resourcename* ...]

Arguments

workstation
Specifies the name of the workstation or workstation class on which the resource is used.

resourcename
: Specifies the name of the resource. The name can contain up to eight alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter.

units
: Specifies the number of available resource units. Values can be **0** through **1024**.

| *"description"*
: | Provides a description of the resource. The description can contain up to | 120 alphanumeric characters. It must be enclosed in double quotes.

The resource units involved in needs dependencies for a job or for a job stream remain busy until the job or job stream is completed (successfully or not). The resource units are released as soon as the job or job stream is completed.

When multiple jobs and job streams depend on the same resource, if not enough resource units are currently available for all of them it is assigned according to the job or job stream priority. The status of a job or job stream becomes READY as soon as all its dependencies are resolved. If the limit CPU set on the workstation does not allow it to run at the moment, it waits in READY state. The only exception to this behavior is when the job or job stream is GO or HI, in which case it starts regardless of the value set for limit CPU.

Examples

The following example defines four resources:

```
$resource
  ux1#tapes  3 "tape units"
  ux1#jobslots  24 "job slots"
  ux2#tapes  2 "tape units"
  ux2#jobslots  16 "job slots"
```

See also

To create a resource definition in the Tivoli Dynamic Workload Console:
1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **Go**
3. In the Working List toolbar of the ensuing popup window, click **New→Resource**
4. Specify your choices in the Properties - Resource panel.

Job stream definition

A job stream consists of a sequence of jobs to be run, together with times, priorities, and other dependencies that determine the order of processing.

A job stream begins with a **schedule** keyword followed by attributes and dependencies. The colon delimiter introduces the jobs invoked by the job stream. Each job has its own attributes and dependencies.

Syntax

schedule [*workstation*#]*jobstreamname*
  # *comment*
  [**validfrom** *date*]

```
    [timezone|tz tzname]
    [description "text"]
    [draft]
    [vartable table_name]
    [freedays calendarname [-sa] [-su]]
    [on [runcycle name]
        [validfrom date] [validto date]
        [description "text"]
        [vartable table_name]
       {date|day|calendar|request|"icalendar"} [,...]
        [fdignore|fdnext|fdprev]
        [({at time [+n day[s]] |
        schedtime time [+n day[s]]}
        [until time [+n day[s]] [onuntil action]]
        [deadline time [+n day[s]]])]]
    [,...]
    [except [runcycle name]
            [validfrom date] [validto date]
            [description "text"]
            {date|day|calendar|request|"icalendar"} [,...]
            [fdignore|fdnext|fdprev]
            [{(at time [+n day[s]])] |
            (schedtime time [+n day[s]])}]
    [,...]
    [{at time [timezone|tz tzname] [+n day[s]] |
    schedtime time [timezone|tz tzname] [+n day[s]]}]
    [until time [timezone|tz tzname] [+n day[s]] [onuntil action]]
    [deadline time [timezone|tz tzname] [+n day[s]]]
    [carryforward]
    [matching {previous|sameday|relative from [+ | -] time to [+ | -] time|
      from time [+ | -n day[s]] to time [+ n day[s]] [,...]}]
    [follows {[netagent::][workstation#]jobstreamname[.jobname |@] [previous|
      sameday|relative from [+|-] time to [+|-] time|
      from time [+|-n day[s]] to time [+|-n day[s]]
        ]} ] [,...]] [...]
    [keysched]
    [limit joblimit]
    [needs { [n] [workstation#]resourcename } [,...] ] [...]
    [opens { [workstation#]"filename" [ (qualifier) ] [,...] }]  [...]
    [priority number | hi | go]
    [prompt {promptname|"[:|!]text"} [,...] ] [...]
  :
job-statement
    # comment
    [{at time [timezone|tz tzname] [+n day[s]] |
    schedtime time [timezone|tz tzname] [+n day[s]]}][,...]
    [until time [timezone|tz tzname] [+n day[s]] [onuntil action]
    [deadline time [timezone|tz tzname] [+n day[s]]] [,...]
    [every rate]
    [follows {[netagent::][workstation#]jobstreamname{.jobname @} [previous|
      sameday|relative from [+|-] time to [+|-] time |
      from time [+|-n day[s]] to time [+|-n day[s]]
        ]} ] [,...]] [...]
    [confirmed]
    [critical]
    [keyjob]
```

[**needs** { [*n*] [*workstation***#**]*resourcename* } [,...]] [...]
[**opens** { [*workstation#*]"*filename*" [**(***qualifier***)**] [**,**...] }] [...]
[**priority** *number* | **hi** | **go**]
[**prompt** {*promptname*|"[**:**|**!**]*text*"} [,...]] [...]

[*job-statement*...]
end

Arguments

Table 39 contains a brief description of the job stream definition keywords. A detailed description of each scheduling keyword is provided in the next subsections.

Table 39. List of scheduling keywords

Keyword	Description	Page
at	Defines the earliest time a job stream or a job run can be launched. When defined in a run cycle specifies the earliest time a job or a job stream can be launched for that specific run cycle.	"at" on page 188
carryforward	Carries the job stream forward if it is not completed.	"carryforward" on page 190
comment	Includes comments in the definition of a job stream or in a job contained in the job stream.	"comment" on page 190
confirmed	Specifies that the completion of this job requires confirmation.	"confirmed" on page 191
critical	Specifies that the job is mission critical and must therefore be managed preferentially.	"critical" on page 191
deadline	Specifies the time within which a job or job stream should complete. When defined in a run cycle specifies the time within which a job or a job stream must complete in that specific run cycle.	"deadline" on page 191
description	Contains a description of the job stream. The maximum length of this field is 120 characters.	"description" on page 192
draft	Specifies that the plan generation process must ignore this job stream.	"draft" on page 193
end	Marks the end of a job stream.	"end" on page 193
every	Launches the job repeatedly at a specified rate.	"every" on page 193
except	Specifies dates that are exceptions to the **on** dates the job stream is selected to run.	"except" on page 196
fdignore | **fdnext** | **fdprev**	Specifies a rule that must be applied when the date selected for exclusion falls on a non-working day.	"except" on page 196
follows	Specifies jobs or job streams that must complete successfully before the job or the job stream that is being defined is launched.	"follows" on page 198

Table 39. List of scheduling keywords (continued)

Keyword	Description	Page
freedays	Specifies a freeday calendar for calculating *workdays* for the job stream. It can also set Saturdays and Sundays as *workdays*.	"freedays" on page 200
job statement	Defines a job and its dependencies.	"job statement" on page 202
keyjob	Marks a job as key in both the database and in the plan for monitoring by applications, such as IBM Tivoli Business Systems Manager or IBM Tivoli Enterprise Console®.	"keyjob" on page 203
keysched	Marks a job stream as key in both the database and in the plan for monitoring by applications, such as IBM Tivoli Business Systems Manager or IBM Tivoli Enterprise Console.	"keysched" on page 203
limit	Sets a limit on the number of jobs that can be launched concurrently from the job stream.	"limit" on page 204
matching	Defines the matching criteria used when a matching criteria is not specified in the follows specifications in the job stream definition or in the job definition within the job stream.	"matching" on page 204
needs	Defines the number of units of a resource required by the job or job stream before it can be launched. The highest number of resources the job stream can be dependent from is 1024.	"needs" on page 205
on	Defines the dates on which the job stream is selected to run.	"on" on page 206
opens	Defines files that must be accessible before the job or job stream is launched.	"opens" on page 211
onuntil	Specifies the action to take on a job or job stream whose until time has been reached.	"until" on page 218
priority	Defines the priority for a job or job stream.	"priority" on page 213
prompt	Defines prompts that must be replied to before the job or job stream is launched.	"prompt" on page 214
runcycle	Specifies a label with a friendly name for the run cycle	• "except" on page 196 • "on" on page 206
schedule	Assigns a name to the job stream.	"schedule" on page 216
schedtime	Specifies the time used to set the job stream in the time line within the plan to determine successors and predecessors.	"schedtime" on page 215
timezone | tz	Specifies the time zone to be used when computing the start time.	"timezone" on page 218

Table 39. List of scheduling keywords (continued)

Keyword	Description	Page
until	Defines a latest time a job or a job stream can be launched. When defined in a run cycle specifies the latest time a job or a job stream can be launched for that specific run cycle.	"until" on page 218
validfrom	Defines the date from which the job stream instance starts.	"validfrom/validto" on page 220
validto	Indicates the date on which the job stream instance ends.	"validfrom/validto" on page 220
vartable	Defines the variable table to be used by the job stream and the run cycle.	"vartable" on page 221

Note:

1. Job streams scheduled to run on workstations marked as *ignored* are not added to the production plan when the plan is created or extended.
2. Wrongly typed keywords used in job definitions lead to truncated job definitions stored in the database. In fact the wrong keyword is considered extraneous to the job definition and so it is interpreted as the job name of an additional job definition. Usually this misinterpretation causes also a syntax error or an inexistent job definition error for the additional job definition.

Time zone specification rules

You can specify a time zone at several keyword levels within a job stream definition; that is:

- For the whole job stream (inclusive of all its keyword specifications)
- At time restriction level (with the `at`, `deadline`, `schedtime`, and `until` keywords)
- For each included job statement

The following rules apply when resolving the time zones specified within a job stream definition:

- When you specify the time zone at job stream level, this applies to the time definitions of the run cycle (defined with the `on` keyword) as well as to those in the time restrictions.
- If you specify a time zone both at job stream level and at time restriction level, they must be the same. If you specify no time zone, either at job stream and time restriction levels, the time zone specified on the workstation is used.
- The time zone specified at job level can differ from the one specified at job stream level and overrides it. If you specify no time zone, either at job stream and job levels, the time zone specified on the workstation running the job is used.

Time restriction specification rules

Within a job stream definition you can specify time restrictions (with the `at`, `deadline`, `schedtime`, and `until` keywords) at both job stream and run cycle levels. When both are specified, the time restrictions specified at run cycle level override the ones specified at job stream level.

Examples

This is an example of job stream definition:

```
SCHEDULE M235062_99#SCHED_FIRST1 VALIDFROM 06/30/2005
ON RUNCYCLE SCHED1_PREDSIMPLE VALIDFROM 07/18/2005 "FREQ=DAILY;INTERVAL=1"
   ( AT 1010 )
ON RUNCYCLE SCHED1_PRED_SIMPLE VALIDFROM 07/18/2005 "FREQ=DAILY;INTERVAL=1"
CARRYFORWARD
PROMPT "parto o no?"
PRIORITY 55
:
M235062_99#JOBMDM
 PRIORITY 30
 NEEDS 16 M235062_99#JOBSLOTS
 PROMPT PRMT3

B236153_00#JOB_FTA
 FOLLOWS JOBMDM
END
```

See also

To create a job stream definition in the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **Go**
3. In the Working List toolbar of the pop-up window that opens, click **New→Job Stream**
4. Specify your choices in the Properties - Job Stream panel.

Job stream definition keyword details

This section describes the job stream definition keywords listed in table Table 39 on page 185.

at

Specifies a time dependency. If the **at** keyword is used, then the job or job stream cannot start before the time set with this keyword.

Syntax

at *time* [**timezone**|**tz** *tzname*][+*n* **day**[**s**]] [**absolute**|**abs**]

Arguments

time Specifies a time of day. Possible values can range from **0000** to **2359**.

tzname Specifies the time zone to be used when computing the start time. See Chapter 16, “Managing time zones,” on page 531 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

Note: If an **at** time and an **until** or **deadline** time are specified, the time zones must be the same.

n Specifies an offset in days from the scheduled start date and time.

absolute
Specifies that the start date is based on the calendar day rather than on the production day.

Comments

If an **at** time is not specified for a job or job stream, its launch time is determined by its dependencies and priority and its position in the preproduction plan is determined by the value assigned to the **schedtime** keyword. For more information about the **schedtime** keyword refer to "schedtime" on page 215.

If the run cycle and job stream start times are both defined, the run cycle start time takes precedence when the job stream is scheduled with **JNextPlan**. When the job stream is launched with the **submit** command, the run cycle start time is not used.

The time value in the **at** option is considered as follows:

- If the time value is less than the value set in the ***startOfDay*** global option, it is taken to be for the following day.
- If the time value is greater than the value set in the ***startOfDay*** global option, it is taken to be for the current day.

If the master domain manager of your network runs with the `enLegacyStartOfDayEvaluation` and `enTimeZone` options set to `yes` to convert the `startOfDay` time set on the master domain manager to the local time zone set on each workstation across the network, you must add the **absolute** keyword to make it work when you submit a job or a job stream.

If neither the **at** nor the **schedtime** keywords are specified in the job stream definition then, by default, the job or job stream instance is positioned in the plan at the time specified in the *startOfDay* global option.

Examples

The following examples assume that the Tivoli Workload Scheduler processing day starts at 6:00 a.m.

- The following job stream, selected on Tuesdays, is launched no sooner than 3:00 a.m. Wednesday morning. Its two jobs are launched as soon as possible after that time.

  ```
  schedule sked7 on tu at 0300:
  job1
  job2
  end
  ```

- The following example launches job stream mysked on Sundays at 8:00 a.m.. Jobs job1, job2, and job3 are all launched on Sundays.

  ```
  schedule mysked on fr at 0800 + 2 days
  :
  job1
  job2 at 0900
  job3 follows job2 at 1200
  end
  ```

- The time zone of workstation `sfran` is defined as America/Los_Angeles, and the time zone of workstation `nycity` is defined as America/New_York. The following job stream is selected to run on Friday. It is launched on workstation `sfran` at 10:00 a.m. America/Los_Angeles Saturday. `job1` is launched on `sfran` as soon as possible after that time. `job2` is launched on `sfran` at 2:00 p.m. America/New_York (11:00 a.m. America/Los_Angeles) Saturday. `job3` is launched on workstation `nycity` at 4:00 p.m. America/New_York (1:00 p.m. America/Los_Angeles) Saturday.

```
sfran#schedule sked8 on fr at 1000 + 1 day:
job1
job2 at 1400 tz America/New_York
nycity#job3 at 1600
end
```

carryforward

Makes a job stream eligible to be carried forward to the next production plan if it is not completed before the end of the current production plan.

Syntax

carryforward

Examples

The following job stream is carried forward if its jobs have not completed before preproduction processing begins for a new production time frame.

```
schedule sked43 on th
carryforward
:
job12
job13
job13a
end
```

comment

Includes comments in a job stream definition and the jobs contained in a job stream.

Syntax

*text*

Comments

Inserts a comment line. The first character in the line must be a pound sign #.

You can add comments in a job stream definition immediately after the line with the **schedule** keyword, or in a job contained in a job stream definition immediately after the *job statement* line.

Examples

The following example includes both types of comments:

```
schedule wkend on fr at 1830
###########################
# The weekly cleanup jobs
###########################
#
carryforward
:
job1
 # final totals and reports
job2
 # update database
end
```

confirmed

Specifies that a job's completion must be confirmed by running a **conman confirm** command. See “confirm” on page 322 for more information.

Syntax

confirmed

Examples

In the following job stream, confirmation of the completion of `job1` must be received before `job2` and `job3` are launched.

```
schedule test1 on fr:
job1 confirmed
job2 follows job1
job3 follows job1
end
```

critical

Specifies that the job is mission-critical and must be processed accordingly.

A mission-critical job gets privileged treatment. Given its deadline and estimated duration, the scheduler:

- While building the plan, or every time it runs the **`submit`** command, calculates the latest start time each of its predecessors can start so that the job successfully meets its deadline. This is called the *critical start time*. The critical job and every one of its predecessors are assigned a critical start time.

  The entire set of predecessors to the critical job is referred to as the *critical network* of the job.
- While running the plan, dynamically recalculates the critical start times within the critical network.

  When a predecessor risks compromising the timely completion of the critical job, it is *promoted*; that is, using different operating system mechanisms, such as implementing the **`nice`** command on UNIX or changing the priority level on Windows, it is assigned additional resources and its submission is prioritized with respect to other jobs that are out of the critical network. This action is recurrently run on any predecessor within the critical network and, if necessary, on the critical job as long as there is a risk that this job becomes late.

Important: Critical jobs must have a deadline specified at job, job stream or run cycle level.

Syntax

critical

deadline

Specifies the time within which a job or job stream must complete. Jobs or job streams that have not yet started or that are still running when the deadline time is reached, are considered *late* in the plan. When a job (or job stream) is late, the following actions are performed:

- Job is shown as late in **conman**.

- An event is sent to the Tivoli Enterprise Console and the IBM Tivoli Business Systems Manager.
- A message is issued to the *stdlist* and console logs.

When a job does not complete before its deadline, a warning message is displayed. If this job is not part of a carried forward job stream and you run `JnextPlan` while it is still running, the job is inserted in USERJOBS. In this case, another warning message about the expired deadline is added in the *TWS_home*`/stdlist/logs/`*yyyymmdd*`_TWSMERGE.log file`.

Note: When using the **deadline** keyword, ensure the **bm check deadline** option is set to a value higher than 0 in the `localopts` configuration file on the workstation you are working on. You can define the **bm check deadline** option on each workstation on which you want to be aware of the deadline expiration, or, if you want to obtain up-to-date information about the whole environment, define the option on the master domain manager. Deadlines for critical jobs are evaluated automatically, independently of the **bm check deadline** option. For more information about the **bm check deadline** option, see Localopts details.

Syntax

deadline *time* [**timezone|tz** *tzname*][+*n* **day**[**s**] [**,...**]

Arguments

time
Specifies a time of day. Possible values range from **0000** to **2359**.

tzname
Specifies the time zone to be used when computing the deadline. See Chapter 16, "Managing time zones," on page 531 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

n
Specifies an offset in days from the scheduled deadline time.

Note: If a **deadline** time and an **until** or **at** time are specified, the time zones must be the same.

Examples

The following example launches job stream `sked7` every day and job `jobc` to start running at 14:30 and to be completed by 16:00.

```
schedule sked7 on everyday :
    jobc at 1430 deadline 1600
end
```

description

Includes a description for the job stream.

Syntax

description "*text*"

Comments

The maximum length of this field is 120 characters.

Examples

```
schedule test1
description "Revenue at the end of the month"
on monthend
:
job1
job2
job3
end
```

draft

Marks a job stream as draft. A draft job stream is not added to the preproduction plan.

Syntax

draft

Comments

A draft job stream is not considered when resolving dependencies and is not added to the production plan. After removing the draft keyword from a job stream you need to run **JnextPlan** command to add the job stream to the preproduction plan and so to the production plan.

Examples

```
schedule test1 on monthend
draft
:
job1
job2
job3
end
```

end

Marks the end of a job stream definition.

Syntax

end

Examples

```
schedule test1 on monthend
:
job1
job2
job3
end << end of job stream >>
```

every

Defines the repetition rate for a job. The job is launched repeatedly at the specified rate. If the job has a dependency that is not satisfied, the iteration is started only after the dependency is satisfied.

Syntax

every *rate*

Arguments

rate The repetition rate expressed in hours and minutes, in the *hhmm* format. The rate can be longer than 24 hours.

Comments

- The **every** iteration of a job does not stop even if one of the job repetitions abends.
- If the **every** option is used without the **at** dependency, the rerun jobs are scheduled respecting the **every** rate specified, starting from the time when the job actually started.
- In the specific case that the **every** option is used with the **at** dependency and one rerun is delayed (for a dependency or for any other reason), then, while Tivoli Workload Scheduler realigns to the **at** time, there might one or two iterations that do not respect the **every** rate. For all other cases the every rate is always respected.

  Example 2 explains how Tivoli Workload Scheduler realigns to the **at** time if the job starts later than the defined **at** time and some iterations are lost.
- If an **every** instance of a job does not start at its expected start time, use the **bm late every** option to set the maximum number of minutes that elapse before Tivoli Workload Scheduler skips the job. The value of the option must be defined in the `<TWSHOME>/localopts` file:

  bm late every = xx
  Where *xx* is the number of minutes.

  This option is local for each agent, therefore it must be defined on every fault-tolerant agent that has **every** jobs with **bm late every** option set.

  The **bm late every** option applies only to jobs with both the **every** option and the **at** time dependency defined, it has no impact on jobs that have only the **every** option defined. Only jobs whose **every** rate is greater than the **bm late every** value will be impacted.

  Example 4 on page 196 shows the behavior of Tivoli Workload Scheduler when the delay of an **every** instance does not exceed the **bm late every** option value.

  Example 5 on page 196 shows the behavior of Tivoli Workload Scheduler when the delay of an **every** instance exceeds the **bm late every** option value.

  Example 6 on page 196 shows the behavior of Tivoli Workload Scheduler when the first instance of a job does not run at its expected start time and exceeds the **bm late every** option value.
- If the **every** keyword is defined for a job when the Daylight Saving Time (DST) switches off, that is the clock is put one hour back, the command **every job** is DST aware and it runs also during the second, repeated time interval.

Examples

1. The following example runs the `testjob` job every hour:
   ```
   testjob every 100
   ```
2. The following example shows the `testjob1` job that is defined to run every 15 minutes, between the hours of 6:00 p.m. and 8:00 p.m.:
   ```
   testjob1 at 1800 every 15 until 2000
   ```

   The job is supposed to run at 1800, 1815, 1830, and so on every 15 minutes.

   If the job is submitted `adhoc` at 1833, the reruns are at 1833, 1834, 1845, etc. The reason for this is explained next:

   At first notice that in a job there are two time values to consider:

- The *start_time*; this is the time when the job is expected to run. It is set to the **at** time specified for the job or to the time when the rerun should be launched. This value can be viewed using `conman showjobs` before the job iteration starts.
- The *time_started*; this is the time when the job actually starts, for example 1833. This value can be viewed by using `conman showjobs` after the job iteration started.

Because `testjob1` was submitted `adhoc` at 1833, this is the information you see immediately after submission:

with `conman showjobs`

```
TESTJOB1 HOLD 1800
```

in the Symphony file

`start_time=1800` (because the job is expected to run **at** 1800)

`time_started=NULL` (because the job has not yet started)

Since the `start_time` (1800) is smaller than the current time (1833), `testjob1` starts immediately and the updated information becomes:

with `conman showjobs`

```
TESTJOB1 SUCC 1833
```

in the Symphony file

`start_time=1800` (because the job was expected to run **at** 1800)

`time_started=1833` (because the job started at 1833)

When **batchman** calculates the time for the next iteration, it uses the following data:

```
start_time=1800
rate=0015
current_time=1833
```

Since the next iteration time (1800+0015=1815) would still be sooner than the *current_time* value (1833), **batchman** identifies the last planned iteration that was not run by adding to the *start_time* as many *every_rate* as possible without exceeding the *current_time*

1800 + 0015 + 0015 = **1830** < 1833

and then issues the command to run that iteration. Assuming that this iteration is run at 1834, the information, after the job starts, becomes the following:

with `conman showjobs`

```
TESTJOB1 SUCC 1834
```

in the Symphony file

`start_time=1830` (because that job iteration was expected to run **at** 1830)

`time_started=1834` (because that job iteration started at 1834)

After this job iteration completed, **batchman** calculates again the time the next iteration has to start using these updated values:

```
start_time=1830
rate=0015
current_time=1834
```

The fact that the next iteration time (1830+0015=1845) is later than the *current_time* value (1834), shows **batchman** that the iteration is recovered. The iteration time, starting from 1845 onwards, can now be realigned with the planned iteration times set in the job definition by the **at** and **every** keywords.

3. The following example does not start the `testjob2` job iteration until job `testjob1` has completed successfully:

   ```
   testjob2 every 15 follows testjob1
   ```

4. In the following example, the delay of an instance of an **every** job does not exceed the **bm late every** option value:

   ```
   bm late every = 10
   JOB  AT  1400 EVERY 0030
   ```

   This job is supposed to run at 1400, 1430, 1500, and so on every thirty minutes.

   If the server is down from 1435 to 1605, the instances at 1500, 1530, and 1600 do not run. At 1605, Tivoli Workload Scheduler restarts. When it analyses the Symphony file, it determines that the potential best time for the next **every** job instance is 1600. Tivoli Workload Scheduler checks if the potential best time (1600) exceeds the maximum allowed delay for an **every** job (10 minutes).

   In this case the delay has not exceeded the **bm late every** option, therefore Tivoli Workload Scheduler behaves as usual and creates the instance of the **every** job with start time set to 1600. The subsequent instances are at 1630, 1700 and so on, every thirty minutes.

5. In the following example, the delay of the instance of an **every** job exceeds the **bm late every** option value:

   ```
   bm late every = 10
   JOB  AT  1400 EVERY 00030
   ```

   This job is supposed to run at 1400, 1430, 1500, and so, on every thirty minutes.

   If the server is down from 1435 to 1620, the instances at 1500, 1530, and 1600 do not run. At 1620, Tivoli Workload Scheduler restarts. When it analyses the Symphony file, it determines that the potential best time for the next **every** job instance is 1600. Tivoli Workload Scheduler checks if the potential best time (1600) exceeds the maximum allowed delay for an **every** instance of a job (10 minutes).

   In this case the delay is greater that the **bm late every** option, therefore Tivoli Workload Scheduler applies the new behavior, it does not launch the instance of the **every** job at 1600 and it creates the instance of the **every** job with start time set to 1630.

6. The following example shows the behaviour of Tivoli Workload Scheduler when the first instance of a job does not run at its expected start time and exceeds the **bm late every** option value:

   ```
   bm late every = 10
   JOB  AT  1400 EVERY 00030
   ```

   This job is supposed to run at 1400, 1430, 1500, and so on, every thirty minutes.

   If the server is down from 1000 to 1415, the first instance of the job does not run. At 1415, Tivoli Workload Scheduler restarts. When it analyses the Symphony file, it determines that the first instance of this **every** job has not run. In this case Tivoli Workload Scheduler launches the job at 1415.

except

Defines the dates that are exceptions to the **on** dates of a job stream. See "on" on page 206 for more information.

Syntax

except [**runcycle** *name*]
    [**validfrom** *date*] [**validto** *date*]
    [**description** *"text"*]
    {*date* | *day* | *calendar* | *request* | *"icalendar"*}
    [**,...**]
    [**fdignore** | **fdnext** | **fdprev**]

Arguments

fdignore | **fdnext** | **fdprev**
: Specifies a rule that must be applied when the date selected for exclusion falls on a non-working day. It can be one of the following:

fdignore
: Do not exclude the date.

fdnext
: Exclude the nearest workday after the non-working day.

fdprev
: Exclude the nearest workday before the non-working day.

For an explanation about remaining keywords contained in the **except** syntax refer to "on" on page 206.

Comments

You can define multiple instances of the **except** keyword for the same job stream. Each instance is equivalent to a run cycle to which you can associate a freeday rule.

Multiple **except** instances must be consecutive within the job stream definition.

Each instance of the keyword can contain any of the values allowed by the **except** syntax.

Examples

The following example selects job stream `testskd2` to run every weekday except those days whose dates appear on calendars named `monthend` and `holidays`:

```
schedule testskd2 on weekdays
except monthend,holidays
```

The following example selects job stream `testskd3` to run every weekday except May 15, 2005 and May 23, 2005:

```
schedule testskd3 on weekdays
except 05/15/2005,05/23/2005
```

The following example selects job stream `testskd4` to run every day except two weekdays prior to any date appearing on a calendar named `monthend`:

```
schedule testskd4 on everyday
except monthend-2 weekdays
```

Select job stream `sked4` to run on Mondays, Tuesdays, and 2 weekdays prior to each date listed in the `monthend` calendar. If the run date is a non-working day, run the job stream on the nearest following workday. Do not run the job stream on Wednesdays.

```
schedule sked4
on mo
on tu, MONTHEND -2 weekdays fdnext
except we
```

Select job stream `testskd2` to run every weekday except for the days listed in `monthend`. If a date in `monthend` falls on a non-working day, exclude the nearest workday before it. In this example, the non-working days are Saturdays, Sundays, and all the dates listed in the default `holidays` calendar .

```
schedule testskd2
on weekdays
except MONTHEND fdprev
```

follows

Defines the other jobs and job streams that must complete successfully before a job or job stream is launched.

Comments

Use the following syntax for job streams:

[**follows** {[*netagent*::][*workstation*#]*jobstreamname*[**.***jobname* |**@**]

[**previous**|**sameday**|**relative from [+/-]** *time* **to [+/-]** *time*|**from** *time* [+/-*n* day[s]] **to** *time* [+/-*n* day[s]]

Use the following syntax for jobs:

[**follows** {[*netagent*::][*workstation*#]*jobstreamname*{.*jobname* | **@**}

[**previous**|**sameday**|**relative from [+/-]** *time* **to [+/-]** *time* | **from** *time* [+/-*n* day[s]] **to** *time* [+/-*n* day[s]]

Arguments

netagent
: The name of the network agent where the internetwork dependency is defined.

workstation
: The workstation on which the job or job stream that must have completed runs. The default is the same workstation as the dependent job or job stream.

  If a *workstation* is not specified with *netagent*, the default is the workstation to which the network agent is connected.

jobstreamname
: The name of the job stream that must have completed. For a job, the default is the same job stream as the dependent job.

time
: Specifies a time of day. Possible values range from **0000** to **2359**.

jobname
: The name of the job that must have completed. An `at` sign (@) can be used to indicate that all jobs in the job stream must complete successfully.

Comments

Dependency resolution criteria define how the job stream or job referenced by an external follows dependency is matched to a specific job stream or job instance in the plan. Because the plan allows the inclusion of multiple instances of the same job or job stream, you can identify the instance that resolves the external follows dependency based on the following resolution criteria:

Closest Preceding
The job or job stream instance that resolves the dependency is the closest preceding the instance that includes the dependency.

Same Day
The job or job stream instance that resolves the dependency is the closest one in time scheduled to start on the day when the instance that includes the dependency is scheduled to run.

Within a Relative Interval
The job or job stream instance that resolves the dependency is the closest one in a time interval of your choice, which is defined relatively to the scheduled start time of the dependent instance.

Within an Absolute Interval
The job or job stream instance that resolves the dependency is the closest one in a time interval of your choice. The time interval is not related to the scheduled start time of the dependent instance.

Regardless of which matching criteria are used, if multiple instances of potential predecessor job streams exist in the specified time interval, the rule used by the product to identify the correct predecessor instance is the following:
1. Tivoli Workload Scheduler searches for the closest instance that precedes the depending job or job stream start time. If such an instance exists, this is the predecessor instance.
2. If there is no preceding instance, Tivoli Workload Scheduler considers the correct predecessor instance as the closest instance that starts after the depending job or job stream start time.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *jobStreamName.workstationName.jobName* format.

When a job stream includes a job with a follows dependency that shares the same job stream name (for example, job stream `schedA` includes a job named `job6` that has a follows dependency on `schedA.job2`), the dependency is added to the plan as an *external* follows dependency. Since Version 8.3, unlike in previous versions, because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

For more information and examples on how external follows dependencies are resolved in the plan refer to “Managing external follows dependencies for jobs and job streams” on page 52.

Examples

The following example specifies to not launch job stream `skedc` until the closest preceding job stream instance `sked4` on workstation `site1` have completed successfully:

```
schedule skedc  on fr  follows site1#sked4 previous
```

The following example specifies to not launch job stream `skedc` until the job stream instance of `sked4` on workstation `site1` that run between 12:00 of 3 days before to 3:00 of the day after have completed successfully:

```
schedule skedc  on fr  follows site1#sked4  from 1200 –3 days to 0300 1 day
```

The following example specifies not to launch job stream `skedc` until job stream `sked4` on workstation `site1` and job `joba` in job stream `sked5` on workstation `site2` have completed successfully:

```
schedule skedc on fr
follows site1#sked4,site2#sked5.joba
```

Do not launch `sked6` until `jobx` in the job stream `skedx` on network agent `cluster4` has completed successfully:

```
sked6 follows cluster4::site4#skedx.jobx
```

The following example specifies not to launch `jobd` until `joba` in the same job stream, and `job3` in job stream `skeda` have completed successfully:

```
jobd follows joba,skeda.job3
```

freedays

Use **freedays** to specify the name of a non-working days calendar that lists the non-working days for your enterprise. If and how a job stream runs on these particular days is defined in a *freedays* rule during the run cycle setup. Tivoli Workload Scheduler uses this calendar as the base calendar for calculating *workdays* for the job stream.

The keyword affects only the scheduling of the job streams for which it is specified.

Syntax

freedays *Calendar_Name* **[-sa] [-su]**

Arguments

Calendar_Name
The name of the calendar that must be used as the non-working days calendar for the job stream. If *Calendar_Name* is not in the database, Tivoli Workload Scheduler issues a warning message when you save the job stream. If *Calendar_Name* is not in the database when the command **schedulr** runs, Tivoli Workload Scheduler issues an error message and uses the default calendar **holidays** in its place. Do not use the names of weekdays for the calendar names.

-sa Saturdays are *workdays*.

-su Sundays are *workdays*.

Comments

If you specify a non-working days calendar in the job stream definition, then the concept of *workdays* takes the following value: *workdays = everyday excluding saturday and sunday (unless you specified -sa or -su along with freedays) and excluding all the dates of Calendar_Name*

If you do not specify **freedays** in the job stream definition, then: *workdays = everyday excluding saturday and sunday and all the dates of the holidays calendar*

By default, *saturday* and *sunday* are considered as non-working days unless you specify the contrary by adding **-sa**, **-su** or both after *Calendar_Name*.

Examples

Select job stream sked2 to run on 01/01/2005 and on all workdays as long as they are not listed in the non-working days calendar named GERMHOL.

```
schedule sked2
freedays GERMHOL
on 01/01/2005, workdays
```

Select job stream sked3 to run two workdays before each date in the PAYCAL calendar. Workdays are every day from Monday to Saturday as long as they are not listed in the non-working days calendar named USAHOL.

```
schedule sked3
freedays USAHOL -sa
on PAYCAL -2 workdays
```

Select job stream sked3 on the dates listed in the APDATES calendar. If the selected date is a non-working day, do not run the job stream. In this example, Sundays and all the dates listed in the GERMHOL calendar are to be considered as non-working days. All days from Monday to Saturday, except for the dates listed in GERMHOL, are workdays.

```
schedule sked3
freedays GERMHOL -sa
on APDATES fdignore
```

Select job stream testsked3 to run every weekday except 5/15/2005 and 5/23/2006. If 5/23/2006 is a non-working day, do not exclude it. In this example, Saturdays, Sundays, and all the dates listed in GERMHOL are to be considered as non-working days. All days from Monday to Friday, except for the dates listed in GERMHOL, are workdays.

```
schedule testskd3
freedays GERMHOL
on weekdays
except 5/15/2005 fdignore
except 5/23/2006
```

Select job stream testsked4 to run every day except two weekdays prior to every date listed in the MONTHEND calendar. If the date to be excluded is a non-working day, do not exclude it, but exclude the nearest following workday. In this example, non-working days are all the dates listed in USAHOL, while workdays are all the days from Monday to Sunday that are not listed in USAHOL.

```
schedule testskd4
freedays USAHOL -sa -su
on everyday
except MONTHEND -2 weekdays fdnext
```

job statement

Jobs can be defined in the database independently (as described in "Job" on page 636), or as part of job streams. In either case, the changes are made in the database and do not affect the production plan until the start of a new production plan.

Syntax

To define a job as part of a job stream, use the following syntax inside the job stream definition:

[*workstation*#]*jobname* [**as** *newname*]
  {**scriptname** *filename* | **docommand** "*command*"}
  streamlogon *username*
  [**description** "*description*"]
  [**tasktype** *tasktype*]
  [**interactive**]
  [**rccondsucc** "*Success Condition*"]
  [**recovery**
    {**stop** | **continue** | **rerun**}
    [**after** [*workstation*#]*jobname*]
    [**abendprompt** "*text*"]]

To use a job already defined in the database in the job stream definition define *job statement* using the following syntax:

[*workstation*#]*jobname* [**as** *newname*]

Arguments

as The name you want to use to refer to the job instance within that job stream.

For the other keywords refer to "Job" on page 636.

Comments

When defining a job as part of a job stream as the job stream definition is added to the database also the new job definition is added and can be referenced, from that moment on, also from other job streams.

Note: Wrongly typed keywords used in job definitions lead to truncated job definitions stored in the database. In fact the wrong keyword is considered extraneous to the job definition and so it is interpreted as the job name of an additional job definition. Usually this misinterpretation causes also a syntax error or an inexistent job definition error for the additional job definition.

When a job stream is added or modified, the attributes or recovery options of its jobs are also added or modified. Remember that when you add or replace a job stream, any job modifications affect all other job streams that use the jobs. Note that the cross reference report, *xref*, can be used to determine the names of the job streams including a specific job. For more information about cross reference report refer to "xref" on page 498.

Note: Jobs scheduled to run on workstations marked as *ignored*, and belonging to job streams scheduled to run on active workstations, are added to the plan even though they are not processed.

Examples

The following example defines a job stream with three previously defined jobs:

```
schedule bkup on fr at 20:00 :
  cpu1#jbk1
  cpu2#jbk2
    needs 1 tape
  cpu3#jbk3
    follows jbk1
end
```

The following job stream definition contains job statements that add or modify the definitions of two jobs in the database:

```
schedule sked4 on mo :
  job1  scriptname "d:\apps\maestro\scripts\jcljob1"
    streamlogon jack
    recovery stop abendprompt "continue production"
  site1#job2  scriptname "d:\apps\maestro\scripts\jcljob2"
    streamlogon jack
    follows job1
end
```

keyjob

The **keyjob** keyword is used to mark a job as key in both the database and in the plan and for monitoring by applications, such as Tivoli Business Systems Manager or Tivoli Enterprise Console. See the *IBM Tivoli Workload Scheduler Integrating with Other Products* guide for information about enabling the key flag mechanism.

Syntax

keyjob

Examples

The following example

```
SCHEDULE cpu1#sched1
ON everyday
KEYSCHED
AT 0100
cpu1#myjob1 KEYJOB
END
```

keysched

The **keysched** keyword is used to mark a job stream as key in both the database and in the plan and for monitoring by applications, such as Tivoli Business Systems Manager. See the *IBM Tivoli Workload Scheduler Integrating with Other Products* guide for information about enabling the key flag mechanism.

Syntax

keysched

Examples

The following example :

```
SCHEDULE cpu1#sched1
ON everyday
KEYSCHED
AT 0100
cpu1#myjob1 KEYJOB
END
```

limit

The **limit** keyword limits the number of jobs that can run simultaneously in a job stream on the same CPU.

Syntax

limit *joblimit*

Arguments

joblimit
: Specifies the number of jobs that can be running at the same time in the job stream. Possible values are **0** through **1024**. If you specify **0**, you prevent all jobs from being launched, including the one with priority set to **GO** or **HI**.

Examples

The following example limits to five the number of jobs that can run simultaneously in job stream `sked2`:

```
schedule sked2 on fr
  limit 5 :
```

matching

Sets a default for the matching criteria to be used in all follows dependencies where a matching criteria is not set in the job stream definition or in the jobs contained in the job stream.

Syntax

matching {**previous** |**sameday** | **relative from [+/-]** *time* **to [+/-]** *time*

Arguments

For information about the keyword used with **matching** see the “follows” on page 198 keyword.

Examples

The following example shows the definition of job stream `SCHED2` that:

- Contains a `job1` that can be run today only if it was run yesterday.
- Needs the instance of job stream `SCHED1` running the same day to complete before running.

```
SCHEDULE PDIVITA1#SCHED2
ON RUNCYCLE RULE1 "FREQ=DAILY;"
ON RUNCYCLE CALENDAR2 CAL1
MATCHING PREVIOUS
FOLLOWS PDIVITA1#SCHED1.@ SAMEDAY
FOLLOWS PDIVITA1#SCHED2.JOB1
:
```

```
PDIVITA1#JOB1

PDIVITA1#JOB2
END
```

In this sample the external follows dependency from PDIVITA1#SCHED2.JOB1 inherits the matching criteria specified in the **matching** keyword.

needs

The **needs** keyword defines resources that must be available before a job or job stream is launched. You can use the **needs** keyword either in a job stream definition or in the definition of the contained jobs, not in both.

Syntax

needs [*n*] [*workstation***#**]*resourcename* [,...]

Arguments

n Specifies the number of resource units required. Possible values are **1** to **1024** for each **needs** statement. The default is 1.

workstation
Specifies the name of the workstation on which the resource is locally defined. If not specified, the default is the workstation where the dependent job or job stream runs. Resources can be used as dependencies only by jobs and job streams that run on the workstation where the resource is defined.

Due to the resources dependencies resolution mechanism, a resource dependency at job stream level can be considered 'local' (and then its use supported) rather than 'global', when both the job stream and all its jobs are defined on the same workstation as the resource.

However, a standard agent and its host can reference the same resources.

resourcename
Specifies the name of the resource.

Comments

A job or job stream can request a maximum of 1024 units of a resource in a **needs** statement. At run time, each **needs** statement is converted in *holders*, each holding a maximum of 32 units of a specific resource. Independently from the amount of available units of the resource, for a single resource there can be a maximum of 32 holders. If 32 holders are already defined for a resource, the next job or job stream waiting for that resource waits until a current holder terminates AND the needed amount of resource becomes available.

Examples

The following example prevents job stream sked3 from being launched until three units of cputime, and two units of tapes become available:

```
schedule sked3 on fr
  needs 3 cputime,2 tapes :
```

The jlimit resource has been defined with two available units. The following example allows no more than two jobs to run concurrently in job stream sked4:

```
schedule sked4 on mo,we,fr :
  joba needs 1 jlimit
  jobb needs 1 jlimit
  jobc needs 2 jlimit    <<runs alone>>
  jobd needs 1 jlimit
end
```

on

This is a job stream keyword that defines when and how often a job stream is selected to run. If omitted the job stream is not added to the preproduction plan. The **on** keyword must follow the **schedule** keyword. See "except" on page 196 for more information.

Syntax

on [**runcycle** *name*]

[**valid from** *date*] [**valid to** *date*]

[**description** "*text*"]

[**vartable** *table_name*]

{*date*|*day*|*calendar*|*request*|"*icalendar*"} [**,...**]

[**fdignore**|**fdnext**|**fdprev**]

Arguments

runcycle *name*
: Specifies a label with a friendly name for the run cycle specified in the following lines.

valid from *date* **... valid to** *date*
: Delimits the time frame during which the job stream is active, that is the job stream is added to the production plan. Note that the date specified as **valid to** value is not included in the run cycle, therefore on this date the job stream is not active.

description "*text***"**
: Contains a description of the run cycle.

vartable
: Specifies the name of the variable table to be used by the run cycle.

date
: Specifies a run cycle that runs on specific dates. The syntax used for this type is:

  yyyymmdd [,yyyymmdd][,...]For example, for a job stream that is scheduled to run on the 25th of May 2009 and on the 12th of June 2009 the value is:

  ```
  on
  20090525,20090612
  ```

day
: Specifies a run cycle that runs on specific days. The syntax used for this type is:

  {**mo**|**tu**|**we**|**th**|**fr**|**sa**|**su**}For example, for a job stream that is scheduled to run every Monday the value is:

  ```
  on
  mo
  ```

calendar

The dates specified in a calendar with this name. The calendar name can be followed by an offset in the following format:

{**+** | **-**}*n* {**day**[**s**] | **weekday**[**s**] | **workday**[**s**]}

Where:

n The number of days, weekdays, or workdays.

days Every day of the week.

weekdays
Every day of the week, except Saturday and Sunday.

workdays
Every day of the week, except for Saturdays and Sundays (unless otherwise specified with the **freedays** keyword) and for the dates marked either in a designated non-working days calendar or in the **holidays** calendar.

request Selects the job stream only when requested. This is used for job streams that are selected by name rather than date. To prevent a scheduled job stream from being selected for **JnextPlan**, change its definition to ON REQUEST.

Note: When attempting to run a job stream that contains "on request" times, consider that:
- "On request" always takes precedence over "at".
- "On request" never takes precedence over "on".

icalendar

Represents a standard used to specify a recurring rule that describes when a job stream runs.

The syntax used for run cycle with type *icalendar* is the following:

FREQ={DAYLY|WEEKLY|MONTHLY|YEARLY}

[;INTERVAL=[-]*n*]

[;{BYFREEDAY|BYWORKDAY|BYDAY=*weekday_list*|

BYMONTHDAY=*monthday_list***}]**

where the default value for keyword **INTERVAL** is 1.

Using *icalendar* you can specify that a job stream runs:

every *n* **days**
by using the following format:

FREQ=DAILY[;INTERVAL=*n*]

where the value set for **valid from** is the first day of the resulting dates.

For example, for a job stream that is scheduled to run daily the value is:

```
FREQ=DAILY
```

For a job stream that is scheduled to run every second day the value is:

```
FREQ=DAILY;INTERVAL=2
```

every free or work days
by using the following format:

FREQ=DAILY[;INTERVAL=*n*]

;BYFREEDAY|BYWORKDAY

For example, for a job stream that is scheduled to run every non-working day the value is:

```
FREQ=DAILY;BYFREEDAY
```

For a job stream that is scheduled to run every second workday the value is:

```
FREQ=DAILY;INTERVAL=2;BYWORKDAY
```

every *n* weeks on specific *weekdays*
by using the following format:

FREQ=WEEKLY[;INTERVAL=*n*]

;BYDAY=*weekday_list*

where the value set for *weekday_list* is

```
[SU][,MO][,TU][,WE][,TH][,FR][,SA]
```

For example, for a job stream that is scheduled to run every week on Friday and Saturday the value is:

```
FREQ=WEEKLY;BYDAY=FR,SA
```

For a job stream that is scheduled to run every three weeks on Friday the value is:

```
FREQ=WEEKLY;INTERVAL=3;BYDAY=FR
```

every *n* months on specific dates of the month
by using the following format:

FREQ=MONTHLY[;INTERVAL=*n*]

;BYMONTHDAY=*monthday_list*

where the value set for *monthday_list* is represented by a list of

```
[+number_of_day_from_beginning_of_month]
[-number_of_day_from_end_of_month]
[number_of_day_of_the_month]
```

For example, for a job stream that is scheduled to run monthly on the 27th day the value is:

```
FREQ=MONTHLY;BYMONTHDAY=27
```

For a job stream that is scheduled to run every six months on the 15th and on the last day of the month the value is:

```
FREQ=MONTHLY;INTERVAL=6;BYMONTHDAY=15,-1
```

every *n* months on specific days of specific weeks
by using the following format:

FREQ=MONTHLY[;INTERVAL=*n*]

;BYDAY=*day_of_m_week_list*

where the value set for *day_of_m_week_list* is represented by a list of

[+*number_of_week_from_beginning_of_month*]
[-*number_of_week_from_end_of_month*]
[*weekday*]

For example, for a job stream that is scheduled to run monthly on the first Monday and on the last Friday the value is:

```
FREQ=MONTHLY;BYDAY=1MO,-1FR
```

For a job stream that is scheduled to run every six months on the 2nd Tuesday the value is:

```
FREQ=MONTHLY;INTERVAL=6;BYDAY=2TU
```

every *n* years
by using the following format:

FREQ=YEARLY[;INTERVAL=*n*]

where the value set for **valid from** is the first day of the resulting dates.

For example, for a job stream that is scheduled to run yearly the value is:

```
FREQ=YEARLY
```

For a job stream that is scheduled to run every two years the value is:

```
FREQ=YEARLY;INTERVAL=2
```

fdignore|fdnext|fdprev
Indicates the rule to be applied if the date selected for running the job or job stream falls on a non-working day. The available settings are:

fdignore
Do not add the date.

fdnext Add the nearest workday after the non-working day.

fdprev
Add the nearest workday before the non-working day.

Comments

You can define multiple instances of the **on** keyword for the same job stream. Multiple **on** instances must be consecutive within the job stream definition. Each instance is equivalent to a run cycle to which you can associate a freeday rule.

Each instance of the keyword can contain any of the values allowed by the **on** syntax.

If the run cycle and job stream start times are both defined, the run cycle start time takes precedence when the job stream is scheduled with **`JNextPlan`**. When the job stream is launched with the **`submit`** command, the run cycle start time is not used.

Examples

The following example selects job stream `sked1` on Mondays and Wednesdays:

```
schedule sked1 on mo,we
```

The following example selects job stream `sked3` on June 15, 2008, and on the dates listed in the `apdates` calendar:

```
schedule sked3 on 6/15/08,apdates
```

The following example selects job stream sked4 two weekdays before each date appearing in the monthend calendar:

```
schedule sked4 on monthend -2 weekdays
```

The following example selects job stream testskd1 every weekday except on Wednesdays:

```
schedule testskd1 on weekdays
  except we
```

The following example selects job stream testskd3 every weekday except May 15, 2008 and May 24, 2008:

```
schedule testskd3 on weekdays
  except 05/16/2008,05/24/2008
```

The following example selects job stream testskd4 every day except two weekdays prior to any date appearing in a calendar named monthend:

```
schedule testskd4 on everyday
  except monthend -2 weekdays
```

Select job stream sked1 to run all Mondays, Fridays, and on 29/12/2009. If Mondays and 29/12/2009 are non-working days, run the job stream on the nearest following workday. If Fridays are non-working days, run the job stream on the nearest preceding day. In this example, the non-working days are Saturdays, Sundays, and all the dates listed in the default HOLIDAYS calendar. Workdays are all days from Monday to Friday if they are not listed in the HOLIDAYS calendar.

```
schedule sked1
on mo, 12/29/2009 fdnext
on fr fdprev
```

This example shows the output of the display command of job stream testcli defined to run on different run cycles on workstation site2:

```
display js=site2#testcli
```

obtained in 120-column format by setting *MAESTROCOLUMNS*=120 before accessing the **composer** command-line:

```
JobstreamName Workstation Draft Valid From  Valid To UpdatedBy UpdatedOn  LockedBy
------------- ----------- ----- ----------  -------- --------- ---------  --------
TESTCLI       SITE2       Y     08/25/2008 -        mdmDBE4   08/25/2008 mdmDBE4

SCHEDULE W5#TESTCLI VALID FROM 08/25/2008 TIMEZONE ACT
DESCRIPTION "Job stream with several run cycle settings."
DRAFT
ON RUNCYCLE M5 VALID FROM 08/25/2008
   DESCRIPTION "monthly"
   "FREQ=MONTHLY;INTERVAL=5;BYMONTHDAY=-3,1"
   ( AT 0000 )
ON RUNCYCLE W4 VALID FROM 08/25/2008
   DESCRIPTION "weekly"
   "FREQ=WEEKLY;INTERVAL=5;BYDAY=MO,WE"
   FDNEXT ( AT 0000 )
ON RUNCYCLE D3 VALID FROM 08/25/2008
   DESCRIPTION "daily"
   "FREQ=DAILY;INTERVAL=2"
   FDPREV ( AT 0000 )
ON RUNCYCLE C2 VALID FROM 08/25/2008
   DESCRIPTION "calendar"
   ITALY +2 DAYS
   ( AT 0000 )
ON RUNCYCLE M6 VALID FROM 08/25/2008
   DESCRIPTION "monthly"
   "FREQ=MONTHLY;INTERVAL=2;BYDAY=1MO,1TH,2WE"
   ( AT 0000 +2 DAYS )
ON RUNCYCLE Y7 VALID FROM 08/25/2008
   DESCRIPTION "yearly"
   "FREQ=YEARLY;INTERVAL=7"
   ( AT 0100 )
ON RUNCYCLE SS1 VALID FROM 08/25/2008
   08/10/2008,08/18/2008,08/20/2008,08/25/2008
   ( AT 0000 UNTIL 0000 +1 DAYS ONUNTIL SUPPR DEADLINE 0000 +2 DAYS )
EXCEPT RUNCYCLE S1 VALID FROM 08/25/2008
   DESCRIPTION "simple"
   08/26/2008,08/28/2008,08/30/2008,09/13/2008
   ( AT 0000 )

CARRYFORWARD
MATCHING SAMEDAY
FOLLOWS LAB235004#SROBY2.@
FOLLOWS X8#COPYOFJS2.RR
FOLLOWS XA15::TPA
KEYSCHED
LIMIT 22
PRIORITY 15
:
X8#PIPPO AS JOBTC
 CONFIRMED
 PRIORITY 13
 KEYJOB
 FOLLOWS W5#POPO.@
 FOLLOWS X8#JS2.F3
END

AWSBIA291I Total objects: 1
```

The calendar `ITALY` is a custom calendar defined in the database that sets the workdays and holidays of the calendar in use in Italy.

opens

Specifies files that must be available before a job or job stream can be launched.

Syntax

opens [*workstation*#]"*filename*" [**(***qualifier***)**] [**,**...]

Arguments

workstation
: Specifies the name of the workstation or workstation class on which the file exists. The default is the workstation or workstation class of the dependent job or job stream. If you use a workstation class, it must be the same as that of the job stream that includes this statement.

filename
: Specifies the name of the file, enclosed in quotation marks. You can use Tivoli Workload Scheduler parameters as part or all of the file name string. If you use a parameter, it must be enclosed in carets (^). Refer to “Variable and parameter definition” on page 175 for additional information and examples.

qualifier

Specifies a valid test condition. In UNIX, the qualifier is passed to a **test** command, which runs as **root** in bin/sh.

In Windows, the test function is performed as the Tivoli Workload Scheduler user.

The valid qualifiers are:

-d %p	True if the file exists and is a directory.
-e %p	True if the file exists.
-f %p	True if the file exists and is a regular file.
-r %p	True if the file exists and is readable.
-s %p	True if the file exists and its size is greater than zero.
-w %p	True if the file exists and is writable.
-a	Boolean operator AND.
-o	Boolean operator OR.

In both UNIX and Windows, the expression **%p**, is used to pass the value assigned to *filename* to the test function.

Entering **(notempty)** is the same as entering **(-s %p)**. If no qualifier is specified, the default is **(-f %p)**.

Comments

The combination of the *path of the file* and the *qualifiers* cannot exceed 120 characters, and the *name of the file* cannot exceed 28 characters.

Examples

The following example checks to see that file `c:\users\fred\datafiles\file88` on workstation `nt5` is available for read access before launching `ux2#sked6`:

```
schedule ux2#sked6 on tu opens nt5#"c:\users\fred\datafiles\file88"
```

The following example checks to see if three directories, `/john`, `/mary`, and `/roger`, exist under `/users` before launching job `jobr2`:

```
jobr2  opens "/users"(-d %p/john -a -d %p/mary -a -d %p/roger)
```

The following example checks to see if `cron` has created its FIFO file before launching job `job6`:

```
job6  opens "/usr/lib/cron/FIFO"(-p %p)
```

The following example checks to see that file `d:\work\john\execit1` on workstation `dev3` exists and is not empty, before running job `jobt2`:

```
jobt2  opens dev3#"d:\work\john\execit1"(notempty)
```

The following example checks to see that file `c:\tech\checker\startf` on workstation `nyc` exists, is not empty, and is writable, before running job `job77`:

```
job77  opens nyc#"C:\tech\checker\startf"(-s %p -a -w %p)
```

Security for test(1) Commands:
In UNIX, a special security feature prevents unauthorized use of other commands in the qualifier. For example, the file below contains a command in the qualifier:

```
/users/xpr/hp3000/send2(-n "`ls /users/xpr/hp3000/m*`" -o -r %p)
```

If the qualifier contains another command, the following checks are made:

- The local option ***jm no root*** must be set to no.
- In the security file, the user documenting the schedule or adding the Open Files dependency with a **conman adddep** command, must have submit access to a job with the following attributes:
  name=`cmdstest.fileeq`
  logon=root
  jcl=the path of the opens files
  cpu=the CPU on which the opens files reside

  Note that `cmdstest` and `fileeq` do not exist.

priority

Sets the priority of a job or job stream. By assigning a different priority to jobs or job streams you determine which one starts first, if the dependencies are solved.

Assuming the jobs and job streams are ready to be launched , if you set a priority for the job streams and for the jobs in the job streams:

- The job stream that starts first is the one with the highest priority.
- Among the jobs in the job stream with the highest priority, the job that starts first is the one with the highest priority.

Syntax

priority *number* | **hi** | **go**

Arguments

number
: Specifies the priority. Possible values are **0** through **99**. A priority of 0 prevents the job or job stream from launching.

hi
: Represents a value higher than any value that can be specified with a number. When set, the job or job stream is immediately launched as soon as it is free from all dependencies.

go
: Represents the highest priority that can be set. When set, the job or job stream is immediately launched as soon as it is free from all dependencies.

Comments

Jobs and job streams with **hi** or **go** priority levels are launched as soon as all their dependencies are resolved. In this case:

- Job streams override the cpu job limit.
- Jobs override the cpu job limit, but they override neither the schedule job limit nor the cpu job fence.

Examples

The following example shows the relationship between job stream and job priorities. The two job streams, `sked1` and `sked2` have the following definitions in the database:

```
schedule sked1 on tu
priority 50
:
job1 priority 15
job2 priority 10
end
schedule sked2 on tu
priority 10
:
joba priority 60
jobb priority 50
end
```

Since the job stream `sked1` has the highest priority then the jobs are launched in the following order: `job1`, `job2`, `joba`, `jobb`.

If, instead, the job stream priorities are the same, the jobs are launched in the following order: `joba`, `jobb`, `job1`, `job2`.

If `job2` has a dependency **A** and `job1` has a dependency **B** and the dependency **A** becomes solved (while **B** remains not solved) then `job2` starts before `job1` even though `job2` has a priority lower than the one set for `job1`.

prompt

Specifies prompts that must be answered affirmatively before a job or job stream is launched.

Syntax

prompt *promptname* [**,**...]

prompt "[**:** | **!**]*text*" [,...]

Arguments

promptname
: Specifies the name of a prompt in the database. You can specify more than one *promptname* separated by commas but you cannot mix under the same **prompt** keyword prompts defined in the database with literal prompts.

text
: Specifies a literal prompt as a text string enclosed in quotes ("). Multiple strings separated by backlash **n** (\n) can be used for long messages. If the string begins with a colon (:), the message is displayed but no reply is necessary. If the string begins with an exclamation mark (!), the message is displayed, but it is not recorded in the log file. You can include backslash **n** (\n) within the text for new lines.

  You can use one or more parameters as part or all of the text string. To use a parameter, place its name between carets (^). Refer to "Variable and parameter definition" on page 175 for additional information and examples.

  Note: Within a local prompt, when not specifying a parameter, carets (^) must be preceded by a backslash (\) or they cause errors in the prompt. Within global prompts, carets do not have to be preceded by a backslash.

Examples

The following example shows both literal and named prompts. The first prompt is a literal prompt that uses a parameter named `sys1`. When a single affirmative reply is received for the prompt named `apmsg`, the dependencies for both `job1` and `job2` are satisfied.

```
schedule sked3 on tu,th
  prompt "All ap users logged out of ^sys1^? (y/n)"
:
  job1 prompt apmsg
  job2 prompt apmsg
end
```

The following example defines a literal prompt that appears on more than one line. It is defined with backlash **n** (\n) at the end of each line:

```
schedule sked5 on fr
  prompt  "The jobs in this job stream consume \n
an enormous amount of cpu time.\n
Do you want to launch it now? (y/n)"
:
  j1
  j2 follows j1
end
```

schedtime

Represents the time when the job stream is positioned in the plan. The value assigned to **schedtime** does not represent a dependency for the job stream. While the production plan is in process, the job or job stream instance might start processing before the time set in the **schedtime** keyword if all its dependencies are resolved and if its priority allows it to start.

Syntax

schedtime *time* [**timezone|tz** *tzname*][+*n* **day**[**s**]] [**,...**]

Arguments

time Specifies a time of day in the format: HHHHmm. Possible values are from **0000** to **320000**.

tzname Specifies the time zone to be used when calculating the start time. See Chapter 16, “Managing time zones,” on page 531 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

n Specifies an offset in days from the scheduled start date and time.

Comments

Differently from the **at** key, the **schedtime** key does not represent a time dependency, that is it does not state a time before which a job or job stream cannot start. Instead, the value specified in the **schedtime** keyword is used only to position the specific job or job stream instance in the preproduction plan. While the production plan is in process, the job or job stream instance might start processing before the time set in the **schedtime** keyword if all its dependencies are resolved and if its priority allows it to start.

For an explanation on how the **schedtime** keyword is used to identify predecessors in the preproduction plan, refer to "Managing external follows dependencies for jobs and job streams" on page 52.

The **at** and **schedtime** keywords are mutually exclusive. If **schedtime** is not specified and the **at** keyword is specified in the job or job stream, then its value is used to position the instance in the preproduction plan.

If neither the **at** nor the **schedtime** keywords are specified in the job or job stream definition, it is assumed by default to be the value assigned to the *startOfDay* global option set on the master domain manager.

For job streams with a **schedtime** definition, the value of the `Start time` field displayed on the Tivoli Dynamic Workload Console depends on the setting of the `enPreventStart` global option (which determines if job streams without an `at` dependency can start immediately, without waiting for the run cycle specified in the job stream):

- If `enPreventStart` is set to `yes`, the start time is 12:00 AM converted to the time zone specified on the graphical user interface.
- If `enPreventStart` is set to `no`, the start time field is blank.

Examples

The following examples assume that the Tivoli Workload Scheduler processing day starts at 6:00 a.m.

- The following job stream, selected on Tuesdays, is scheduled to start at 3:00 a.m. on Wednesday morning. Its two jobs are launched as soon as possible after the job stream starts processing.

  ```
  schedule sked7 on tu schedtime 0300:
  job1
  job2
  end
  ```

- The time zone of workstation `sfran` is defined as America/Los_Angeles, and the time zone of workstation `nycity` is defined as America/New_York. Job stream `sked8` is selected to run on Friday. It is scheduled to start on workstation `sfran` at 10:00 a.m. America/Los_Angeles Saturday (as specified by the `+ 1 day` offset). Job `job1` is launched on `sfran` as soon as possible after the job stream starts processing. Job `job2` is launched on `sfran` at 2:00 p.m. America/New_York (11:00 a.m. America/Los_Angeles) Saturday. `job3` is launched on workstation `nycity` at 4:00 p.m. America/New_York (1:00 p.m. America/Los_Angeles) Saturday.

  ```
  sfran#schedule sked8 on fr schedtime 1000 + 1 day:
  job1
  job2 at 1400 tz America/New_York
  nycity#job3 at 1600
  end
  ```

schedule

Specifies the job stream name. With the exception of comments, this must be the first keyword in a job stream, and must be followed by the **on** keyword.

Syntax

schedule [*workstation***#**]*jstreamname*

[**timezone|tz** *tzname*]

Arguments

workstation
: Specifies the name of the workstation on which the job stream is launched. The default is the workstation on which **composer** runs to add the job stream.

jstreamname
: Specifies the name of the job stream. The name must start with a letter, and can contain alphanumeric characters, dashes, and underscores. It can contain up to 16 characters.

timezone|tz *tzname*
: Specifies the time zone to be used when managing for the job stream. This setting is ignored if the global option *enTimeZone* is set to **no** on the master domain manager. For information on time zone settings, refer to Chapter 16, "Managing time zones," on page 531.

Comments

In a job stream definition you can set a time zone for the entire job stream by using the **timezone** keyword in the validity interval or when specifying time restrictions using **at**, **until**, or **deadline**.

You can set also a time zone for a job contained in a job stream by setting keywords **at**, **until**, or **deadline** for that job.

Regardless of whether you are defining a job or a job stream, if you use a time zone in a time restriction, for example **at** then you must use the same time zone when specifying the other time restrictions, such as **deadline** and **until**.

In a job stream definition you can set a time zone for the entire job stream and for the jobs it contains. These time zones can differ from each other, in which case the time zone set for the job is converted into the time zone set for the job stream.

To manage all possible time zone settings, the time zone conversion that is performed when processing jobs and job streams across the Tivoli Workload Scheduler network is made respecting the following criteria:

1. If a time zone is not set for a job within a job stream, then that job inherits the time zone set on the workstation where the job is planned to run.
2. If a time zone is not set for a job stream, then the time zone set is the one set on the workstation where the job stream is planned to run.
3. If none of the mentioned time zones is set, then the time zone used is the one set on the master domain manager.

Examples

This is the definition of e time zone of workstation `sfran` defined on workstation `sfran` on which is set the time zone `America/New_York`. The time zone set for `job2` to run of workstation LA is defined as America/Los_Angeles.

```
schedule sfran#sked8
tz America/New_York
on fr at 1000 + 1 day:
job1
LA#job2 at 1400 tz America/Los_Angeles
end
```

timezone

Specifies-at job stream level-the time zone used to calculate the time when the job stream must start processing.

Syntax

timezone|tz *tzname*

Arguments

tzname Specifies the name of the time zone. See Chapter 16, "Managing time zones," on page 531 for time zone names.

Comments

The time zone specified at job stream level applies to the time definitions for the run cycles and the time restrictions (defined by the `at`, `deadline`, `schedtime`, and `until` keywords).

If you specify a time zone for the job stream and one for a time restriction keyword, they must be the same.

If you specify no time zone, either at job stream and time restriction levels, the time zone specified for the workstation is used.

until

Depending on the object definition the until keyword belongs to, specifies the latest time a job stream must be completed or the latest time a job can be launched.

Syntax

until *time* [**timezone|tz** *tzname*][**+***n* **day**[**s**]][**absolute|abs**][**onuntil** *action*]

Arguments

time Specifies the time of day. The possible values are **0000** through **2359**.

tzname Specifies the time zone to be used when computing the time. See Chapter 16, "Managing time zones," on page 531 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

Note: If an **until** time and an **at** or **deadline** time are specified, the time zones must be the same.

n Specifies an offset, in days, from the scheduled date and time.

absolute
Specifies that the **until** date is based on the calendar day rather than on the production day.

onuntil *action*
Depending on the object definition the until keyword belongs to, specifies:

- The action to be taken on a job whose until time has expired but the job has not yet started.
- The action to be taken on a job stream whose until time has expired but the job stream is not yet completed in SUCC state.

The following are the possible values of the *action* parameter:

suppr The job or job stream and any dependent job or job stream do not run. This is the default behavior.

Once the until time expired on a job stream, the status for the job stream is calculated following the usual rules; suppressed jobs are not considered in the calculation. In case the job stream contains at least one **every** job its status is HOLD.

When the until time expires for a job, the job moves to HOLD status or keeps any previous status which is a final status.

If the **`until`** time is passed together with the **onuntil suppr** and the **carryforward** options, the job stream is *carry forwarded* by **`JnextPlan`** only if the **`until`** date is equal to the date when JnextPlan runs. If the **`until`** and the JnextPlan run dates are not the same, the job stream is not *carry forwarded*.

cont The job or job stream runs when all necessary conditions are met and a notification message is written to the log when the until time elapses.

If the **`until`** time is passed together with the **onuntil cont** and the **carryforward** options, the job stream is always *carry forwarded* by **`JnextPlan`**.

canc A job or job stream is cancelled when the **until** time specified expires. When using *onuntil canc* on jobs, the cancel operation on the job is issued by the FTA on which the job runs. Any job or job stream that was dependent on the completion of a job or job stream that was cancelled, runs because the dependency no longer exists.

If the **`until`** time is passed together with the **onuntil canc** and the **carryforward** options, the job stream is not *carry forwarded* by **`JnextPlan`** because it is already canceled.

Note: When using *onuntil canc* at job stream level, define as owner of the job stream the workstation highest in the hierarchy of the scheduling environment, among all workstations that own jobs contained in the job stream.

Note: Both the keyword **until** and **deadline** can be used in the same definition but they must be expressed using the same time zone setting.

Examples

The following example prevents `sked1` from launching after 5:00 p.m. on Tuesdays:

```
schedule sked1  on tu  until 1700 :
```

The following example launches `sked1` at 5:00 p.m., when its **until** time is reached:

```
schedule sked1 until 1700 onuntil cont
```

The following example launches `job1` between 1:00 p.m. and 5:00 p.m. on weekdays:

```
schedule sked2  on weekdays :
  job1  at 1300  until 1700
end
```

The following example launches joba every 15 minutes between 10:30 p.m. and 11:30 p.m. on Mondays:

```
schedule sked3 on mo :
  joba  at 2230  every 0015  until 2330
end
```

The following example launches job stream sked4 on Sundays between 8:00 a.m. and 1:00 p.m. The jobs are to be launched within this interval:

```
schedule sked4  on fr  at 0800 + 2 days
  until 1300 + 2 days
:
  job1
  job2  at 0900  <<launched on sunday>>
  job3  follows job2  at 1200  <<launched on sunday>>
end
```

The following example launches job stream sked8 on weekdays at 4:00 p.m. and should complete running by 5 p.m. If the job stream is not completed by 5 p.m., it is considered a late job stream. The jobs are to be launched as follows: job1 runs at 4 p.m., or at the latest, 4:20 p.m., at which time, if job1 has not yet started, a notification message is written to the log and it starts running. job2 runs at 4:30 p.m. or at the latest 4:50 p.m., at which time, if job2 has not yet started, it is cancelled.

```
schedule sked8 on weekdays at 1600 deadline 1700 :
   job1  at 1600 until 1620 onuntil cont
   job2 at 1630 until 1650 onuntil canc
end
```

The following example launches job stream sked01. When the **until** event occurs, the job stream sked02 is run because the job stream sked01 is placed in SUCC state. The job stream sked03, instead, is not run because it has a punctual time dependency on job job01 and this dependency has not been released.

```
SCHEDULE sked01 on everyday:
job01 until   2035 onuntil suppr
end
```

```
SCHEDULE sked02 on everyday follows sked01.@
:
job02
end
```

```
SCHEDULE sked03 on everyday follows  sked01.job01
:
job03
END
```

validfrom/validto

You can set a validity time for a job stream, which is a time frame within which the job stream is included in the preproduction plan. The validity time is set using the **validfrom** key in the job stream definition.

Syntax

validfrom *date*

Arguments

validfrom *date*
: Defines from which date the job stream is active, that is it must be included in a production plan if the production plan duration includes that date.

Comments

Different *versions* of the same job stream can be defined by creating different job streams with the same name and workstation, but having different validity intervals. The concept of versions of the same job stream sharing the same *jobstreamname* and the same *workstationname* are key when managing dependency on that job stream. In fact when you define an external follows dependencies on a job stream you identify the predecessor job stream using its *jobstreamname* and *workstationname*. The job stream identified as the dependency is the one whose validity interval is during the period the dependency is active.

If you change the *jobstreamname* or the *workstationname* in one version of the job stream, the change is propagated in all its versions.

If you lock a version of the job stream, all versions of that job stream are locked.

If you change the name of a job defined in a job stream version then the new job name is propagated in all versions of the job stream. This means that, if you modify something, other than the *jobstreamname* or the *workstationname*, the internal and external job stream associations remain consistent.

When defining a job stream version, you are only asked to enter the **validfrom** date, and the **validto** date is automatically set to the value of the **validfrom** date of the following version. The **validto** date is shown when issuing **list** and **display** command when *MAESTROCOLUMNS* is set to 120. Different versions of the same job stream continue to share the name and workstation fields after their creation. If you modify the name of a version or change the workstation on which it was defined, the change is applied to all versions of that job stream.

Note: If the keywords used in the job stream definition are **validfrom** and **validto**, the corresponding filtering keywords used when issuing commands against object definitions stored in the database are **validfrom** and **validto**. For more information refer to Chapter 9, "Managing objects in the database - composer," on page 235.

The date specified as **validto** value is not included in the run cycle, therefore the job stream is not active on this date.

vartable

Using variable tables you assign different values to the same variable and therefore reuse the same variable in job definitions and when defining prompts and file dependencies.

Syntax

vartable *tablename*

Arguments

`vartable` *`tablename`*
The name of the variable table. The name can contain up to 80 alphanumeric characters including dashes (-) and underscores (_), and must start with a letter.

Event rule definition

A scheduling event rule defines a set of actions that are to run upon the occurrence of specific event conditions. The definition of an event rule correlates events and triggers actions.

An event rule definition is identified by a rule name and by a set of attributes that specify if the rule is in draft state or not, the time interval it is active, the time frame of its validity, and other information required to decide when actions are triggered. It includes information related to the specific events (`eventCondition`) that the rule must detect and the specific actions it is to trigger upon their detection or timeout (`action`). Complex rules may include multiple events and multiple actions.

Refer to Chapter 7, "Running event-driven workload automation," on page 107 for an overview of scheduling event rules.

Syntax

You define event rules directly in XML language with the use of any XML editor. You can configure an environment variable on your computer to automatically open an XML editor of your choice to work with event rule definitions. See "The composer editor" on page 236 for details. The XML describing the event rule must match the rule language schemas defined in `EventRules.xsd` and in `FilteringPredicate.xsd`.

The rule language schemas defined in `eventRules.xsd` and in `FilteringPredicate.xsd` are used to validate your rule definitions and, depending upon the XML editor you have, to provide syntactic help. The files are located in the `schemas` subdirectory of the Tivoli Workload Scheduler installation directory.

The following is a list of all the language elements used for defining an event rule. Table 40 explains the meaning of the notation that follows each language element. *n* represents an unbounded number.

Table 40. Explanation of the notation defining the number of occurrences for a language element.

Notation	Meaning
(0, 1)	0 indicates that the language element is optional. 1 indicates that if coded, only 1 occurrence is allowed.
(0, n)	0 indicates that the language element is optional. n indicates that if coded, multiple occurrences are allowed.
(1, 1)	The first 1 indicates that the language element is required. The second 1 indicates that only 1 occurrence is allowed.
(1, 2)	1 indicates that the language element is required. 2 indicates that 2 occurrences are required.
(1, n)	1 indicates that the language element is required. n indicates that multiple occurrences are allowed.

- eventRule name=" " ruleType=" " isDraft=" " (1, 1)
  - description (0, 1)
  - timeZone (0, 1)
  - validity from=" " to=" " (0, 1)
  - activeTime start=" " end=" " (0, 1)

  - timeInterval amount=" " unit=" " (0, 1)
  - eventCondition eventProvider=" " eventType=" " (1, n)
    - scope (0, 1)
    - filteringPredicate (0, 1)
      - attributeFilter name=" " operator="eq" (0, n)
        - value (1, n)
      - attributeFilter name=" " operator="ne" (0, n)
        - value (1, n)
      - attributeFilter name=" " operator="le" (0, n)
        - value (1, 1)
      - attributeFilter name=" " operator="ge" (0, n)
        - value (1, 1)
      - attributeFilter name=" " operator="range" (0, 1)
        - value (1, 2)
  - correlationAttributes (0, 1)
    - attribute name=" " (1, n)
  - action actionProvider=" " actionType=" " responseType=" " (0, n)
    - description (0, 1)
    - scope (0, 1)
    - parameter name=" "(1, n)
    - value (1, 1)

Event rule definitions are grouped into event rule sets.
- eventRuleSet (1, 1)
  - eventRule (1, n)

Use the `eventRuleSet` language element also if you have to enclose a single rule definition.

Note that none of the comments you might write in the XML, in the form `<!--`*`text`*`-->`, are saved in the database. The next time you open a rule definition, the comments you wrote the first time will be gone. Use the `description` attribute to write any additional information instead.

Arguments

The keywords describing an event rule are the following XML tags:

eventRule

The scheduling object that encloses the definition of multiple event conditions and multiple rule actions in addition to a set of attributes that define when the rule is activated. An `eventRule` element typically includes:
- A number of required and optional rule attributes
- One or more event conditions
- One or more rule actions, although rules with no actions are also allowed

The rule attributes are:
- Required attributes:

  name The name of the event rule. It can be up to 40 alphanumeric characters in length, it must start with a letter, and cannot contain blanks. Underscore (`_`) and dash (`-`) characters are allowed.

ruleType

The rule type is based on the number of events - and on their correlation - that the rule is defined to detect. It can be one of the following:

filter The rule is activated upon the detection of a single specific event.

sequence

The rule is activated when an ordered sequence of events arrives within a specific time interval.

set The rule is activated when an unordered sequence of events arrives within a specific time interval.

Rules of type `set` and `sequence` can also be activated on **timeout**, when one or more events arrive but the complete sequence does not arrive within the specified time window.

isDraft

Specifies if the rule definition is currently enabled. Values can be **yes** or **no**. The default is **no**.

- Optional attributes:

description

A description of the rule. It can be of up to 120 characters.

Remember to write any XML special characters you might use in the XML special notation, such as:
– `&amp;` for &
– `&gt;` for >
– `&lt;` for <
– `&quot;` for "

and so on.

timeZone

Specifies a different time zone for the execution of the rule. The default time zone is the time zone defined on the workstation where the event processing server resides.

The application of daylight saving time (DST) has an impact on the `activeTime` interval (described next) of event rules:
– On the day that DST is turned on (the clock is adjusted forward one hour) the rule operation times that fall within the hour absorbed by the application of DST are moved forward by one hour. For example, 2:10 becomes 3:10.
– On the day that DST is turned off (the clock is adjusted backward one hour) the rule operation times that fall within the hour duplicated by the application of DST are regarded without DST.

validity

Specifies the rule validity period in terms of:

from *yyyy-mm-dd*

The validity period starts at midnight (of the rule time zone) of the specified day.

to *yyyy-mm-dd*

The validity period ends at midnight (of the rule time zone) of the specified day.

activeTime
Specifies the rule activity time window within each day of validity in terms of:

start *hh:mm:ss*
The beginning of the time window when the rule is active in hours, minutes, and seconds.

end *hh:mm:ss*
The end of the time window when the rule is active in hours, minutes, and seconds. If the time is earlier than in **start** *hh:mm:ss*, it refers to the next day.

timeInterval
Applies to rules that include timeout actions. Specifies the time interval within which all the events specified in the rule must have been received before a corrective timeout action is started. The time interval starts from the moment the first event specified in the rule is detected. Specify the time interval in terms of:

amount
The length of the time interval in time units.

unit The time unit in one of the following:
- hours
- seconds
- milliseconds

This attribute is mandatory when there are timeout actions in the event rule definition.

eventCondition
The event condition is made up by the following attributes:

- Required attributes:

eventProvider
Identifies the event monitoring provider that can capture a type of event. The event providers supplied at installation time are:

TWSObjectsMonitor
Monitors the status of Tivoli Workload Scheduler plan objects. This event provider runs on every Tivoli Workload Scheduler agent and sends the events to the event processing server.

TWSApplicationMonitor
Monitors Tivoli Workload Scheduler processes, file system, and message box.

FileMonitor
Monitors events affecting files.

eventType
Specifies the type of event that is to be monitored. Every event can be referred to an event provider. The following tables list the event types by event provider. To see the properties of each event type, go to Appendix A, "Event-driven workload automation event and action definitions," on page 575.

Table 41 on page 226 lists the `TWSObjectsMonitor` events.

Table 41. `TWSObjectsMonitor` events.

Event type	This event is sent when...
JobStatusChanged	the status of a job changes
JobUntil	the latest start time of a job has elapsed
JobSubmit	a job is submitted
JobCancel	a job is cancelled
JobRestart	a job is restarted
JobLate	a job becomes late
JobStreamStatusChanged	the status of a job stream changes
JobStreamCompleted	a job stream has completed
JobStreamUntil	the latest start time of a job stream has elapsed
JobStreamSubmit	a job stream is submitted
JobStreamCancel	a job stream is cancelled
JobStreamLate	a job stream becomes late
WorkstationStatusChanged	a workstation is started or stopped
ApplicationServerStatusChanged	the embedded WebSphere Application Server has stopped or is restarting
ChildWorkstationLinkChanged	the workstation defined in the event rule links or unlinks from its parent workstation (the parent workstation sends the event)
ParentWorkstationLinkChanged	the parent workstation links or unlinks from the workstation defined in the event rule (the child workstation sends the event)
PromptStatusChanged	a prompt is asked or replied
ProductAlert	The Symphony file in the workstation specified in the event rule has a corrupt record

Note: A rule with event type **`ParentWorkstationLinkChanged`** is not applicable when the Filters Workstation is set to agent, pool, dynamic pool, or remote engine and the **`ParentWorkstation`** attribute is set to broker. To monitor a link status change between the workload broker server and a workstation managed by the workload broker server, define a rule with event type equal to **`ChildWorkstationLinkChanged`**.

A rule with event type equal to **`ChildWorkstationLinkChanged`** works only when the broker workstation is linked, unlinked, stopped, or started. If the change in the link status is due to a stop or start operation on the agent workstation with the **`StartupLwa`** and **`ShutdownLwa`** commands, no action is started. To monitor stop or start operations on agent workstations, define a rule with event type equal to **`WorkstationStatusChanged`**.

Table 42 on page 227 lists the `TWSApplicationMonitor` events.

Table 42. `TWSApplicationMonitor` events.

Event type	This event is sent when...
MessageQueuesFilling	a specified mailbox exceeds the percentage full value.
TivoliWorkloadSchedulerFileSystemFilling	the file system where the Tivoli Workload Scheduler instance is installed exceeds the percentage full value.
TivoliWorkloadSchedulerProcessNotRunning	a specified process is not running.

Table 43 lists the `FileMonitor` events.

Table 43. `FileMonitor` events.

Event type	This event is sent when...
FileCreated	a file is created
FileDeleted	a file is deleted
ModificationCompleted	a file is modified (the event is sent only if two consecutive monitoring cycles have passed since the file was created or modified with no additional changes being detected)
LoggedMessageWritten	a specific string is found in a file (this event can be used to monitor application or system logs)

- Optional attributes:

  scope One or more qualifying attributes that describe the event. It can be of up to 120 characters. The scope is automatically generated from what is defined in the XML. It cannot be specified by users.

  filteringPredicate
  The filtering predicate sets the event conditions that are to be monitored for each event type. It is made up by:

  attributeFilter
  The attribute filter is a particular attribute of the event type that is to be monitored:

  – Is defined by the following elements:

    name The attribute, or property name, of the event type that is to be monitored. Refer to "Event providers and definitions" on page 575 for a list of property names for every event type.

    operator
    Can be:
    - `eq` (equal)
    - `ne` (not equal)
    - `ge` (equal or greater than)
    - `le` (equal or less than)
    - `range` (range)

  – Includes one or more:

    value The value on which the operator must be matched.

Note that every event type has a number of mandatory attributes, or property names. Not all the mandatory property names have default values. All the mandatory property names without a default value must have a filtering predicate defined.

correlationAttributes
The correlation attributes provide a way to direct the rule to create a separate rule copy for each group of events that share common characteristics. Typically, each active rule has one rule copy that runs in the event processing server. However, sometimes the same rule is needed for different groups of events, which are often related to different groups of resources. Using one or more correlation attributes is a method for directing a rule to create a separate rule copy for each group of events with common characteristics. Use with `set` and `sequence` rule types.

You can specify one or more attributes. Each is defined by:

attribute name=" "
The object attribute that you are correlating.

action The action that is to be triggered if the event is detected. Event rule definitions with events but no actions are syntactically accepted, although they may have no practical significance. You may save such rules as draft and add actions later before they are deployed.

- Is defined by the following required attributes:

  actionProvider
  The name of the action provider that can implement one or more configurable actions. The action providers available at installation time are:

  GenericAction
  Runs non-Tivoli Workload Scheduler commands. The commands are run on the same computer where the event processor runs.

  MailSender
  Connects to an SMTP server to send an email.

  MessageLogger
  Logs the occurrence of a situation in an internal auditing database.

  TECEventForwarder
  Forwards the event to an external TEC (Tivoli Enterprise Console) server (or any other application capable of listening to events in TEC format).

  TWSAction
  Runs one of the following `conman` commands:
  - submit job (`sbj`)
  - submit job stream (`sbs`)
  - submit command (`sbd`)
  - reply to prompt (`reply`)

  | **TWSForZosAction**
  | Adds an application occurrence (job stream) to the
  | current plan on Tivoli Workload Scheduler for z/OS.
  | This provider is for use in Tivoli Workload Scheduler
  | end-to-end scheduling configurations.

  | The application description of the occurrence to be
  | added must exist in the AD database of Tivoli Workload
  | Scheduler for z/OS.

actionType
Specifies the type of action that is to be triggered when a specified event is detected. Every action can be referred to an action provider. The following table lists the action types by action provider. To see the properties of each action type, go to Appendix A, "Event-driven workload automation event and action definitions," on page 575.

Table 44. Action types by action provider.

Action provider	Action types
GenericAction	RunCommand
MailSender	SendMail
MessageLogger	PostOperatorMessage
TECEventForwarder	TECFWD
TWSAction	reply (ReplyPrompt)
	sbd (SubmitAdHocJob)
	sbj (SubmitJob)
	sbs (SubmitJobStream)
TWSForZosAction	AddJobStream

responseType
Specifies when the action is to run. Values can be:
onDetection
The action starts as soon as all the events defined in the rule have been detected. Applies to all rule types. See also "Rule operation notes" on page 122.
onTimeOut
The action starts after the time specified in `timeInterval` has expired but not all the events defined in the rule have been received. Applies to `set` and `sequence` rules only.

Note that timeout actions are not run if you do not specify a time interval. The scheduler will however let you save event rules where timeout actions have been defined without specifying a time interval, because you could set the time interval at a later time. Until then, only actions with the `onDetection` response type are processed.

Timeout actions for which a time interval was not defined are run only when the rules are deactivated. An event rule is deactivated in either of two cases:
– The `planman deploy -scratch` command is issued
– The rule is modified (it is then deactivated as soon as the `planman deploy` command is run)

In either case the rule is first deactivated and then reactivated. At this time all pending actions are executed.

• Includes the following optional attributes:
description
A description of the action. It can be of up to 120 characters.

Remember to write any XML special characters you might use in the XML special notation, such as:

- `&amp;` for &
- `&gt;` for >
- `&lt;` for <
- `&quot;` for "

and so on.

scope One or more qualifying attributes that describe the action. It can be of up to 120 characters. The scope is automatically generated from what is defined in the XML. It cannot be specified by users.

- Includes a list of one or more parameters, or property names. All action types have at least one mandatory parameter. Every parameter is defined by:

  parameter name=" "
  See "Action providers and definitions" on page 586 for a list of parameters, or property names, available for every action type.

  value See "Action providers and definitions" on page 586 for a list of possible values or value types.

  You can use variable substitution. This means that when you define action parameters, you can use the property names of the events that trigger the event rule to replace the value of the action property name. To do this, write the value for the action parameter you intend to substitute in either of these two forms:

  - `${event.property}`

    Replaces the value as is. This is useful to pass the information to an action that works programmatically with that information, for example the `schedTime` of a job stream.
  - `%{event.property}`

    Replaces the value formatted in human readable format. This is useful to pass the information to an action that forwards that information to a user, for example to format the `schedTime` of a job stream in the body of an email.

  Where:

  event Is the name you set for the triggering `eventCondition`.

  property
  Is the `attributeFilter` name in the filtering predicate of the triggering event condition. The value taken by the attribute filter when the rule is triggered is replaced as a parameter value in the action definition before it is performed.

  Note that you can use variable substitution also if no `attributeFilter` was specified for an attribute and also if the attribute is read-only.

  For example, the task of an event rule is to detect when any of the jobs that have a name starting with `job15` end in error and, when that happens, submit that job again. The `eventCondition` section of the rule is coded as follows:

```
<eventCondition
      name="event1"
      eventProvider="TWSObjectsMonitor"
      eventType="JobStatusChanged">
  <filteringPredicate>
     <attributeFilter
          name="JobName"
          operator="eq">
       <value>job15*</value>
      </attributeFilter>
      <attributeFilter
            name="Workstation"
            operator="eq">
         <value>*</value>
      </attributeFilter>
      <attributeFilter
            name="Status"
            operator="eq">
         <value>Error</value>
      </attributeFilter>
   </filteringPredicate>
 </eventCondition>
```

Wild cards (* for multiple characters or ? for single characters) are used to generalize the desired event condition to all the job instances whose name starts with `job15` and to their associated workstation. Variable substitution is used in the `action` section to submit again the specific job that ended in error on the same workstation. The `action` section is:

```
<action
      actionProvider="TWSAction"
      actionType="sbj"
      responseType="onDetection">
  <description>Submit again the job that ended in error</description>
      <parameter name="JobDefinitionName">
        <value>${event1.JobName}</value>
      </parameter>
      <parameter name="JobDefinitionWorkstationName">
        <value>${event1.Workstation}</value>
      </parameter>
</action>
```

Examples

`JOB7` has a file dependency on `DAILYOPS.XLS`. As soon as the file is received, `JOB7` must start to process the file. The following rule controls that `JOB7` starts within one minute after the transfer of `DAILYOPS.XLS` is finished. If this does not happen, an email is sent to the evening operator. This is accomplished by defining two sequential event conditions that have to monitored:

1. The first event that triggers the rule is the creation of file `DAILYOPS.XLS` on the workstation to which it is to be transferred. As soon as this event is detected, a rule instance is created and a one minute interval count is begun to detect the next event condition.
2. The second event is the submission of `JOB7`. If this event fails to be detected within the specified time interval, the rule times out and the `SendMail` action is started.

```
<?xml version="1.0"?>
<eventRuleSet
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules
      EventRules.xsd">
   <eventRule
         name="sample_rule"
         ruleType="sequence"
         isDraft="no">
      <description>An email is sent if job JOB7 does not start within
                   a minute after file DAILYOPS.XLS is created</description>
      <timeZone>America/Indianapolis</timeZone>
      <validity
            from="2007-01-01"
            to="2007-12-31" />
      <activeTime
```

```
            start="20:00:00"
            end="22:00:00" />
      <timeInterval
            amount="60"
            unit="seconds" />
      <eventCondition
            name="DAILYOPS_FTPed_event"
            eventProvider="FileMonitor"
            eventType="FileCreated">
         <filteringPredicate>
            <attributeFilter
                  name="FileName"
                  operator="eq">
               <value>c:/dailybus/DAILYOPS.XLS</value>
            </attributeFilter>
            <attributeFilter
                  name="Workstation"
                  operator="eq">
               <value>ACCREC03</value>
            </attributeFilter>
            <attributeFilter
                  name="SampleInterval"
                  operator="eq">
               <value>300</value>
            </attributeFilter>
         </filteringPredicate>
      </eventCondition>
      <eventCondition
            name="job_JOB7_problem_event"
            eventProvider="TWSObjectsMonitor"
            eventType="JobSubmit">
        <filteringPredicate>
           <attributeFilter
                 name="JobStreamWorkstation"
                 operator="eq">
              <value>ACCREC03</value>
           </attributeFilter>
           <attributeFilter
                 name="Workstation"
                 operator="eq">
              <value>ACCREC03</value>
           </attributeFilter>
           <attributeFilter
                 name="JobStreamName"
                 operator="eq">
              <value>JSDAILY</value>
           </attributeFilter>
           <attributeFilter
                 name="JobName"
                 operator="eq">
              <value>JOB7</value>
           </attributeFilter>
        </filteringPredicate>
      </eventCondition>
      <action
            actionProvider="MailSender"
            actionType="SendMail"
            responseType="onTimeOut">
         <description>Send email to evening operator stating job did not
                     start</description>
         <parameter name="To">
            <value>eveoper@bigcorp.com</value>
         </parameter>
         <parameter name="Subject">
            <value>Job JOB7 failed to start within scheduled time on
                  workstation ACCREC03.</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

Note that the scope does not show the first time the rule is defined.

Go to "Event rule examples" on page 116 to find more event rule examples.

See also

To create an event rule definition in the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Design→Create Event Rules**
2. Select an engine name and click **Go**
3. Specify your choices in the Event Rule Editor panel.

Chapter 9. Managing objects in the database - composer

This chapter describes how you use the **composer** command-line program to manage scheduling objects in the Tivoli Workload Scheduler database. It is divided into the following sections:

- "Setting up the composer command-line program"
- "Running commands from composer" on page 239
- "Composer commands" on page 243

Setting up the composer command-line program

The **composer** command line program manages scheduling objects in database.

You must install the Tivoli Workload Scheduler Command Line Client feature on fault-tolerant agents and systems outside the Tivoli Workload Scheduler network to use the **composer** command-line program.

Setting up the composer environment

This section describes how you set up your composer environment.

Terminal output

The shell variables named ***MAESTROLINES*** and ***MAESTROCOLUMNS*** determine the output to your computer. If either variable is not set, the standard shell variables, ***LINES*** and ***COLUMNS***, are used. At the end of each screen page, composer prompts to continue. If ***MAESTROLINES*** (or ***LINES***) is set to zero or a negative number, composer does not pause at the end of a page.

Depending on the value set in the ***MAESTROCOLUMNS*** local variable, two different sets of information are displayed about the selected object. There are two possibilities:

- MAESTROCOLUMNS < 120 characters
- MAESTROCOLUMNS >= 120 characters

The value set in the ***MAESTROCOLUMNS*** local variable cannot be higher than 1024.

Refer to Table 51 on page 257 and Table 52 on page 267 to learn about the different output formats.

Offline output

The **;offline** option in composer commands is used to print the output of a command. When you include it, the following variables control the output:

Windows variables:

MAESTROLP
Specifies the file into which a command's output is written. The default is **stdout**.

MAESTROLPLINES
Specifies the number of lines per page. The default is 60.

MAESTROLPCOLUMNS
: Specifies the number of characters per line. The default is 132.

UNIX variables:
The **;offline** option in composer commands is used to print the output of a command. When you include it, the following shell variables control the output:

MAESTROLP
: Specifies the destination of a command's output. Set it to one of the following:

  > *file* Redirects output to a file, overwriting the contents of that file. If the file does not exist, it is created.

  >> *file* Redirects output to a file, appending the output to the end of the file. If the file does not exist, it is created.

  | *command*
  Pipes output to a system command or process. The system command is run whether or not output is generated.

  || *command*
  Pipes output to a system command or process. The system command is not run if there is no output.

  The default value for ***MAESTROLP*** is **| lp -tCONLIST** which directs the command output to the printer and places the title “CONLIST” in the banner page of the printout.

MAESTROLPLINES
: Specifies the number of lines per page. The default is **60**.

MAESTROLPCOLUMNS
: Specifies the number of characters per line. The default is **132**.

You must export the variables before you run **composer**.

The composer editor

Several composer commands automatically open a text editor. You can select which editor you want composer to use.

In addition, in both Windows and UNIX, you can set the `XMLEDIT` environment variable to point to an XML editor of your choice to edit event rule definitions. The XML editor opens automatically each time you run the `composer add`, `new`, or `modify` commands on an event rule.

Windows:
In Windows, **Notepad** is used as the default editor. To change the editor, set the *EDITOR* environment variable to the path and name of the new editor before you run **composer**.

UNIX:
Several commands you can issue from **composer** automatically open a text editor. The type of editor is determined by the value of two shell variables. If the variable ***VISUAL*** is set, it defines the editor, otherwise the variable ***EDITOR*** defines the editor. If neither of the variables is set, a **vi** editor is opened.

Selecting the composer prompt on UNIX

The **composer** command prompt is defined in the `*TWS_home*/localopts` file. The default command prompt is a dash (-). To select a different prompt, edit the

composer prompt option in the `localopts` file and change the dash. The prompt can be up to ten characters long, not including the required trailing pound sign (#):

```
#----------------------------------------------------------------------------
# Custom format attributes
#
date format =              1         # The possible values are 0-ymd, 1-mdy,
2-dmy, 3-NLS.
composer prompt =          -
conman prompt =            %
switch sym prompt =       <n>%
#----------------------------------------------------------------------------
```

For additional information about `localopts` configuration file, refer to *IBM Tivoli Workload Scheduler Administration Guide*.

Running the composer program

To configure your environment to use **composer**, set the *PATH* and *TWS_TISDIR* variables by running one of the following scripts:

In UNIX:

- `. ./`*`TWS_home`*`/`**`tws_env.sh`** for Bourne and Korn shells
- `. ./`*`TWS_home`*`/`**`tws_env.csh`** for C shells

In Windows:

- *`TWS_home`*`\`**`tws_env.cmd`**

Then use the following syntax to run commands from the **composer** user interface:

composer [-**file** *filename*][***connection_parameters***] [**"***command*[**&**[*command*]][...]**"**]

where:

-file *filename*
: Indicates an alternate custom properties file containing the settings for the connection parameters, used in place of the **`useropts`** and **`localopts`** files.

connection_parameters
: If you are using **composer** from the master domain manager, the connection parameters were configured at installation and do not need to be supplied, unless you do not want to use the default values.

  If you are using **composer** from the command line client on another workstation, the connection parameters might be supplied by one or more of these methods:

  - Stored in the `localopts` file
  - Stored in the `useropts` file
  - Supplied to the command in a parameter file
  - Supplied to the command as part of the command string

  For an overview of these options see "Setting up options for using the user interfaces" on page 45. For full details of the configuration parameters see the topic on configuring the command-line client access in the *Tivoli Workload Scheduler: Administration Guide*.

The **composer** command-line program is installed automatically when installing the master domain manager. It must be installed separately on top of a Tivoli Workload Scheduler agent workstation or stand-alone on a node outside the Tivoli Workload Scheduler network. The feature that installs the **composer** command-line program is named *Command Line Client*. For information on how to install the *Command Line Client* feature, refer to *IBM Tivoli Workload Scheduler: Planning and Installation Guide*.

Once installed, you can use the **composer** command line both in *batch* and in *interactive* mode.

When running **composer** in *interactive* mode, you first launch the **composer** command-line program and then, from the **composer** command line prompt, you run commands one at a time, for example:

```
composer –username  admin2  –password  admin2pwd
   add myjobs.txt
   create myjobs.txt from jobs=@
```

When running **composer** in *batch* mode, you launch the **composer** command-line program specifying as input parameter the command to be issued. When the command is processed, the **composer** command-line program exits, for example,

```
composer –f "c:\TWS\network\mylocalopts" add myjobs.txt
```

Note: If you use the batch mode to issue more than one command from within the **composer**, make sure you manage the ; character in one of the following ways:

- Using double-quotes, for example:
  ```
  composer "delete dom=old_domain; noask"
  ```
- Using a space character, for example:
  ```
  composer delete dom=old_domain noask
  ```
- Escaping the ; character, for example:
  ```
  composer delete dom=old_domain \; noask
  ```

Other examples on how to use the command, assuming connection parameters are set in the local configuration scripts, are the following:

- Runs **print** and **version** commands, and quits:
  ```
  composer "p parms&v"
  ```
- Runs **print** and **version** commands, and then prompts for a command:
  ```
  composer "p parms&v&"
  ```
- Reads commands from `cmdfile`:
  ```
  composer < cmdfile
  ```
- Pipes commands from `cmdfile` to **composer**:
  ```
  cat cmdfile | composer
  ```

Note: On Windows workstations, if the User Account Control (UAC) is turned on and the UAC exception list does not contain the `cmd.exe` file, you must open the DOS command prompt shell with the `"Run As Admnistrator"` option to run **composer** on your workstation as a generic user different from Administrator or Tivoli Workload Scheduler user.

Control characters

You can enter the following control characters in conversational mode to interrupt **composer** if your ***stty*** settings are configured to do so.

Ctrl+c **composer** stops running the current command at the next step that can be interrupted and returns a command prompt.

Ctrl+d **composer** quits after running the current command.

Running commands from composer

Composer commands consist of the following elements:

commandname selection arguments

where:

commandname
: Specifies the command name.

selection
: Specifies the object or set of objects to be acted upon.

arguments
: Specifies the command arguments.

Filters and wild cards

In Tivoli Workload Scheduler **composer** you can use wild cards and filters when issuing some specific commands to filter scheduling objects defined in the database. The wild cards you can use from **composer** are:

@	Replaces one or more alphanumeric characters.
?	Replaces one alphanumeric character.

To search for occurrences with names that contain either @ or ?, make sure you use the backslash character \ before @ or ? to escape them so that they are not interpreted as wild cards. Similarly, the backslash character must be prefixed by another backslash character to be interpreted as an occurrence to be found. The following examples clarify these rules, which also apply when specifying search strings using the **filter** keyword.

S@E	Search for all strings starting with S and ending with E, whatever is their length.
S?E	Search for all strings starting with S and ending with E, and whose length is three characters.
S\@E	Search for an exact match with string S@E.
S\?E	Search for an exact match with string S?E.
S\\E	Search for an exact match with string S\E.

The commands you can issue from composer and that support filtering are:
- display
- create
- delete
- list
- lock
- modify
- print
- unlock

The syntax used to filter objects when issuing one of these commands is the following:

command_name type_of_object=selection; [*option*;] [**filter** *filter_keyword*=selection [...]]

Table 45 shows the scheduling objects you can filter when issuing the commands listed above, and for each object, which fields can be filtered (in *italic*) or which key (in **bold)** is used to filter its fields:

Table 45. Scheduling objects filtering criteria

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
workstation	*workstationname*	Applies the command to the workstations whose name satisfies the criteria.	list ws=p@
	domain	Applies the command to the workstations which belong to a domain.	mod ws=@; filter **domain**=dom1
	vartable	Applies the command to the workstations which refer the specified variable table.	mod ws=@; filter **vartable**=table2
domain	*domainname*	Applies the command to the domains whose name satisfies the criteria.	display dom=dom?
	parent	Applies the command to the domains whose parent domain satisfies the criteria.	list dom=@; filter **parent**=rome
prompt	*promptname*	Applies the command to the global prompts whose name satisfies the criteria.	lock prompt=p@
user	*workstationname*# *username*	Applies the command to the users whose identifier satisfies the criteria.	list users=cpu1#operator?
resource	*workstationname*# *resourcename*	Applies the command to the resources whose identifier satisfies the criteria.	print res=cpu?#operator?
variable	*variablename*	Applies the command to the parameters whose name satisfies the criteria.	delete vb=mytable.myparm@
job definition	*jobname*	Applies the command to the job definitions whose name satisfies the criteria.	mod jd=mycpu#myjob@
	RecoveryJob	Applies the command to the jobs whose definition contains the specified recovery job definition.	list jobdefinition=@; filter **RecoveryJob**=CPUA#job01

Table 45. Scheduling objects filtering criteria (continued)

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
job stream	*workstationname# jobstreamname*	Applies the command to the job definitions whose name satisfies the criteria.	mod js=mycpu#myjs@
	Calendar or **FreeCalendar**	Applies the command to the job streams that contain the calendar or non-working days calendar specified in the filter.	list js=@#@; filter **Calendar**=cal1
	Jobdefinition	Applies the command to the job streams that contain the job definition specified in the filter.	list js=@#@; filter **jobdefinition**=CPUA#job01
	Resource	Applies the command to the job streams that refer to the resource specified in the filter.	list js=@#@; filter **Resource**=cpu1#disk1
	Prompt	Applies the command to the job streams that refer to the prompt specified in the filter.	list js=@#@; filter **Prompt**=myprompt
	Vartable	Applies the command to the job streams that refer to the variable table specified in the filter. The variable table can be specified either in the run cycle or in the job stream section.	list js=@#@; filter **Vartable**=table1
	Rcvartable	Applies the command to the run cycles in the job streams that refer to the variable table specified in the filter.	list js=@#@; filter **Rcvartable**=table1
	Jsvartable	Applies the command to the job streams that refer to the variable table specified in the filter regardless of what is specified in the run cycle.	list js=@#@; filter **Jsvartable**=table1
event rule	*eventrulename*	Applies the command to the event rules that include an action on a specific job or job stream.	list er=@; filter js=accrecjs5
vartable	*vartablename*	Applies the command to the variable tables whose name satisfies the criteria.	list vartable=A@
	isdefault	Applies the command to the default variable table.	list vartable=A@; filter isdefault

You can combine more than one filter for the same object type as shown in the following example:

```
list js=@#@; filter Calendar=cal1 FreeCalendar=VACATIONS jobdefinition=CPUA#job01
```

The output of the command is a list of job streams using calendar `cal1`, non-working days calendar `VACATIONS`, and containing a job with job definition `CPUA#job01`.

Delimeters and special characters

Table 46 lists characters have special meanings in composer commands.

Table 46. Delimeters and special characters for composer

Character	Description
&	Command delimiter. See "Running the composer program" on page 237.
;	Argument delimiter. For example: `;info;offline`
=	Value delimiter. For example: `sched=sked5`
: !	Command prefixes that pass the command on to the system. These prefixes are optional; if composer does not recognize the command, it is passed automatically to the system. For example: `!ls or :ls`
<< >>	Comment brackets. Comments can be placed on a single line anywhere in a job stream. For example: `schedule foo <<`*`comment`*`>> on everyday`

*	Comment prefix. When this prefix is the first character on a line, the entire line is a comment. When the prefix follows a command, the remainder of the line is a comment. For example: *`*comment`*
	`or`
	`print&` *`*comment`*

>	Redirects command output to a file, overwriting the contents of that file. If the file does not exist, it is created. For example: `display parms > parmlist`
>>	Redirects command output to a file and appends the output to the end of file. If the file does not exist, it is created. For example: `display parms >> parmlist`
|	Pipes command output to a system command or process. The system command is run whether or not output is generated. For example: `display parms | grep alparm`
||	Pipes command output to a system command or process. The system command is not run if there is no output. For example: `display parms || grep alparm`

Composer return codes

Composer return codes

Composer return codes management

When you run a composer command, the command line can show an outcome return code. You can find the return code performing the following action:

on Windows Operating systems

```
echo %ERRORLEVEL%
```

on UNIX Operating systems

```
echo $?
```

The following is a list of the return codes for composer command line:

0	command completed successfully.
4	command completed with a warning.
8	command completed with an error.
16	command fails.
32	command has a syntax error.

Composer commands

Table 47 lists the **composer** commands.

Note: Command names and keywords can be entered in either uppercase or lowercase characters, and can be abbreviated to as few leading characters as are needed to uniquely distinguish them from each other. Some of the command names also have short forms.

However there are some abbreviations, such as **v**, that point to a specific command, **version** in this case, even though they do not uniquely identify that command in the list. This happens when the abbreviation is hard coded in the product and so mismatches in invoking the wrong command are automatically bypassed.

Table 47. List of composer commands

Command	Short Name	Description	See page
add	**a**	Adds a scheduling objects definition to the database from a text file.	"add" on page 248
authenticate	**au**	Changes the credentials of the user running **composer**.	"authenticate" on page 250
continue	**c**	Ignores the next error.	"continue" on page 251
create	**cr**	Extracts an object definition from the database and writes it in a text file. Synonym for the **extract** command.	"extract" on page 260
delete	**de**	Deletes scheduling objects.	"delete" on page 251
display	**di**	Displays the details of the specified scheduling object.	"display" on page 255
edit	**ed**	Edits a file.	"edit" on page 259
exit	**e**	Exits **composer**.	"exit" on page 260
extract	**ext**	Extracts an object definition from the database and writes it in a text file.	"extract" on page 260
help	**h**	Invoke the help on line for a command.	"help" on page 264
list	**l**	Lists scheduling objects.	"list" on page 265

Table 47. List of composer commands (continued)

Command	Short Name	Description	See page
lock	**lo**	Locks the access to database objects.	"lock" on page 270
modify	**m**	Modifies scheduling objects.	"modify" on page 273
new		Creates a scheduling object using a text file where the object definition is inserted online. If you specify the type of scheduling object you want to define after *new* command, a predefined object definition is written in the text file.	"new" on page 278
print	**p**	Prints scheduling objects.	"display" on page 255
redo	**red**	Edits and reruns the previous command.	"redo" on page 279
rename	**rn**	Changes the object name.	"rename" on page 281
replace	**rep**	Replaces scheduling objects.	"replace" on page 283
system command		Invokes an operating system command.	"system command" on page 284
unlock	**u**	Releases lock on the scheduling object defined in the database.	"unlock" on page 284
validate	**val**	Validates the syntax, semantic, and data integrity of an object definition.	"validate" on page 288
version	**v**	Displays the **composer** command-line program banner.	"version" on page 288

Referential integrity check

Tivoli Workload Scheduler automatically performs referential checks to avoid lack of integrity in the object definitions in the database whenever you run commands that create, modify, or delete the definition of a referenced object. These are the checks performed by the product:

- Every time you use a command that creates a new object in the database, Tivoli Workload Scheduler checks that:
  - An object of the same type and with the same identifier does not already exist.
  - The objects referenced by this object already exist in the database.
- Every time you run a command that modifies an object definition in the database, Tivoli Workload Scheduler checks that:
  - The object definition to be modified exists in the database.
  - The objects referenced by this object exist in the database.
  - To avoid integrity inconsistencies, the object definition does not appear in the definition of an object belonging to the chain of his ancestors.
- Every time you run a command that deletes an object definition in the database, Tivoli Workload Scheduler checks that:
  - The object definition to be deleted exists in the database.
  - The object definition to be deleted is not referenced by other objects defined in the database.

Note that there is no referential integrity check for event rules.

Table 48 shows, for each object type, the identifiers that are used to uniquely identify the object in the database when creating or modifying object definitions:

Table 48. Object identifiers for each type of object defined in the database

Object type	**Object identifiers**
domain	*domainname*
workstation	*workstationname* (checked across workstations and workstation classes)
workstation class	*workstationclassname* (checked across workstations and workstation classes)
calendar	*calendarname*
job definition	*workstationname* and *jobname*
user	*workstationname* and *username*
job stream	*workstationname* and *jobstreamname* and, if defined, *validfrom*
job within a job stream	*workstationname* and *jobstreamname*, *jobname*, and, if defined, *validfrom*
resource	*workstationname* and *resourcename*
prompt	*promptname*
variable table	*variabletablename*
variable	*variabletablename.variablename*
event rule	*eventrulename*

In general, referential integrity prevents the deletion of objects when they are referenced by other objects in the database. However, in some cases where the deletion of an object (for example a workstation) implies only the update of a referencing object (for example a workstation class that includes it), the deletion might be allowed. Table 49 shows all cases when a referenced object can be deleted even if other objects reference it:

Table 49. Object definition update upon deletion of referenced object

Object	**References**	**Upon deletion of the referenced object ...**
Internetwork Dependency	Workstation	... remove the dependency from the job or job stream
External Follows Depend	Job Stream	... remove the dependency from the job or job stream
	Job	... remove the dependency from the job or job stream
Internal Dependency	Job	... remove the dependency from the job or job stream
Workstation Class	Workstation	... remove the workstation from the workstation class

Table 50 on page 246 describes how the product behaves when it is requested to delete an object referenced by another object with using a specific relationship:

Table 50. Referential integrity check when deleting an object from the database

Object to be deleted	Referenced by object	Relationship	Delete rule
domain A	domain B	domain A is parent of domain B	An error specifying the existing relationship is displayed.
	workstation B	workstation B belongs to domain A	An error specifying the existing relationship is displayed.
workstation A	workstation B	workstation A is host for workstation B	An error specifying the existing relationship is displayed.
	job B	job B is defined on workstation A	An error specifying the existing relationship is displayed.
	job stream B	job stream B is defined on workstation A	An error specifying the existing relationship is displayed.
	user B	user B is defined on workstation A	An error specifying the existing relationship is displayed.
	job stream B	workstation A works as network agent for internetwork dependencies set in job stream B	Both workstation A and the internetwork dependency are deleted
	job stream B	job stream B has a file dependency from a file defined on workstation A	Both workstation A and the file dependency are deleted
	job B within job stream B	workstation A works as network agent for internetwork dependencies set in job B	Both workstation A and the internetwork dependency are deleted
	job B within job stream B	job B has a file dependency from a file defined on workstation A	Both workstation A and the file dependency are deleted
	resource B	resource B is defined on workstation A	An error specifying the existing relationship is displayed.
	file B	file B is defined on workstation A	An error specifying the existing relationship is displayed.
	workstation class B	workstation A belongs to workstation class B	Both workstation A and its entry in workstation class B are deleted.
	job B within job stream B	job B contained in job stream B is defined on workstation A	An error specifying the existing relationship is displayed.

Table 50. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
job A	job B	job A is recovery job for job B	An error specifying the existing relationship is displayed.
	job stream B	job A is contained in job stream B	An error specifying the existing relationship is displayed.
	job stream B	job stream B follows job A	job A and the follows dependency in job stream B are deleted.
	job B within job stream B	job B follows job A	job A and the follows dependency in job B are deleted.
	event rule B	job A is in the action definition of event rule B (and does not use variable substitution)	An error specifying the existing relationship is displayed.
calendar A	job stream B	job stream B uses calendar A	An error specifying the existing relationship is displayed.
workstation class A	job B	job B is defined on workstation class A	An error specifying the existing relationship is displayed.
	job stream B	job stream B is defined on workstation class A	An error specifying the existing relationship is displayed.
	resource B	resource B is defined on workstation class A	An error specifying the existing relationship is displayed.
	file B	file B is defined on workstation class A	An error specifying the existing relationship is displayed.
resource A	job stream B	needs dependency defined in job stream B	An error specifying the existing relationship is displayed.
	job B within job stream B	needs dependency defined in job B	An error specifying the existing relationship is displayed.
prompt A	job stream B	prompt dependency defined in job stream B	An error specifying the existing relationship is displayed.
	job B within job stream B	prompt dependency defined in job B	An error specifying the existing relationship is displayed.

Table 50. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
variable A	job stream B	variable A is used in job stream B in: • in the text of an ad hoc prompt • or in the file name specified in a file dependency	variable A is deleted without checking
	job B	variable A is used in job stream B in: • in the text of an ad hoc prompt • or in the file name specified in a file dependency • or in the value specified for streamlogon • or in the value specified for scriptname	variable A is deleted without checking
	prompt B	variable A is used in the text of prompt B	variable A is deleted without checking
variable table A	job stream B	variable table A is referenced in job stream B	variable table A is not deleted
	job B	variable table A is referenced in job B	variable table A is not deleted
	prompt B	variable table A is referenced in the text of prompt B	variable table A is not deleted
job stream A	job stream B	job stream B follows job stream A	job stream A and the follows dependency in job stream B are deleted.
	job B within a job stream B	job B follows job stream A	job stream A and the follows dependency in job B are deleted.
	event rule B	job stream A is in the action definition of event rule B (and does not use variable substitution)	An error specifying the existing relationship is displayed.

add

Adds or updates scheduling objects to the database.

Authorization

You must have ***add*** access to add a new scheduling object. If the object already exists in the database you must have:

- ***modify*** access to the object if the object is not locked.
- ***modify*** and ***unlock*** access to the object if the object is locked by another user.

Syntax

{**add** | **a**} *filename* [**;unlock**]

Arguments

filename
: Specifies the name of the text file that contains the object definitions. For event rules, *filename* specifies the name of the XML file containing the definitions of the event rules you wish to add (see "Event rule definition" on page 222 for XML reference and see "The composer editor" on page 236 for details on setting up an XML editor).

;unlock
: Indicates that the object definitions must be unlocked if locked by the same user in the same session. If you did not lock the object and you use the ;unlock option, when you issue the command you receive an error message and the object is not replaced.

Comments

The text file is validated at the client and, if correct, objects are inserted into the database on the master domain manager. **Composer** transforms object definitions into an XML definition used at the server; otherwise the command is interrupted and an error message is displayed. This does not apply to event rule definitions since they are provided directly in XML format.

With the **add** command, if an object already exists, you are asked whether or not to replace it. This behavior does not affect existing job definitions inside job streams, and the job definitions are automatically updated without prompting any message. You can use the option **unlock** to update existing objects you previously locked using only one command. For all new objects inserted, the option is ignored. If you change the name of an object, it is interpreted by **composer** as a new object and will be inserted. A **rename** command is recommended in this case.

The add command checks for loop dependencies inside job streams. For example, if `job1` follows `job2`, and `job2` follows `job1` there is a loop dependency. When a loop dependency inside a job stream is found, an error is displayed. The add command does not check for loop dependencies between job streams because, depending on the complexity of the scheduling activities, this check might be too time and CPU consuming.

Examples

To add the jobs from the file `myjobs`, run the following command:

```
add myjobs
```

To add the job streams from the file `mysked`, run the following command:

```
a mysked
```

To add the workstations, workstation classes, and domains from the file `cpus.src`, run the following command:

```
a cpus.src
```

To add the user definitions from the file `users_nt`, run the following command:

```
a users_nt
```

To add the event rule definitions you edited in a file named `newrules.xml`, run:

```
a newrules.xml
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**
2. Choose one of the following:
   - To add workstations, click **Scheduling Environment→Design→Create Workstations**
   - To add event rules, click **Workload→Design→Create Event Rules**
   - To add all other objects, click **Workload→Design→Create Workload Definitions**
3. Select an engine name and click **Go**
4. Specify one of the following:
   - For workstations, specify your choices in the properties panel
   - For event rules, specify your choices in the Event Rule Editor panel
   - For all other objects:
     a. Click **New** in the Working List toolbar of the ensuing popup window
     b. Select the type of object that you want to add
     c. Specify your choices in the properties panel

authenticate

Switches to another user credentials while running **composer**.

Authorization

Any user authorized to run **composer** is authorized to issue this command.

Syntax

{**authenticate** | **au**} [**username=***username* **password=***password*]

Arguments

username=*username*
: The username of the user you want to switch to.

password=*password*
: The password of the user you want to switch to.

Comments

A message is displayed communicating the authentication failure or success. This command is used only in interactive mode.

Examples

To switch to user **tws_user1** with password **mypasswd1** from within the **composer** command-line program, run the following command:

```
au username=tws_user1 password=mypasswd1
```

continue

Specifies that the next command error is to be ignored.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**continue** | **c**}

Comments

This command is useful when multiple commands are entered on the command line or redirected from a file. It instructs **composer** to continue running commands even if the next command, following **continue**, results in an error. This command is not needed when you enter commands interactively because **composer** does not quit on an error.

Examples

If you want the composer to continue with the **print** command if the **delete** command results in an error, run the following command:

```
composer "c&delete cpu=site4&print cpu=@"
```

delete

Deletes object definitions in the database.

Authorization

You must have ***delete*** access to the objects being deleted.

Syntax

{**delete** | **de**}
{[**calendars** | **calendar** | **cal**=*calname*] |
[**eventrule** | **erule** | **er**=*eventrulename*] |
[**parms** | **parm** | **vb**=[*tablename.*]*variablename*] |
[**vartable** | **vt**=*tablename*] |
[**prompts** | **prom**=*promptname*] |
[**resources** | **resource** | **res**=[*workstationname#*]*resourcename*] |
[**cpu**={*workstationname* [**;force**] | *workstationclassname* [**;force**]| *domainname*}]
[**workstation** | **ws**=*workstationname*] [**;force**] |
[**workstationclass** | **wscl**=*workstationclassname*] [**;force**] |
[**domain** | **dom**=*domainname*] |
[**jobs** | **jobdefinition** | **jd**=[*workstationname#*]*jobname*] |
[**sched** | **jobstream** | **js**= [*workstationname#*]*jstreamname*
  [***valid from*** *date*|***valid to*** *date* |***valid in*** *date date*]
 ] |
[**users** | **user**=[*workstationname#*]*username*]}
[**;noask**]

Arguments

calendars | calendar | cal
If no argument follows, deletes all calendar definitions.

If argument *calname* follows, deletes the *calname* calendar. Wildcard characters are permitted.

parms|parm|vb
If no argument follows, deletes all global variable definitions found in the default variable table.

If argument *tablename.variablename* follows, deletes the *variablename* variable of the *tablename* table. If *tablename* is omitted, **composer** looks for the variable definition in the default variable table. Wildcard characters are permitted on both *tablename* and *variablename*. For example:

```
delete parms=@.@
```

Deletes all variables from all tables.

```
delete parms=@
```

Deletes all variables from the default table.

```
delete parms=@.acct@
```

Deletes all the variables whose name starts with `acct` from all the existing tables.

Remember: While you delete a variable, the variable table that contains it is locked. This implies that, while the table is locked, no other user can run any other locking commands on it or on the variables it contains.

vartable | vt
If no argument follows, deletes all variable table definitions.

If argument *tablename* variable table follows, deletes the *tablename* variable table. Wildcard characters are permitted.

prompts | prom
If no argument follows, deletes all prompt definitions.

If argument *promptname* follows, deletes the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res
If no argument follows, deletes all resource definitions.

If argument *workstationame#resourcename* follows, deletes the *resourcename* resource of the *workstationame* workstation on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *resourcename*.

cpu Deletes workstations, workstation classes, or domains.

workstation
The name of the workstation. Wildcard characters are permitted. If you specify the **force** argument, the workstation definition is removed from the Tivoli Workload Scheduler database.

workstationclass
The name of the workstation class. Wildcard characters are

permitted. If you specify the **force** argument, the workstationclass definition is removed from the Tivoli Workload Scheduler database.

domain The name of the domain. Wildcard characters are permitted.

workstation | ws
If no argument follows, deletes all workstation definitions.

If argument *workstationname* follows, deletes the *workstationname* workstation. Wildcard characters are permitted. If you specify the **force** argument, the workstation definition is removed from the Tivoli Workload Scheduler database.

domain | dom
If no argument follows, deletes all domain definitions.

If argument *domainname* follows, deletes the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl
If no argument follows, deletes all workstation class definitions.

If argument *workstationclassname* follows, deletes the *workstationclassname* workstation class. Wildcard characters are permitted. If you specify the **force** argument, the workstationclass definition is removed from the Tivoli Workload Scheduler database.

jobs | jobdefinition | jd
If no argument follows, deletes all job definitions.

If argument *workstationame#jobname* follows, deletes the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jobname*.

sched | jobstream | js
If no argument follows, deletes all job stream definitions.

If argument *workstationame#jstreamname* follows, deletes the *jstreamname* job stream of the *workstationame* workstation on which the job stream is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jstreamname*.

valid from
date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to
date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in
date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

users | user
If no argument follows, deletes all user definitions.

If argument *workstationame#username* follows, deletes the *username* user of the *workstationame* workstation on which the user is defined. If *workstationame* is omitted, the default is the workstation on which

composer is running. Wildcard characters are permitted for both *workstationname* and *username*. The password field is not copied for security reasons.

eventrule | erule | er
If no argument follows, deletes all event rule definitions.

If argument *eventrulename* follows, deletes the *eventrulename* event rule. Wildcard characters are permitted.

;noask Specifies not to prompt for confirmation before taking action on each qualifying object.

Comments

If you use wildcard characters to specify a set of definitions, **composer** requires confirmation before deleting each matching definition. A confirmation is required before deleting each matching definition if you do not specify the **noask** option.

To delete an object, it must not be locked. If some matching objects are locked during the command processing, an error message with the list of these objects is shown to the user.

Examples

To delete `job3` that is launched on workstation `site3`, run the following command:

```
delete jobs=site3#job3
```

To delete all workstations with names starting with `ux`, run the following command:

```
de cpu=ux@
```

To delete all job streams with names starting with `test` on all workstations, run the following command:

```
de sched=@#test@
```

To delete all the event rules named from `rulejs320` to `rulejs329`, run the following command:

```
de erule=rulejs32?
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**
2. Choose one of the following:
   - To delete workstations, click **Scheduling Environment→Design→List Workstations**
   - To delete event rules, click **Workload→Design→List Event Rules**
   - To delete all other objects, click **Workload→Design→List Workload Definitions**
3. Select an engine name
4. List the objects that you want to delete using the provided filters and click or select **Delete**.

display

Displays the details of one or more object definitions of the same type stored in the database. The entire definition of the object is displayed.

Authorization

You must have ***display*** access to the object being displayed. If you want to use the **full** keyword you must have also the ***display*** access to the jobs contained in the job stream definition.

Syntax

{**display** | **di**}
{[**calendars** | **calendar** | **cal**=*calname*] |
[**eventrule** | **erule** | **er**=*eventrulename*] |
[**parms** | **parm** | **vb**=*variablename*.]*variablename*] |
[**vartable** | **vt**=*tablename*] |
[**prompts** | **prom**=*promptname*] |
[**resources** | **resource** | **res**=[*workstationname*#]*resourcename*] |
[**cpu**={*workstationname* | *workstationclassname* | *domainname*}]
[**workstation** | **ws**=*workstationname*] |
[**workstationclass** | **wscl**=*workstationclassname*] |
[**domain** | **dom**=*domainname*] |
[**jobs** | **jobdefinition** | **jd**=[*workstationname*#]*jobname*] |
[**sched** | **jobstream** | **js**= [*workstationname*#]*jstreamname*
  *[**valid from** date|**valid to** date |**valid in** date date]*
  *[**;full**]*] |
[**users** | **user**=[*workstationname*#]*username*]}
[**;offline**]

Arguments

calendars | calendar | cal
: If no argument follows, displays all calendar definitions.

  If argument *calname* follows, displays the *calname* calendar. Wildcard characters are permitted.

eventrule | erule | er
: If no argument follows, displays all event rule definitions.

  If argument *eventrulename* follows, displays the *eventrulename* event rule. Wildcard characters are permitted.

parms | parm | vb
: If no argument follows, displays all global variable definitions found in the default variable table.

  If argument *tablename.variablename* follows, displays the *variablename* variable of the specified table. If *tablename* variable table is omitted, **composer** looks for the variable definition in the default variable table. Wildcard characters can be used on both *tablename* variable table and *variablename* variable. For example:

```
display parms=@.@
```

  Displays all variables on all tables.

```
display parms=@
```

Displays all variables on the default table.

```
display parms=@.acct@
```

Displays all the variables whose name starts with `acct` on all the existing tables.

vartable | vt
If no argument follows, displays all variable table definitions.

If argument *tablename* variable table follows, displays the *tablename* variable table. Wildcard characters are permitted.

prompts | prom
If no argument follows, displays all prompt definitions.

If argument *promptname* follows, displays the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res
If no argument follows, displays all resource definitions.

If argument *workstationame#resourcename* follows, displays the *resourcename* resource of the *workstationame* workstation on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *resourcename*.

cpu Displays workstations, workstation classes, or domains.

workstation
The name of the workstation. Wildcard characters are permitted.

workstationclass
The name of the workstation class. Wildcard characters are permitted.

domain The name of the domain. Wildcard characters are permitted.

workstation | ws
If no argument follows, displays all workstation definitions.

If argument *workstationname* follows, displays the *workstationname* workstation. Wildcard characters are permitted.

domain | dom
If no argument follows, displays all domain definitions.

If argument *domainname* follows, displays the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl
If no argument follows, displays all workstation class definitions.

If argument *workstationclassname* follows, displays the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd
If no argument follows, displays all job definitions.

If argument *workstationame#jobname* follows, displays the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jobname*.

sched | jobstream | js
If no argument follows, displays all job stream definitions.

If argument *workstationname#jstreamname* follows, displays the *jstreamname* job stream of the *workstationname* workstation on which the job stream is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *jstreamname*.

valid from
date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to
date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in
date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

full Displays also all job definitions contained in the job stream.

users | user
If no argument follows, displays all user definitions.

If argument *workstationname#username* follows, displays the *username* user of the *workstationname* workstation on which the user is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *username*.

;offline
Sends the output of the command to the **composer** output device. For information about this device, see "UNIX variables" on page 236.

Results

The **display** command returns you the following information about the object to be displayed:

- a summary row containing information about the selected object
- the selected object definition

Depending on the value set in the ***MAESTROCOLUMNS*** local variable the summary row shows different sets of information about the selected object.

Table 51 shows an example of the output produced based on the value set for the ***MAESTROCOLUMNS*** variable.

Table 51. Output formats for displaying scheduling objects

Object Type	Output format if MAESTROCOLUMNS<120	Output format if MAESTROCOLUMNS ≥ 120
Calendar	"CalendarName : UpdatedOn : UpdatedBy : LockedBy"	"CalendarName : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Domain	"DomainName : ParentDomain : Master : UpdatedOn : LockedBy"	"DomainName : ParentDomain : Master : UpdatedBy : UpdatedOn : LockedBy : LockedOn"

Table 51. Output formats for displaying scheduling objects (continued)

Object Type	Output format if MAESTROCOLUMNS<120	Output format if MAESTROCOLUMNS ≥ 120
Event rule	"EventRuleName : Type : Draft : Status : UpdatedOn : LockedBy"	"EventRuleName : Type : Draft : Status : UpdatedOn : LockedBy : LockedOn"
Job	"Workstation : JobDefinitionName : UpdatedOn : LockedBy"	"Workstation : JobDefinitionName : TaskType : UpdatedBy : LockedBy : LockedOn"
Job Stream	"Workstation : JobstreamName : Validfrom : UpdatedOn : LockedBy"	"Workstation : JobstreamName : Draft : ValidFrom : ValidTo : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Parameter	"VariableTableName : VariableName : UpdatedOn : LockedBy"	"VariableTableName : VariableName : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Prompt	"PromptName : UpdatedOn : LockedBy "	"PromptName : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Resource	"Workstation : ResourceName : Quantity : UpdatedOn : LockedBy "	"Workstation : ResourceName : Quantity : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Variable Table	"VariableTableName : Default : UpdatedOn : LockedBy "	"VariableTableName : Default : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
User	"Workstation : UserName : UpdatedOn : LockedBy"	"UserName : Workstation : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Workstation	"WorkstationName : Type : Domain : Ignored : UpdatedOn : LockedBy"	"WorkstationName : Type : Domain : OsType : Ignored : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Workstation Class	"WorkstationClassName : Ignored : UpdatedOn : LockedBy"	"WorkstationClassName : Ignored : UpdatedBy : UpdatedOn : LockedBy : LockedOn"

See "Offline output" on page 235 for more information on how to set ***MAESTROCOLUMNS*.**

Examples

To display all calendars, run the following command:

```
display calendars=@
```

this is a sample output:

```
Calendar Name     Updated On  Locked By
----------------  ----------  ---------------------
HOLIDAYS          12/31/2005  tws83
HOLIDAYS
  01/01/2006 02/15/2006 05/31/2006


Calendar Name     Updated On  Locked By
----------------  ----------  ---------------------
MONTHEND          01/01/2006  -

MONTHEND
  "Month end dates 1st half 2006"
  01/31/2006 02/28/2006 03/31/2006 04/30/2006 05/31/2006 06/30/2006

Calendar Name     Updated On  Locked By
----------------  ----------  ---------------------
PAYDAYS           01/02/2006  -

PAYDAYS
  01/15/2006 02/15/2006 03/15/2006 04/15/2006 05/14/2006 06/15/2006
```

To print the output of the display command on all job streams that are launched on workstation `site2`, run the following command:

```
di sched=site2#@;offline
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**
2. Choose one of the following:
   - To display workstations, click **Scheduling Environment→Design→List Workstations**
   - To display event rules, click **Workload→Design→List Event Rules** and launch or create a task to retrieve event rule definitions
   - To display all other objects, click **Workload→Design→List Workload Definitions**
3. Select an engine name
4. List the objects that you want to display using the provided filters and click **Display**.

edit

Edits a file.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**edit** | **ed**} *filename*

Arguments

filename
The name of the file to be edited.

Comments

An editor is started and the specified file is opened for editing. See "The composer editor" on page 236 for more information.

Examples

To open the file `mytemp` for editing, run the following command:

```
edit mytemp
```

To open the file `resfile` for editing, run the following command:

```
ed resfile
```

exit

Exits the **composer** command line program.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**exit** | **e**}

Comments

When you are running the **composer** command line program in help mode, this command returns **composer** to command input mode.

Examples

To exit the **composer** command line program, run the following command:

```
exit
```

or:

```
e
```

extract

Creates a text file containing object definitions extracted from the database.

Authorization

You must have ***display*** access to the objects being copied and, if you want to use the **;lock** keyword, also the ***modify*** access.

Syntax

{**create** | **cr** | **extract** | **ext**} *filename* **from**
{[**calendars** | **calendar** | **cal**=*calname*] |
[**eventrule** | **erule** | **er**=*eventrulename*] |
[**parms** | **parm** | **vb**=[*tablename*.]*variablename*] |
[**vartable** | **vt**=*tablename*] |
[**prompts** | **prom**=*promptname*] |
[**resources** | **resource** | **res**=[*workstationame#*]*resourcename*] |
[**cpu**={*workstationame* | *workstationclassname* | *domainame*}] |
[**workstation** | **ws**=*workstationame*] |
[**workstationclass** | **wscl**=*workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*workstationame#*]*jobname*] |
[**sched** | **jobstream** | **js**= [*workstationame#*]*jstreamname*
*[**valid from** date|**valid to** date |**valid in** date date]*
*[**;full**]*] |
| [**users** | **user**=[*workstationame#*]*username* [;password]]}
[**;lock**]

Arguments

filename
Specifies the name of the file to contain the object definitions.

calendars | calendar | cal
If no argument follows, copies all calendar definitions into the file.

If argument *calname* follows, copies the *calname* calendar into the file. Wildcard characters are permitted.

eventrule | erule | er
If no argument follows, copies all event rule definitions into the XML file.

If argument *eventrulename* follows, copies the *eventrulename* event rule into the file. Wildcard characters are permitted.

parms | parm | vb
If no argument follows, copies all global variable definitions found in the default variable table into the file.

If argument *tablename.variablename* follows, copies the *variablename* variable of the specified *tablename* variable table into the file. If the *tablename* variable table is omitted, **composer** looks for the variable definition in the default variable table. Wildcard characters are permitted on both *tablename* variable table and *variablename* variable.

For example:

```
create parmfile from parms=@.@
```

Copies all variables from all tables.

```
create parmfile from parms=@
```

Copies all variables from the default table.

```
create parmfile from parms=@.acct@
```

Copies all the variables whose name starts with `acct` from all the existing tables.

Remember: Using the **;lock** option on a variable locks the variable table that contains it. This implies that, while the table is locked, no other user can run any other locking commands on it or on the variables it contains.

vartable | vt
If no argument follows, copies all variable table definitions into the file.

If argument *tablename* variable table follows, copies the *tablename* variable table into the file. Wildcard characters are permitted.

prompts | prom
If no argument follows, copies all prompt definitions into the file.

If argument *promptname* follows, copies the *promptname* prompt into the file. Wildcard characters are permitted.

resources | resource | res
If no argument follows, copies all resource definitions into the file.

If argument *workstationname#resourcename* follows, copies the *resourcename* resource of the *workstationname* workstation on which the resource is defined into the file. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *resourcename*.

cpu Copies workstations, workstation classes, or domains into the file.

workstation
: The name of the workstation. Wildcard characters are permitted.

workstationclass
: The name of the workstation class. Wildcard characters are permitted.

domain The name of the domain. Wildcard characters are permitted.

workstation | ws
If no argument follows, copies all workstation definitions into the file.

If argument *workstationname* follows, copies the *workstationname* workstation into the file. Wildcard characters are permitted.

domain | dom
If no argument follows, copies all domain definitions into the file.

If argument *domainname* follows, copies the *domainname* domain into the file. Wildcard characters are permitted.

workstationclass | wscl
If no argument follows, copies all workstation class definitions into the file.

If argument *workstationclassname* follows, copies the *workstationclassname* workstation class into the file. Wildcard characters are permitted.

jobs | jobdefinition | jd
If no argument follows, copies all job definitions into the file.

If argument *workstationame#jobname* follows, copies the *jobname* job of the *workstationame* workstation on which the job runs into the file. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jobname*.

sched | jobstream | js
If no argument follows, copies all job stream definitions into the file.

If argument *workstationame#jstreamname* follows, copies the *jstreamname* job stream of the *workstationame* workstation on which the job stream is defined into the file. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jstreamname*.

valid from
date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to
date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in
date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

full Copies also all job definitions contained in the job stream.

users | user
If no argument follows, copies all user definitions into the file.

If argument *workstationname#username* follows, copies the *username* user of the *workstationname* workstation on which the user is defined into the file. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *username*.

If you do not add the **;password** option, the password defined for the user is saved in the output file as a sequence of 10 asterisks (*) and cannot be reused.

If you do add the **;password** option, the password defined for the user is encrypted and saved in the output file. It can thus be re-imported and used again.

;lock Specifies to keep locked the selected object.

Comments

You can use this command to create a file containing parameter definitions to be imported into the parameter database defined locally on a workstation. For more information on how to import parameter definitions locally, refer to "parms" on page 457.

You can invoke the command with the old name "create" or the new name "extract". Without the **lock** option, database locking is not checked and all matching objects are extracted to the file. After you create a file, you can use the **edit** command to make changes to the file and the **add** or **replace** command to add or update the database.

You can specify with the **lock** option, if the objects that respond to the selected criteria, must remain locked by the user in the database. If **composer**, during the extraction, find some of these objects already locked by someone else, these objects are not inserted into the file and a message to *stdout* is presented for each locked object.

Examples

To create a file named `caltemp` containing all calendars, run the following command:

```
create caltemp from calendars=@
```

To create a file named `stemp` containing all job streams defined on the workstation where **composer** runs, run the following command:

```
cr stemp from jobstream=@
```

To create a file named `alljobs.txt` containing all job definitions, run the following command:

```
extract alljobs.txt from jd=@#@
```

To create a file named `allrules.xml` containing all event rule definitions, run the following command:

```
ext allrules.xml from erule=@
```

To create a file named dbmainadm.txt with the definition of user princeps of workstation dbserv349, including the encrypted password, run:

```
composer extract c:\dbmainadm.txt from user=dbserv349#princeps;password
```

The contents of file dbmainadm.txt will be:

```
USERNAME princeps
  PASSWORD "ENCRYPT:EIu7PP+gvS8="
END
```

help

Displays the on-line help for a command or displays the list of commands that can be issued from **composer**. Not available in Windows.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**help** | **h**} {*command*|*keyword*}

Arguments

command
: Specifies the name of a **composer** or system command. For **composer** commands, enter the full command name; abbreviations and short forms are not supported.

keyword
: You can also enter the following keywords:

COMMANDS
: Lists all composer commands.

RUNCONPOSER
: How to run composer.

SETUPCOMPOSER
: Describes how to setup to use composer.

SPECIALCHAR
: Describes the wildcards, delimiters and other special characters you can use.

Examples

To display a list of all **composer** commands, run the following command:

```
help commands
```

To display information about the **add** command, run the following command:

```
help add
```

To display information about the special characters you can use, run the following command:

```
h specialchar
```

list

Lists, or prints summary information about objects defined in the Tivoli Workload Scheduler database. List provides you with the list of objects names with their attributes. Print sends the list of objects names with their attributes to the device or file specified in the ***MAESTROLP*** local variable. The print command can be used to send the output to a local printer, if the ***MAESTROLP*** variable is set accordingly.

Authorization

You must have ***display*** access to the object being listed or printed if the ***enListSecChk*** option is set to **yes** on the master domain manager.

Syntax

{**list** | **l**}
{[**calendars** | **calendar** | **cal**=*calname*] |
[**eventrule** | **erule** | **er**=*eventrulename*] |
[**parms** | **parm** | **vb**=[*tablename.*]*variablename*] |
[**vartable** | **vt**=*tablename*] |
[**prompts** | **prom**=*promptname*] |
[**resources** | **resource** | **res**=[*workstationame#*]*resourcename*] |
[**cpu**={*workstationame* | *workstationclassname* | *domainame*}]
[**workstation** | **ws**=*workstationame*] |
[**workstationclass** | **wscl**=*workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*workstationame#*]*jobname*] |
[**sched** |**jobstream** | **js**= [*workstationame#*]*jstreamname*
  *[**valid from** date*|
   valid to** date* |valid in** date date]* |
[**users** | **user**=[*workstationame#*]*username*]}
[**;offline**]

Arguments

calendars | calendar | cal
: If no argument follows, lists or prints all calendar definitions.

  If argument *calname* follows, lists or prints the *calname* calendar. Wildcard characters are permitted.

eventrule | erule | er
: If no argument follows, lists or prints all event rule definitions.

  If argument *eventrulename* follows, lists or prints the *eventrulename* event rule. Wildcard characters are permitted.

parms | parm | vb
: If no argument follows, lists or prints all global variable definitions found in the default variable table.

  If argument *tablename.variablename* follows, lists or prints the *variablename* variable of the *tablename* table. If *tablename* is omitted, **composer** looks for the variable definition in the default variable table. Wildcard characters can be used on both *tablename* and *variablename*. For example:

```
list parms=@.@
```

  Lists all variables on all tables.

```
list parms=@
```

Lists all variables on the default table.

```
list parms=@.acct@
```

Lists all the variables whose name starts with `acct` on all the existing tables.

vartable | vt
If no argument follows, lists or prints all variable table definitions.

If argument *tablename* variable table follows, lists or prints the *tablename* variable table. Wildcard characters are permitted.

prompts | prom
If no argument follows, lists or prints all prompt definitions.

If argument *promptname* follows, lists or prints the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res
If no argument follows, lists or prints all resource definitions.

If argument *workstationame#resourcename* follows, lists or prints the *resourcename* resource of the *workstationame* workstation on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *resourcename*.

cpu Lists or prints workstations, workstation classes, or domains.

workstation
The name of the workstation. Wildcard characters are permitted.

workstationclass
The name of the workstation class. Wildcard characters are permitted.

domain The name of the domain. Wildcard characters are permitted.

workstation | ws
If no argument follows, lists or prints all workstation definitions.

If argument *workstationname* follows, lists or prints the *workstationname* workstation. Wildcard characters are permitted.

domain | dom
If no argument follows, lists or prints all domain definitions.

If argument *domainname* follows, lists or prints the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl
If no argument follows, lists or prints all workstation class definitions.

If argument *workstationclassname* follows, lists or prints the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd
If no argument follows, lists or prints all job definitions.

If argument *workstationame#jobname* follows, lists or prints the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jobname*.

sched | jobstream | js

If no argument follows, lists or prints all job stream definitions.

If argument *workstationname#jstreamname* follows, lists or prints the *jstreamname* job stream of the *workstationname* workstation on which the job stream is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *jstreamname*.

valid from

date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

users | user

If no argument follows, lists or prints all user definitions.

If argument *workstationname#username* follows, lists or prints the *username* user of the *workstationname* workstation on which the user is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *username*.

;offline

Sends the output of the command to the **composer** output device. For information about this device, see “UNIX variables” on page 236. The **list ..... ;offline** command is equivalent to the **print** command.

Results

List provides you with the list of objects names with their attributes. Print sends the list of objects names with their attributes to the device or file set in the *MAESTROLP* local variable. The print command can be used to send the output to a local printer, if you set the *MAESTROLP* variable accordingly. Make sure the *MAESTROLP* is set in your environment before running the print command.

Depending on the value set in the ***MAESTROCOLUMNS*** local variable the different sets of information about the selected object can be shown.

Table 52 shows an example of the output produced according to the value set for the ***MAESTROCOLUMNS*** variable.

Table 52. Output formats for displaying scheduling objects

Object Type	Output format if MAESTROCOLUMNS<120	Output format if MAESTROCOLUMNS ≥ 120
Calendar	"CalendarName : UpdatedOn : UpdatedBy : LockedBy"	"CalendarName : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Domain	"DomainName : ParentDomain : Master : UpdatedOn : LockedBy"	"DomainName : ParentDomain : Master : UpdatedBy : UpdatedOn : LockedBy : LockedOn"

Table 52. Output formats for displaying scheduling objects (continued)

Object Type	Output format if MAESTROCOLUMNS<120	Output format if MAESTROCOLUMNS ≥ 120
Event rule	"EventRuleName : Type : Draft : Status : UpdatedOn : LockedBy"	"EventRuleName : Type : Draft : Status : UpdatedOn : LockedBy : LockedOn"
Job	"Workstation : JobDefinitionName : UpdatedOn : LockedBy"	"Workstation : JobDefinitionName : TaskType : UpdatedBy : LockedBy : LockedOn"
Job Stream	"Workstation : JobstreamName : Validfrom : UpdatedOn : LockedBy"	"Workstation : JobstreamName : Draft : ValidFrom : ValidTo : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Parameter	"VariableTableName : VariableName : UpdatedOn : LockedBy"	"VariableTableName : VariableName : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Prompt	"PromptName : UpdatedOn : LockedBy "	"PromptName : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Resource	"Workstation : ResourceName : Quantity : UpdatedOn : LockedBy "	"Workstation : ResourceName : Quantity : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Variable Table	"VariableTableName : Default : UpdatedOn : LockedBy "	"VariableTableName : Default : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
User	"Workstation : UserName : UpdatedOn : LockedBy"	"UserName : Workstation : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Workstation	"WorkstationName : Type : Domain : Ignored : UpdatedOn : LockedBy"	"WorkstationName : Type : Domain : OsType : Ignored : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
Workstation Class	"WorkstationClassName : Ignored : UpdatedOn : LockedBy"	"WorkstationClassName : Ignored : UpdatedBy : UpdatedOn : LockedBy : LockedOn"

See "Offline output" on page 235 for more information on how to set ***MAESTROLP***.

Examples

- To list all calendars, run the following command:

  ```
  list calendars=@
  ```

  this is a sample output:

  ```
  Calendar Name     Updated On  Locked By
  ----------------  ----------  ---------------------
  HOLIDAYS          03/02/2010
  PAYDAYS           03/02/2010
  HOLIDAYS          03/02/2010
    01/01/2010 02/15/2010 05/31/2010


  Calendar Name     Updated On  Locked By
  ----------------  ----------  ---------------------
  MONTHEND          01/01/2010  -

  MONTHEND
    "Month end dates 1st half 2010"
  ```

```
  01/31/2010 02/28/2010 03/31/2010 04/30/2010 05/31/2010 06/30/2010

Calendar Name     Updated On  Locked By
----------------  ----------  ---------------------
PAYDAYS           01/02/2010  -

PAYDAYS
  01/15/2010 02/15/2010 03/15/2010 04/15/2010 05/14/2010 06/15/2010
```

- To list all your defined event rules, run the following command:

  ```
  list er=@
  ```

  If MAESTROCOLUMNS=80, the output looks something like this:

  ```
  Event Rule Name   Type       Draft  Status     Updated On  Locked By
  ----------------  ---------  -----  ---------  ----------  -------------
  EVENT-MULTIPLE1   filter            active     06/06/2009  -
  EVENT-MULTIPLE2   filter            active     06/06/2009  -
  EVENT-MULTIPLE3   filter            active     06/06/2009  -
  M_SUCC_12_S       sequence   Y      inactive   06/07/2009  -
  M_SUCC_12_S_A     filter            active     06/07/2009  -
  M_SUCC_12_S_B     filter     Y      inactive   06/07/2009  -
  NEWEVENTRULE      filter            active     06/01/2009  administrator
  ```

  If MAESTROCOLUMNS≥120, the output looks something like this:

  ```
  Event Rule Name         Type      Draft Status   Updated On  Locked By
  ---------------------   --------- ----- -------- ---------- ---------
  EVENT-MULTIPLE1         filter          active   06/06/2009 -
  EVENT-MULTIPLE2         filter          active   06/06/2009 -
  EVENT-MULTIPLE3         filter          active   06/06/2009 -
  M_SUCC_12_S             sequence   Y    inactive 06/07/2009 -
  M_SUCC_12_S_A           filter          active   06/07/2009 -
  M_SUCC_12_S_B           filter     Y    inactive 06/07/2009 -
  NEWEVENTRULE            filter          active   06/01/2009 administrator
  ```

- To view the properties of the NC1150691 dynamic agent workstation, run the following command:

  ```
  list ws=NC1150691
  ```

  An output similar to the following is displayed:

  ```
  Workstation Name  Type     Domain            Ignored  Updated On  Locked By
  ----------------  -------  ----------------  -------  ----------  ----------
  NC1150691         agent    -                          03/31/2010  -

  CPUNAME NC1150691
    DESCRIPTION "This workstation was automatically created at agent
                installation time."
    OS WNT
    NODE nc115069.romelab.it.ibm.com SECUREADDR 22114
    TIMEZONE GMT+1
    FOR MAESTRO HOST NC115069_DWB
      TYPE AGENT
      PROTOCOL HTTPS
  END
  ```

- To view the properties of the POOL_A pool workstation, including all its members, run the following command:

  ```
  list ws=POOL_A
  ```

  An output similar to the following is displayed:

  ```
  Workstation Name  Type     Domain            Ignored  Updated On  Locked By
  ----------------  -------  ----------------  -------  ----------  ----------------
  POOL_A            pool     -                          03/31/2010  -

  CPUNAME POOL_A
  ```

```
    DESCRIPTION "This is a manually created pool"
    VARTABLE TABLE1
    OS OTHER
    TIMEZONE America/Argentina/Buenos_Aires
    FOR MAESTRO HOST NC115069_DWB
      TYPE POOL
    MEMBERS
      NC1150691
      NC1150692
  END
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**
2. Choose one of the following:
   - To list workstations, click **Scheduling Environment→Design→List Workstations**
   - To list event rules, click **Workload→Design→List Event Rules**
   - To list all other objects, click **Workload→Design→List Workload Definitions**
3. Select an engine name
4. List the objects that you want to see using the provided filters.

lock

Locks the access to scheduling objects definitions in the database.

Authorization

You must have ***modify*** access to the object.

Syntax

{**lock** | **lo**}
{[**calendars** | **calendar** | **cal**=*calname*] |
[**eventrule** | **erule** | **er**=*eventrulename*]
[**parms** | **parm** | **vb**=[*tablename.*]*variablename*] |
[**vartable** | **vt**=*tablename*] |
[**prompts** | **prom**=*promptname*] |
[**resources** | **resource** | **res**=[*workstationame#*]*resourcename*] |
[**cpu**={*workstationame* | *workstationclassname* | *domainame*}]
[**workstation** | **ws**=*workstationame*] |
[**workstationclass** | **wscl**=*workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*workstationame#*]*jobname*] |
[**sched**|**jobstream**|**js**= [*workstationame#*]*jstreamname*
  [***valid from*** *date*|***valid to*** *date* |***valid in*** *date date*]] |
[**users** | **user**=[*workstationame#*]*username*]}

Arguments

calendars
: Locks all calendar definitions.

calendars | calendar | cal
: If no argument follows, locks all calendar definitions.

If argument *calname* follows, locks the *calname* calendar. Wildcard characters are permitted.

eventrule | erule | er
If no argument follows, locks all event rule definitions.

If argument *eventrulename* follows, locks the *eventrulename* event rule. Wildcard characters are permitted.

parms | parm | vb
If no argument follows, locks the entire default variable table.

If argument *tablename.variablename* follows, locks the entire table containing the *variablename* variable. If *tablename* is omitted, **composer** locks the entire default variable table.

Note: When you lock a variable, this locks the entire variable table that contains it. This implies that, while the table is locked, no other user can run any other locking commands on it.

Wildcard characters can be used on both *tablename* and *variablename*. For example:

```
lock parms=@.@
```

Locks all variables on all tables. As a result, all variable tables are locked.

```
lock parms=@
```

Locks all variables on the default table. As a result, the variable table is locked.

```
lock parms=@.acct@
```

Locks all the variables whose name starts with `acct` on all the existing tables. As a result, all the variable tables that contain at least one variable named in this way are locked.

vartable | vt
If no argument follows, locks all variable table definitions.

If argument *tablename* variable table follows, locks the *tablename* variable table. Wildcard characters are permitted.

prompts | prom
If no argument follows, locks all prompt definitions.

If argument *promptname* follows, locks the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res
If no argument follows, locks all resource definitions.

If argument *workstationame#resourcename* follows, locks the *resourcename* resource of the *workstationame* workstation on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *resourcename*.

cpu Locks workstations, workstation classes, or domains.

workstation
The name of the workstation. Wildcard characters are permitted.

workstationclass
The name of the workstation class. Wildcard characters are permitted.

domain The name of the domain. Wildcard characters are permitted.

workstation | ws
If no argument follows, locks all workstation definitions.

If argument *workstationname* follows, locks the *workstationname* workstation. Wildcard characters are permitted.

domain | dom
If no argument follows, locks all domain definitions.

If argument *domainname* follows, locks the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl
If no argument follows, locks all workstation class definitions.

If argument *workstationclassname* follows, locks the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd
If no argument follows, locks all job definitions.

If argument *workstationame#jobname* follows, locks the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jobname*.

sched | jobstream | js
If no argument follows, locks all job stream definitions.

If argument *workstationame#jstreamname* follows, locks the *jstreamname* job stream of the *workstationame* workstation on which the job stream is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jstreamname*.

valid from
date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to
date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in
date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

users | user
If no argument follows, locks all user definitions.

If argument *workstationame#username* follows, locks the *username* user of the *workstationame* workstation on which the user is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *username*.

Comments

Objects are locked to make sure that definitions in the database are not overwritten by different users accessing concurrently to the same objects.

With this command the user explicitly acquires locks of database objects. When one user has an object locked, any other user has read only access until the object is released or explicitly unlocked by the administrator. If one user tries to lock an object that is already locked by someone else (other user), an error message is returned.

Locks on database objects are acquired by the user using `username` and `session`, where `session` is a string that can be set in the environment variable **TWS_SESSION** identifying that specific user work session.

This means that, on a machine, the **TWS_SESSION** identifier is different for:
- a user connected in two different shells to the **composer** command line program.
- a user connected, disconnected and then connected again to the **composer** command line from the same shell.

If no value is assigned to **TWS_SESSION**, then the default value identifying the session is set as follows:
- If using **composer** in batch mode, the default value is the *username* used by the user when connecting to the master domain manager.
- If using **composer** in interactive mode, the default value corresponds to an alphanumeric string automatically created by the product.

Note: In the database the *username* of the user locking an object definition is saved in uppercase.

Examples

To lock the calendar named `Holidays`, run the command:

```
lock calendar=HOLIDAYS
```

See also

In the Tivoli Dynamic Workload Console, objects are automatically locked as long as you or another user have them open using the **Edit** button. Objects are not locked if you or another user opened them with **View**.

modify

Modifies or adds scheduling objects. When modifying objects, the **modify** command extracts only the objects that can be locked by the current user.

Authorization

You must have **add** access if you add a new scheduling object. If the object already exists in the database, you must have **modify** access to the object.

Syntax

{**modify** | **m**}
{[**calendars** | **calendar** | **cal**=*calname*] |

[**eventrule** | **erule** | **er**=*eventrulename*] |
[**parms** | **parm** | **vb**=[*tablename*.]*variablename*] |
[**vartable** | **vt**=*tablename*] |
[**prompts** | **prom**=*promptname*] |
[**resources** | **resource** | **res**=[*workstationame*#]*resourcename*] |
[**cpu**={*workstationame* | *workstationclassname* | *domainame*}]
[**workstation** | **ws**=*workstationame*] |
[**workstationclass** | **wscl**=*workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*workstationame*#]*jobname*] |
[**sched**|**jobstream**|**js**= [*workstationame*#]*jstreamname*
*[**valid from** date*|***valid to** date* |***valid in** date date]*
*[;**full**]*] |
[**users** | **user**=[*workstationame*#]*username*]}

Arguments

calendars | **calendar** | **cal**
If no argument follows, modifies all calendar definitions.

If argument *calname* follows, modifies the *calname* calendar. Wildcard characters are permitted.

eventrule | **erule** | **er**
If no argument follows, modifies all event rule definitions.

If argument *eventrulename* follows, modifies the *eventrulename* event rule. Wildcard characters are permitted.

parms|**parm**|**vb**
If no argument follows, modifies all global variable definitions found in the default variable table.

If argument *tablename.variablename* follows, modifies the specified variable of the *tablename* table. If *tablename* is omitted, **composer** looks for the *variablename* variable definition in the default variable table. Wildcard characters can be used on both *tablename* and *variablename*. For example:

```
modify parms=@.@
```

Modifies all variables on all tables.

```
modify parms=@
```

Modifies all variables on the default table.

```
modify parms=@.acct@
```

Modifies all the variables whose name starts with `acct` on all the existing tables.

Remember: The action of modifying or adding a variable locks the variable table that contains it. This implies that, while the table is locked, no other user can run any other locking commands on it or on the variables it contains.

vartable | **vt**
If no argument follows, modifies all variable table definitions.

If argument *tablename* variable table follows, modifies the *tablename* variable table. Wildcard characters are permitted.

prompts | prom
If no argument follows, modifies all prompt definitions.

If argument *promptname* follows, modifies the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res
If no argument follows, modifies all resource definitions.

If argument *workstationame#resourcename* follows, modifies the *resourcename* resource of the *workstationame* workstation on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *resourcename*.

cpu Modifies workstations, workstation classes, or domains.

workstation
The name of the workstation. Wildcard characters are permitted.

workstationclass
The name of the workstation class. Wildcard characters are permitted.

domain The name of the domain. Wildcard characters are permitted.

workstation | ws
If no argument follows, modifies all workstation definitions.

If argument *workstationname* follows, modifies the *workstationname* workstation. Wildcard characters are permitted.

domain | dom
If no argument follows, modifies all domain definitions.

If argument *domainname* follows, modifies the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl
If no argument follows, modifies all workstation class definitions.

If argument *workstationclassname* follows, modifies the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd
If no argument follows, modifies all job definitions.

If argument *workstationame#jobname* follows, modifies the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jobname*.

sched | jobstream | js
If no argument follows, modifies all job stream definitions.

If argument *workstationame#jstreamname* follows, modifies the *jstreamname* job stream of the *workstationame* workstation on which the job stream is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jstreamname*.

valid from
date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to

date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in

date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

full Modifies all job definitions contained in the job stream.

users | user

If no argument follows, modifies all user definitions.

If argument *workstationame#username* follows, modifies the *username* user of the *workstationame* workstation on which the user is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *username*.

Comments

The **modify** command performs the following sequence of actions:

1. Locks the objects in the database.
2. Copies the objects definition into a temporary file.
3. Edits the file.
4. Replaces the definition contained in the temporary file to the database.
5. If the **modify** command fails on a subset of the selected objects, **composer** asks *"do you want to re-edit?"* and the file saved before is reopened for editing and the next steps of the sequence are repeated.
6. Unlocks the objects in the database.

Event rule definitions are opened with an XML editor (see “Event rule definition” on page 222 for XML reference and see “The composer editor” on page 236 for details on setting up an XML editor).

If you modify with the same **modify** command two or more objects linked together by any relationship, for example a successor job and its predecessor job, then it might be relevant for the successful result of the **modify** command the order in which the objects are listed in the temporary file. This happens because the **modify** command reads in sequence the objects contained in the temporary file; so, if the referencing object is displayed before the object being referenced, the modify command might fail on the referencing object.

For example, if the command:

```
modify FTA1#@PROVA
```

produces the following temporary file:

```
SCHEDULE FTA1#PROVA VALIDFROM 08/31/2005
MATCHING SAMEDAY
:
FTA2#MY-JOB
 FOLLOWS FTA1#COPYOFPROVA.MY-JOB06
END

SCHEDULE FTA1#COPYOFPROVA VALIDFROM 08/31/2005
```

```
MATCHING SAMEDAY
:
FTA1#MY-JOB06
END
```

and you change the name of the predecessor job from FTA1#MY-JOB06 to FTA1#MY-JOB05 in both job streams FTA1#PROVA and FTA1#COPYOFPROVA, then the **modify** command:

1. At first tries to change the definition of job stream FTA1#PROVA and it fails because it finds a follows dependency from a job FTA1#MY-JOB05 which is still unknown.
2. Then it tries to change the definition of FTA1#COPYOFPROVA and it succeeds.

The second time you run **modify** to change the predecessor job from FTA1#MY-JOB06 to FTA1#MY-JOB05 in job stream FTA1#PROVA, the command is successfully performed since the predecessor job FTA1#MY-JOB05 now exists in the database.

If job stream FTA1#COPYOFPROVA had been listed in the temporary file before FTA1#PROVA, then the **modify** command would have run successfully the first time because the name of the predecessor job would have been modified before changing the dependency definition in the successor job.

For user definitions, if the password field keeps the "*******" value when you exit the editor, the old password is retained. To specify a null password use two consecutive double quotes (“”).

The modify command checks for loop dependencies inside job streams. For example, if job1 follows job2, and job2 follows job1 there is a loop dependency. When a loop dependency inside a job stream is found an error is displayed. The modify command does not check for loop dependencies between job streams because, depending on the complexity of the scheduling activities, this check might be too time and CPU consuming.

Examples

To modify all calendars, run the following command:

```
modify calendars=@
```

To modify job stream sked9 that is launched on workstation site1, run the following command:

```
m sched=site1#sked9
```

To modify all the event rules that include an action with job DPJOB10, run:

```
mod er=@;filter job=DPJOB10
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**
2. Choose one of the following:
   - To modify workstations, click **Scheduling Environment→Design→List Workstations**
   - To modify event rules, click **Workload→Design→List Event Rules**

- To modify all other objects, click **Workload→Design→List Workload Definitions**

3. Select an engine name
4. List the objects that you want to modify using the provided filters, select them, and click **Edit**.

new

Adds a new scheduling object definition in the database.

Authorization

You must have ***add*** access if you add a new scheduling object. If the object already exists in the database you must have ***modify*** access to the object.

Syntax

new
[**calendar** |
domain |
eventrule |
job |
jobstream |
parameter |
prompt |
resource |
user |
vartable |
workstation |
workstation_class]

Arguments

The object you want to define: a calendar, a domain, an event rule, a job, a job stream, a variable, a prompt, a resource, a user, a variable table, a workstation, or a workstation class.

Comments

The command opens a predefined template that helps you edit the object definition and adds it in the database when you save it.

The object templates are located in the `templates` subfolder in the Tivoli Workload Scheduler installation directory. They can be customized to fit your preferences.

Event rule definitions are opened with an XML editor (see "Event rule definition" on page 222 for XML reference and see "The composer editor" on page 236 for details on setting up an XML editor).

While you create a variable, the destination variable table is locked. This implies that, while the table is locked, no other user can run any other locking commands on it.

Examples

To create a new user definition, run:

```
new user
```

To create a new prompt definition, run:

```
new prompt
```

To create a new event rule definition, run:

```
new erule
```

To create a new variable table definition, run:

```
new vartable
```

To create a new variable definition, run:

```
new parameter
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**
2. Choose one of the following:
   - To add workstations, click **Scheduling Environment→Design→Create Workstations**
   - To add event rules, click **Workload→Design→Create Event Rules**
   - To add all other objects, click **Workload→Design→Create Workload Definitions**
3. Select an engine name and click **Go**
4. Do one of the following:
   - For workstations, specify your choices in the properties panel
   - For event rules, specify your choices in the Event Rule Editor panel
   - For all other objects:
     a. Click **New** in the Working List toolbar of the ensuing popup window
     b. Select the type of object that you want to add
     c. Specify your choices in the properties panel

print

This is a synonym for the **`list`** command. See “list” on page 265 for details.

redo

Edits and runs the previous command again.

Note: If the previous command was **authenticate**, **redo** does not display the password specified.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**redo** | **red**}

Context

When you run the **redo** command, composer displays the previous command, so that it can be edited and run again. Use the spacebar to move the cursor under the character to be modified, and enter the following directives.

Directives

d[*dir***]** Deletes the character above the **d**. This can be followed by other directives.

i*text* Inserts text before the character above the **i**.

r*text* Replaces one or more characters with text, beginning with the character above the **r**. Replace is implied if no other directive is entered.

>*text* Appends text to the end of the line.

>d[*dir* | *text***]**
Deletes characters at the end of the line. This can be followed by another directive or text.

>r*text* Replaces characters at the end of the line with text.

Directive Examples

ddd Deletes the three characters above the **d**s.

iabc Inserts **abc** before the character above the **i**.

rabc Replaces the three characters, starting with the one above the **r**, with **abc**.

abc Replaces the three characters above **abc** with **abc**.

d diabc
Deletes the character above the first **d**, skips one character, deletes the character above the second **d**, and inserts **abc** in its place.

>abc Appends **abc** to the end of the line.

>ddabc
Deletes the last two characters in the line, and inserts **abc** in their place.

>rabc Replaces the last three characters in the line with **abc**.

Examples

To insert a character, run the following command:

```
redo
dislay site1#sa@
   ip
display site1#sa@
```

To replace three characters, for example change `site` into `serv` by replacing `ite` with `erv`, run the following command:

```
redo
display site1#sa@
        rerv
display serv1#sa@
```

rename

Renames a scheduling object already existing in the database. The new name must not identify an object already defined in the database.

Authorization

You must have ***delete*** access to the object with the old name and ***add*** access to the object with the new name.

Syntax

{**rename** | **rn**}
{**calendars**|**calendar**|**cal** |
parms|**parm**|**vb** |
vartable|**vt** |
prompts|**prom** |
resorces|**resource**|**res** |
workstation|**ws** |
workstationclass|**wscl** |
domain|**dom** |
jobs|**jobdefinition**|**jd** |
jobsched|**jb** |
eventrule|**erule**|**er**
sched|**jobstream**|**js** |
users|**user** }
old_object_identifier *new_object_identifier*

Arguments

old_object_identifier
: Specifies the old external key identifying the scheduling object, for example calendar name `cal1` as identifier for a defined calendar object to be renamed.

new_object_identifier
: Specifies the new external key identifying the scheduling object, for example calendar name `cal2` as new identifier to be assigned to calendar object previously named `cal1`.

For what concerns jobs, job streams, resources and users both the ***old_object_identifier*** and ***new_object_identifier*** have the following formats:

[*workstationname*#]*jobname*
: The command applies to this job definition. This format is used with the **jobs**|**jobdefinition**|**jd** key.

[*workstationname*#]*jstreamname*
: The command applies to all versions of this job stream. This format is used with the **sched**|**jobstream**|**js** key.

[*workstationname*#]*jstreamname*+**valid from** *date*
: The command applies only to this version of this job stream. This format is used with the **sched**|**jobstream**|**js** key.

[*workstationname***#]***jstreamname.jobname*

The command applies to this job instance defined in this job stream. See the **js** keyword in the "Job stream definition" on page 183 syntax for additional details. This format is used with the **jobsched|jb** key.

[*workstationname***#]***resourcename*

The command applies to this resource definition. This format is used with the **resources|resource|res** key.

[*workstationname***#]***[domain\]username*

The command applies to this user definition. This format is used with the **users|user** key.

For what concerns variables (global parameters):

old_object_identifier

Must be specified in the *tablename.variablename* format. If *tablename* is omitted, composer looks for the variable in the default variable table.

new_object_identifier

Must be specified in the *variablename* format. Adding the table name here generates an error.

Comments

To be renamed the object must be unlocked or locked by the user who issues the rename command.

The variable table containing the variable is locked, while the variable is renamed. This implies that, while the table is locked, no other user can run any other locking commands on it.

If an object named as specified in the *old_object_identifier* field does not exist in the database an error message is displayed.

The use of wild cards is not allowed with this command.

When ***workstationame*** is not specified for objects that have the workstation name as part of their object identifier (for example, job or job stream definitions), the scheduler uses one of the following for ***workstationame***:

- The default workstation specified in the *localopts* file
- The master domain manager if the **composer** command line program is running on a node outside the Tivoli Workload Scheduler network. In this case, in fact, the default workstation set in the *localopts* file is the master domain manager.

The **rename** command is used to assign new names to objects already existing in the database. The new name assigned to an object becomes immediately effective in the database, while it becomes effective in the plan after the **JnextPlan** script is run again. This can lead to incongruences when submitting ad-hoc jobs before generating again the production plan.

Examples

To rename domain object `DOMAIN1` to `DOMAIN2` , run the following command:

```
rename dom=DOMAIN1 DOMAIN2
```

To rename job stream LABJST1 to LABJST2 on workstation CPU1, run the following command:

```
rename js=CPU1#LABJST1 CPU1#LABJST2
```

To rename variable ACCTOLD (defined in table ACCTAB) to ACCTNEW, run the following command:

```
rename parm=ACCTAB.ACCTOLD ACCTNEW
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**
2. Choose one of the following:
   - To rename workstations, click **Scheduling Environment→Design→List Workstations**
   - To rename event rules, click **Workload→Design→List Event Rules**
   - To rename all other objects, click **Workload→Design→List Workload Definitions**
3. Select an engine name
4. List the objects that you want to rename using the provided filters, select them, and click **Create like** (workstations or workload definitions), or **Duplicate** (event rules).
5. Save them with the new name and delete the definitions with the older name.

replace

Replaces scheduling object definitions in the database.

Authorization

You must have ***add*** access if you add a new scheduling object. If the object already exists in the database you must have:

- ***modify*** access to the object if the object is not locked.
- ***modify*** and ***unlock*** accesses to the object if you want to use the **;unlock** option against objects locked by other users.

Syntax

{**replace** | **rep**} *filename* [**;unlock**]

Arguments

filename
: Specifies the name of a file containing the object definitions to replace. The file can contain all types of scheduling objects definition.

unlock
: Updates existing objects previously locked and unlocks them. An error is displayed if the objects are not previously locked. For all new objects inserted, this option, if specified, is ignored.

Comments

The **replace** command is similar to the **add** command, except that there is no confirmation prompt to replace existing objects. For more information, refer to "add" on page 248.

The replace command checks for loop dependencies inside job streams. For example, if `job1` follows `job2`, and `job2` follows `job1` there is a loop dependency. When a loop dependency inside a job stream is found an error is displayed. The replace command does not check for loop dependencies between job streams because, depending on the complexity of the scheduling activities, this check might be too time and CPU consuming.

Examples

To replace the jobs from the file `myjobs`, run the following command:

```
replace myjobs
```

To replace all resources with those contained in the file `myres`, run the following command:

```
rep myres
```

You want to change some existing event rule definitions in the database. You also want to add some new ones as well. You use this command in the following way:

1. You write the entire definitions in an XML file you name `2Q07rules.xml`.
2. You run:
   ```
   rep 2Q07rules.xml
   ```

system command

Runs a system command.

Syntax

[**:** | **!**] *system-command*

Arguments

system-command
: Specifies any valid system command. The prefix of colon (:) or exclamation mark (!) is required only when the command is spelled the same as a composer command.

Examples

To run a **ps** command on UNIX, run the following command:

```
ps -ef
```

To run a **dir** command on Windows, run the following command:

```
dir \bin
```

unlock

Releases access locks on scheduling objects defined in the database. By default to unlock an object, the object must have been locked using the same user and session.

Authorization

You must have the ***unlock*** access to unlock objects locked by other users.

Syntax

{**unlock** | **u**}
{[**calendars** | **calendar** | **cal**=*calname*] |
[**eventrule** | **erule** | **er**=*eventrulename*] |
[**parms** | **parm** | **vb**=[*tablename.*]*variablename*] |
[**vartable** | **vt**=*tablename*] |
[**prompts** | **prom**=*promptname*] |
[**resources** | **resource** | **res**=[*workstationame#*]*resourcename*] |
[**cpu**={*workstationame* | *workstationclassname* | *domainame*}]
[**workstation** | **ws**=*workstationame*] |
[**workstationclass** | **wscl**=*workstationclassname*] |
[**domain** | **dom**=*domainame*] |
[**jobs** | **jobdefinition** | **jd**=[*workstationame#*]*jobname*] |
[**sched**|**jobstream**|**js**= [*workstationame#*]*jstreamname*
*[**valid from** date|**valid to** date |**valid in** date date]*] |
[**users** | **user**=[*workstationame#*]*username*]}
[**;forced**]

Arguments

calendars | calendar | cal
If no argument follows, unlocks all calendar definitions.

If argument *calname* follows, unlocks the *calname* calendar. Wildcard characters are permitted.

eventrule | erule | er
If no argument follows, unlocks all event rule definitions.

If argument *eventrulename* follows, unlocks the *eventrulename* event rule. Wildcard characters are permitted.

parms|parm|vb
If no argument follows, unlocks the default variable table.

If argument *tablename.variablename* follows, unlocks the entire table containing the *variablename* variable. If *tablename* is omitted, unlocks the default variable table. Wildcard characters can be used on both *tablename* and *variablename*. For example:

```
unlock parms=@.@
```

Unlocks all tables.

```
unlock parms=@
```

Unlocks the default table.

```
unlock parms=@.acct@
```

Unlocks all the tables containing the variables whose name starts with `acct`.

```
unlock parms=acct@
```

Unlocks the default table.

Remember: The action on a single variable unlocks the variable table that contains it.

vartable | vt
If no argument follows, unlocks all variable table definitions.

If argument *tablename* variable table follows, unlocks the *tablename* variable table. Wildcard characters are permitted.

prompts | prom
If no argument follows, unlocks all prompt definitions.

If argument *promptname* follows, unlocks the *promptname* prompt. Wildcard characters are permitted.

resources | resource | res
If no argument follows, unlocks all resource definitions.

If argument *workstationame#resourcename* follows, unlocks the *resourcename* resource of the *workstationame* workstation on which the resource is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *resourcename*.

cpu Unlocks workstations, workstation classes, or domains.

workstation
The name of the workstation. Wildcard characters are permitted.

workstationclass
The name of the workstation class. Wildcard characters are permitted.

domain The name of the domain. Wildcard characters are permitted.

workstation | ws
If no argument follows, unlocks all workstation definitions.

If argument *workstationname* follows, unlocks the *workstationname* workstation. Wildcard characters are permitted.

domain | dom
If no argument follows, unlocks all domain definitions.

If argument *domainname* follows, unlocks the *domainname* domain. Wildcard characters are permitted.

workstationclass | wscl
If no argument follows, unlocks all workstation class definitions.

If argument *workstationclassname* follows, unlocks the *workstationclassname* workstation class. Wildcard characters are permitted.

jobs | jobdefinition | jd
If no argument follows, unlocks all job definitions.

If argument *workstationame#jobname* follows, unlocks the *jobname* job of the *workstationame* workstation on which the job runs. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jobname*.

sched | jobstream | js
If no argument follows, unlocks all job stream definitions.

If argument *workstationame#jstreamname* follows, unlocks the *jstreamname* job stream of the *workstationame* workstation on which the job stream is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *jstreamname*.

valid from
date Restricts the selection to job streams that have a *valid from* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid to
date Restricts the selection to job streams that have a *valid to* date equal to the indicated value. The format is *mm/dd/yyyy*.

valid in
date date The time frame during which the job stream can run. The format is *mm/dd/yyyy - mm/dd/yyyy*. One of the two dates can be represented by @.

users | user
If no argument follows, unlocks all user definitions.

If argument *workstationame#username* follows, unlocks the *username* user of the *workstationame* workstation on which the user is defined. If *workstationame* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationame* and *username*.

forced If specified, allows the user who locked the object to unlock it regardless of the session.

If this option is used by the *superuser*, then the **unlock** command can operate regardless to the user and the session used to lock the object.

Comments

If a user, other than the *superuser*, tries to unlock an object that is locked by another user, an error message is returned.

Examples

To unlock job definition JOBDEF1, run the following command:

```
unlock jd=@#JOBDEF1
```

To unlock event rule definition ERJS21, run the following command:

```
unlock erule=ERJS21
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**
2. Choose one of the following:
   - To unlock workstations, click **Scheduling Environment→Design→List Workstations**
   - To unlock event rules, click **Workload→Design→List Event Rules**
   - To unlock all other objects, click **Workload→Design→List Workload Definitions**
3. Select an engine name

4. List the objects that you want to unlock using the provided filters, select them, and click **Unlock**.

validate

Performs the validation of the object definitions contained in a user file.

Authorization

You do not need any specific authorization to objects to run this command.

Syntax

{**validate** | **val**} *filename* **[;syntax]**

Arguments

filename
: Specifies the name of a file that contains calendars, workstations, workstation classes, domains, jobs, parameters, prompts, resources, job streams, event rules, or variable tables. For event rule definitions the file must be in the XML language. See "Event rule definition" on page 222 for details on writing event rule definitions.

syntax Checks the file for syntax errors.

Comments

The output of the **validate** command can be redirected to a file as follows:

```
composer "validate filename" > outfile
```

To include error messages in the output file, use the following:

```
composer "validate filename" > outfile 2>&1
```

Examples

To check the syntax of a file containing workstation definitions, run the following command:

```
validate mycpus;syntax
```

version

Displays the **composer** command line program banner.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

{**version** | **v**}

Examples

To display the **composer** command line program banner, run the following command:

```
version
```

or:

∨

Chapter 10. Managing objects in the plan - conman

The Tivoli Workload Scheduler production plan environment is managed using the **conman** command-line program. The **conman** program is used to start and stop processing, alter and display the `Symphony` production plan, and control workstation linking in a network. It can be used from the master domain manager and from any fault-tolerant agent in the Tivoli Workload Scheduler network. This chapter is divided into the following sections:

Setting up the conman command line program

The **conman** command-line program manages the production plan environment.

You can use the **conman** program from the master domain manager and from any fault-tolerant agent workstation in the Tivoli Workload Scheduler network.

Setting up the conman environment

This section gives you the information about the setup you can choose for your **conman** environment.

Note: On Windows systems, before running **conman** ensure the code-page and fonts are correctly set in the DOS shell to avoid bad character conversion. For additional information on the required settings refer to the *Internationalization notes* section in the *IBM Tivoli Workload Scheduler: Release Notes*.

Terminal output

The output to your computer is determined by the shell variables named ***MAESTROLINES*** and ***MAESTROCOLUMNS***. If either is not set, the standard shell variables, ***LINES*** and ***COLUMNS***, are used. The variables can be set as follows:

MAESTROLINES
: Specifies the number of lines per screen. The default is **24**. At the end of each screen page, **conman** prompts to continue. If ***MAESTROLINES*** (or ***LINES***) is set to zero or a negative number, **conman** does not pause at the end of a page.

  Use of MAESTROLINES is recommended since the LINES variable is a shell operating system variable and in most operating systems it is automatically reset by the operating system itself.

MAESTROCOLUMNS
: Specifies the number of characters per line. The following options are available:
  - Less than 120
  - Equal to or more than 120

MAESTRO_OUTPUT_STYLE
: Specifies how object names are displayed. If set to **LONG**, full names are displayed. If set to any value other than **LONG**, long names are truncated to eight characters followed by a plus sign (+). The default value is LONG.

Offline output

The **;offline** option in **conman** commands is generally used to print the output of a command. When you include it, the following shell variables control the output:

MAESTROLP
: Specifies the destination of a command's output. Set it to one of the following:

  > *file* Redirects output to a file and overwrites the contents of that file. If the file does not exist, it is created.

  >> *file* Redirects output to a file and appends the output to the end of that file. If the file does not exist, it is created.

  | *command*
  Pipes output to a system command or process. The system command is run whether or not output is generated.

  || *command*
  Pipes output to a system command or process. The system command is not run if there is no output.

  The default value for ***MAESTROLP*** is **| lp -tCONLIST** which pipes the command output to the printer and places the title "CONLIST" in the printout's banner page.

MAESTROLPLINES
: Specifies the number of lines per page. The default is **60**.

MAESTROLPCOLUMNS
: Specifies the number of characters per line. The default is **132**.

The variables must be exported before running **conman**.

Selecting the conman prompt on UNIX

The **conman** command prompt is, by default, a percent sign (%). This is defined in the *TWS_home*/`localopts` file. The default command prompt is a dash (-). To select a different prompt, edit the **conman** prompt option in the `localopts` file and change the dash. The prompt can be up to 10 characters long, not including the required trailing pound sign (#).

```
#----------------------------------------------------------------------------
# Custom format attributes
#
date format =              1         # The possible values are 0-ymd, 1-mdy,
2-dmy, 3-NLS.
composer prompt =          -
conman prompt =            %
switch sym prompt =       <n>%
#----------------------------------------------------------------------------
```

Running conman

To configure the environment for using **conman** set the *PATH* and *TWS_TISDIR* variables by running one of the following scripts:

In UNIX:

- `. ./`*`TWS_home`*`/`**`tws_env.sh`** for Bourne and Korn shells

- `. ./`*`TWS_home`*`/`**`tws_env.csh`** for C shells

In Windows:

- *`TWS_home`*`\`**`tws_env.cmd`**

Then use the following syntax to run commands from the **conman** user interface:

conman [*connection_parameters*] [**"***command*[**&**[*command*]...] [**&**]**"**]

where:

connection_parameters
: If you are using **conman** from the master domain manager, the connection parameters were configured at installation and do not need to be supplied, unless you do not want to use the default values.

  If you are using **conman** from the command line client on another workstation, the connection parameters might be supplied by one or more of these methods:
  - Stored in the `localopts` file
  - Stored in the `useropts` file
  - Supplied to the command in a parameter file
  - Supplied to the command as part of the command string

  For an overview of these options see "Setting up options for using the user interfaces" on page 45. For full details of the configuration parameters see the topic on configuring the command-line client access in the *Tivoli Workload Scheduler: Administration Guide*.

You can invoke the **conman** command-line both in *batch* and in *interactive* mode.

When running **conman** in *interactive* mode, you at first launch the **conman** command-line program and then, from the **conman** command-line prompt you run commands a time, for example:

```
conman –username  admin2  –password  admin2pwd
   ss @+state=hold;deps
   dds sked5;needs=2 tapes
```

When running **conman** in *batch* mode, you first launch the **conman** command-line program specifying as input parameter the command to be issued. Once the command is processed, the **conman** command-line program exits, for example

```
conman"sj&sp"
```

When issuing commands from **conman** in batch mode make sure you enclose the commands between double quotes. The following are examples of using batch mode to issue more than one command from within **conman**:

- **conman** runs the **sj** and **sp** commands, and then quits:
  ```
  conman "sj&sp"
  ```
- **conman** runs the **sj** and **sp** commands, and then prompts for a command:
  ```
  conman "sj&sp&"
  ```
- **conman** reads commands from the file **cfile**:
  ```
  conman < cfile
  ```
- commands from the file **cfile** are piped to **conman**:
  ```
  cat cfile | conman
  ```

Note: On Windows workstations, if the User Account Control (UAC) is turned on and the UAC exception list does not contain the `cmd.exe` file, you must open the DOS command prompt shell with the `"Run As Admnistrator"` option to run **conman** on your workstation as a generic user different from Administrator or Tivoli Workload Scheduler user.

Control characters

You can enter the following control characters to interrupt **conman**.

Control+c
: **conman** stops running the current command at the next step that can be interrupted, and returns a command prompt.

Control+d
: **conman** quits after running the current command, on UNIX workstations only.

Running system commands

When you enter a system command using a pipe or a system command prefix (: or !), it is run by a child process. The child process's effective user ID is set to the ID of the user running **conman** to prevent security breaches.

User prompting

When you use wildcard characters to select the objects to be acted upon by a command, **conman** prompts for confirmation after finding each matching object. Responding with **yes** allows the action to be taken, and **no** skips the object without taking the action.

When you run **conman** interactively, the confirmation prompts are issued at your computer. Pressing the **Return** key in response to a prompt is interpreted as a **no** response. Prompting can be disabled by including the **;noask** option in a command.

Although no confirmation prompts are issued when **conman** is not running in interactive mode, it acts as though the response had been **no** in each case, and no objects are acted on. It is important, therefore, to include the **;noask** option on commands when **conman** is not run in interactive mode.

Running commands from conman

conman commands consist of the following elements:

commandname selection arguments

where:

commandname
: Specifies the command name.

selection
: Specifies the object or set of objects to be acted upon.

arguments
: Specifies the command arguments.

The following is an example of a **conman** command:

```
sj sked1(1100 03/05/2006).@+state=hold~priority=0;info;offline
```

where:

sj The abbreviated form of the **showjobs** command.

sked1(1100 03/05/2006).@+state=hold~priority=0
Selects all jobs in the job stream **sked1(1100 03/05/2006)** that are in the HOLD state with a priority other than zero.

;info;offline
Arguments for the **showjobs** command.

Wildcards

The following wildcard characters are permitted:

@ Replaces one or more alphanumeric characters.

? Replaces one alphanumeric character.

% Replaces one numeric character.

Delimiters and special characters

Table 53 lists characters having special meanings in **conman** commands:

Table 53. Delimiters and special characters for conman

Char.	Description
&	Command delimiter. See "Setting up the conman command line program" on page 291.
+	A delimiter used to select objects for commands. It adds an attribute the object must have. For example: `sked1(1100 03/05/2006).@~priority=0`
~	A delimiter used to select objects for commands. It adds an attribute the object must not have. For example: `sked1(1100 03/05/2006).@~priority=0`
;	Argument delimiter. For example: `;info;offline`
,	Repetition and range delimiter. For example: `state=hold,sked,pend`
=	Value delimiter. For example: `state=hold`
:!	Command prefixes that pass the command on to the system. These prefixes are optional; if **conman** does not recognize the command, it is passed automatically to the system. For example: `!ls or :ls`
*	Comment prefix. The prefix must be the first character on a command line or following a command delimiter. For example: `*comment` or `sj& *comment`
>	Redirects command output to a file and overwrites the contents of that file. If the file does not exist, it is created. For example: `sj> joblist`

Table 53. Delimiters and special characters for conman (continued)

Char.	Description
>>	Redirects command output to a file and appends the output to the end of that file. If the file does not exist, it is created. For example: `sj >> joblist`
|	Pipes command output to a system command or process. The system command is run whether or not output is generated. For example: `sj| grep ABEND`
||	Pipes command output to a system command or process. The system command is not run if there is no output. For example: `sj || grep ABEND`

Conman commands processing

The **conman** program performs the commands that change the status of objects, such as start or stop for a workstation, and the commands that modify objects in the plan in an *asynchronous* way. This means that you might notice a delay between the time you submit the command and the time the information stored in the `Symphony` file is updated with the result of the command.

This occurs because the **conman** program does not update the information stored in the `Symphony` file; **conman** submits the commands to **batchman** which is the only process which can access and update the information contained in the `Symphony` file. For this reason you need to wait for **batchman** to process the request of modifying the object issued by **conman** and then to update the information about the object stored in the `Symphony` file before seeing the updated information in the output of the ***showobj*** command.

For example, if you request to delete a dependency using the **conman deldep** command, **conman** submits the **deldep** command by posting an event in the `Mailman.msg` mailbox. The **mailman** process gets the information about the deletion request from `Mailman.msg` and puts it in the `Intercom.msg` mailbox on the workstation that owns the resource you delete the dependency from. On each workstation, **batchman** receives the events in its `Intercom.msg` mailbox and processes them in the same order as they were received. If **batchman** is busy for any reason, events carrying requests to perform **conman** commands continue being queued in the `Intercom.msg` file waiting to be read and processed by **batchman**.

In addition, when **batchman** processes the event, the operator is not notified. As a result, you could delete a dependency and it might appear that it had not been deleted because **batchman** was too busy to immediately perform the requested operation. If you run the command again, the deletion might have already been successful, even though a message saying that the command has been successfully forwarded to **batchman** is displayed in the **conman** prompt.

Selecting jobs in commands

For commands that operate on jobs, the target jobs are selected by means of attributes and qualifiers. The job selection syntax is shown below, and described on the following pages.

Syntax

[*workstation*#] {*jobstreamname*(*hhmm*[*date*]) *job* | *jobnumber*} [{**+** | **~**}*jobqualifier*[...]]

or

[*workstation*#]*jobstream_id*.*job* [{**+** | **~**]*jobqualifier*[...]]**;schedid**

or:

netagent::[*workstation*#] {*jobstream*(*hhmm*[*date*]).*job* | *jobstream_id*.*job***;schedid**}

Arguments

workstation
: When used with *jobstream.job*, this specifies the name of the workstation on which the job stream runs. When used with *jobnumber*, it specifies the workstation on which the job runs. Except when also using `schedid`, wildcard characters are permitted. This argument might be required depending on the workstation where you launch the command, as follows:
- If you launch the command on the workstation where the target jobs have run, the *workstation* argument is optional.
- If you launch the command on a hosted workstation, the *workstation* argument is required. Hosted workstations are:
  - extended agents
  - dynamic agents
  - pools
  - dynamic pools

jobstreamname
: Specifies the name of the job stream in which the job runs. Wildcard characters are permitted.

(*hhmm* **[***date***])**
: Indicates the time and date the job stream instance is located in the preproduction plan. The value *hhmm* corresponds to the value assigned to the **schedtime** keyword in the job stream definition if no **at** time constraint was set. After the job stream instance started processing, the value of ***hhmm* [*date*]** is set to the time the job stream started. The use of wildcards in this field is not allowed. When issuing inline **conman** commands from the shell prompt enclose the **conman** command in double quotes **" "**. For example, run this command as follows:

```
conman "sj my_workstation#my_js(2101 02/23).@"
```

jobstream_id
: Specifies the unique job stream identifier. See "Arguments" on page 305 for more information on job stream identifiers.

schedid Indicates that the job stream identifier is used in selecting the job stream.

jobname
: Specifies the name of the job. Wildcard characters are permitted.

jobnumber
: Specifies the job number.

jobqualifier
: See the following section.

netagent

Specifies the name of the Tivoli Workload Scheduler network agent that interfaces with the remote Tivoli Workload Scheduler network containing the target job. The two colons (::) are a required delimiter. Wildcard characters are permitted. For more information refer to Chapter 18, "Managing internetwork dependencies," on page 549.

Note: Tivoli Workload Scheduler helps you to identify the correct job stream instance when the job stream selection provides an ambiguous result if more than one instance satisfy your selection criteria. For example when more than one instances of `WK1#J1` are included in the production plan and so the job stream selection provides an ambiguous result the following prompt is automatically generated to allow you to choose the correct instance:

```
Process WK1#J1[(0600 03/04/06),(0AAAAAAAAAAAABTB)]
        (enter "y" for yes, "n" for no)?y
Command forwarded to batchman for WK1#J1[(0600 03/04/06),(0AAAAAAAAAAAABTB)]
Process WK1#J1[(1010 03/04/06),(0AAAAAAAAAAAABTC)]
        (enter "y" for yes, "n" for no)?n
```

In the output only the job stream instance scheduled on `(0600 03/04/06)` and with identifier `0AAAAAAAAAAAABTB` is selected to run the command.

Job qualifiers

Job qualifiers specify the attributes of jobs to be acted on by a command. They can be prefixed by + or ~. If a job qualifier is preceded by + then the jobs containing that specific attribute are selected for running the command. If a job qualifier is preceded by ~ then the jobs containing that specific attribute are excluded from running the command.

Job qualifier keywords can be abbreviated to as few leading characters as needed to uniquely distinguish them from each other.

at[=*time* **|** *lowtime,* **|** *,hightime* **|** *lowtime,hightime* **]**

Selects or excludes jobs based on the time specified in the **at** dependency.

time

Specifies the time as follows:

hhmm[**+***n* **days** | *date*] [**timezone|tz** *tzname*]

where:

hhmm The hour and minute.

+*n* **days**
The next occurrence of *hhmm* in *n* number of days.

date The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*
The name of the time zone of the job. See Chapter 16, "Managing time zones," on page 531 for valid names.

time conforms to the following rules:

- When *hhmm* is earlier than the current time, the start time is tomorrow; when *hhmm* is later than the current time, the start time is today.

- When *hhmm* is greater than 2400, it is divided by 2400. Of the division result, the whole part represents the number of + *days*, while the decimal part represents the time.

lowtime
: Specifies the lower limit of a time range, expressed in the same format as *time*. Jobs are selected that are scheduled to start after this time.

hightime
: Specifies the upper limit of a time range, expressed in the same format as *time*. Jobs are selected that are scheduled to start before this time.

If **at** is used alone and it is preceded by + then the jobs selected are those containing an **at** dependency.

If **at** is used alone and it is preceded by ~ then the jobs selected are those not containing an **at** dependency.

confirmed
: Selects or excludes jobs that were scheduled using the **confirm** keyword.

critical
: Selects or excludes jobs that were flagged with the **critical** keyword in a job stream definition.

critnet Selects or excludes jobs that are flagged as **critical** or are predecessors of critical jobs. Hence, it applies to all the jobs that have a critical start time set.

The critical start time of a critical job is calculated by subtracting its estimated duration from its deadline. The critical start time of a predecessor is calculated by subtracting its estimated duration from the critical start time of its successor. Within a critical network, critical start times are calculated starting from the critical job and working backwards along the line of its predecessors.

deadline[**=***time* | *lowtime***,** | **,***hightime* | *lowtime***,***hightime*]
: Specifies the time within which a job must complete.

hhmm[**+***n* **days** | *date*] [**timezone**|**tz** *tzname*]

hhmm The hour and minute.

+*n* **days**
: An offset in days from the scheduled deadline time.

date The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|**tz** *tzname*
: Specifies the time zone to be used when computing the deadline. See Chapter 16, “Managing time zones,” on page 531 for time zone names. The default is the time zone of the workstation on which the job or job stream is launched.

lowtime
: Specifies the lower limit of a time range, expressed in the same format as *time*. Selected jobs have a scheduled deadline not earlier than this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Selected jobs have a scheduled deadline not later than this time.

every[=*rate* **|** *lowrate***,** **|** **,***highrate* **|** *lowrate***,***highrate***]**

Selects or excludes jobs based on whether or not they have a repetition rate.

rate Specifies the scheduled run rate, expressed as hours and minutes (*hhmm*).

lowrate Specifies the lower limit of a rate range, expressed in the same format as *rate*. Jobs are selected that have repetition rates equal to or greater than this rate.

highrate

Specifies the upper limit of a rate range, expressed in the same format as *rate*. Jobs are selected that have repetition rates less than or equal to this rate.

If **every** is used alone and it is preceded by + then the jobs selected are those containing any repetition rate.

If **every** is used alone and it is preceded by ~ then the jobs selected are those not containing any repetition rate.

finished[=*time* **|** *lowtime***,** **|** **,***hightime* **|** *lowtime***,***hightime***]**

Selects or excludes jobs based on whether or not they have finished.

time Specifies the exact time the job finished, expressed as follows:

hhmm [*date*] [**timezone|tz** *tzname*]

hhmm The hour and minute.

date The next occurrence of *hhmm* on date, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*

The name of the time zone of the job. See Chapter 16, "Managing time zones," on page 531 for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Jobs are selected that finished at or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Jobs are selected that finished at or before this time.

If **finished** is used alone and it is preceded by + then the jobs selected are the jobs that have finished running.

If **finished** is used alone and it is preceded by ~ then the jobs selected are the jobs that have not finished running.

follows=[*netagent***::][***workstation***#]{***jobstreamname***(***hhmm* **[***mm***/***dd***[/***yy***]])[.***job* **|** **@]** **|** *jobstream_id***.***job***;schedid}|** *job***[;nocheck][,...]**

Selects or excludes jobs based on whether or not they have a `follows` dependency.

netagent

Specifies the name of the Tivoli Workload Scheduler network agent that interfaces with the remote Tivoli Workload Scheduler network containing the prerequisite job. Wildcard characters are permitted. For more information refer to Chapter 18, "Managing internetwork dependencies," on page 549.

workstation
: Specifies the name of the workstation on which the prerequisite job runs. Wildcard characters are permitted.

jobstreamname
: Specifies the name of the job stream in which the prerequisite job runs. Wildcard characters are permitted. If you enter *jobstreamname***.@**, you specify that the target job follows all jobs in the job stream.

jobname
: Specifies the name of the prerequisite job. When entered without a *jobstreamname*, it means that the prerequisite job is in the same job stream as the target job. Do not specify the job name using wildcard characters for a follows dependency.

jobstream_id
: Specifies the unique job stream identifier. See "Arguments" on page 305 for more information on job stream identifiers.

schedid
: This keyword, if present, applies to all the job streams identifiers specified in the clause and indicates that for all the job streams specified you are using the *jobstream_id*s and not the *jobstreamname*s. If you want to select some job streams using the *jobstream_id* and some job streams using the *jobstreamname*, you must use two different **follows** clauses, one containing the job streams identified by the *jobstreamname* without the **schedid** keywords, and the other containing the job streams identified by the *jobstream_id* with the **schedid** keyword.

nocheck
: Is valid only for the submission commands and used in conjunction with the**schedid** keyword. The **nocheck** keyword indicates that **conman** does not have to check for the existence of the prerequisite job *jobstream_id.job* in the `Symphony` file. It is assumed that *jobstream_id.job* exists, in case it does not exist **batchman** prints a warning message in the `stdlist`.

If **follows** is used alone and it is preceded by + then the jobs are selected if they contain `follows` dependencies.

If **follows** is used alone and it is preceded by ~ then the jobs are selected if they contain no `follows` dependency.

logon=*username*
: Select jobs based on the user names under which they run. If *username* contains special characters it must be enclosed in quotes ("). Wildcard characters are permitted.

needs[=[*workstation***#]***resourcename***]**
: Selects or excludes jobs based on whether or not they have a resource dependency.

workstation
: Specifies the name of the workstation on which the resource is defined. Wildcard characters are permitted.

resourcename
: Specifies the name of the resource. Wildcard characters are permitted.

If **needs** is used alone and it is preceded by + then the jobs are selected if they contain `resource` dependencies.

If **needs** is used alone and it is preceded by ~ then the jobs are selected if they contain no `resource` dependency.

opens[=[*workstation***#]***filename***[(***qualifier***)]]**
Select jobs based on whether or not they have a `file` dependency. A `file` dependency occurs when a job or job stream is dependent on the existence of one or more files before it can begin running.

workstation
Specifies the name of the workstation on which the file exists. Wildcard characters are permitted.

filename
Specifies the name of the file. The name must be enclosed in quotes (") if it contains special characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted.

qualifier
A valid test condition. If omitted, jobs are selected or excluded without regard to a qualifier.

If **opens** is used alone and it is preceded by + then the jobs are selected if they contain `file` dependencies.

If **opens** is used alone and it is preceded by ~ then the jobs are selected if they contain no `file` dependency.

priority=*pri* **|** *lowpri***,** **|** **,***highpri* **|** *lowpri***,***highpri*
Selects or excludes jobs based on their priorities.

pri Specifies the priority value. You can enter **0** through **99**, **hi** or **go**.

lowpri Specifies the lower limit of a priority range. Jobs are selected with priorities equal to or greater than this value.

highpri Specifies the upper limit of a priority range. Jobs are selected with priorities less than or equal to this value.

prompt[=*promptname* **|** *msgnum***]**
Selects or excludes jobs based on whether or not they have a `prompt` dependency.

promptname
Specifies the name of a global prompt. Wildcard characters are permitted.

msgnum
Specifies the message number of a local prompt.

If **prompt** is used alone and it is preceded by + then the jobs are selected if they contain `prompt` dependencies.

If **prompt** is used alone and it is preceded by ~ then the jobs are selected if they contain no `prompt` dependency.

recovery=*recv-option*
Selects or excludes jobs based on their recovery options.

recv-option
: Specifies the job recovery option as **stop**, **continue**, or **rerun**.

scriptname=*filename*
: Selects or excludes jobs based on their executable file names.

  filename
  : Specifies the name of an executable file. The name must be enclosed in quotes (") if it contains special characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted.

started[=*time* **|** *lowtime***,** **|** **,***hightime* **|** *lowtime***,***hightime***]**
: Selects or excludes jobs based on whether or not they have started.

  time
  : Specifies the exact time the job started, expressed as follows:

    hhmm [*date*] [**timezone|tz** *tzname*]

    hhmm
    : The hour and minute.

    date
    : The next occurrence of *hhmm* on date, expressed as *mm/dd[/yy]*.

    timezone|tz *tzname*
    : The name of the time zone of the job. See Chapter 16, "Managing time zones," on page 531 for valid names.

  lowtime
  : Specifies the lower limit of a time range, expressed in the same format as *time*. Only jobs that started at or after this time are selected.

  hightime
  : Specifies the upper limit of a time range, expressed in the same format as *time*. Only jobs that started at or before this time are selected.

  If **started** is used alone and it is preceded by +, then only the jobs that have started running at this time are selected.

  If **started** is used alone and it is preceded by ~, then only the jobs that have started running at or after this time and that are still running are selected.

state=*state***[,...]**
: Selects or excludes jobs based on their states.

  state
  : Specifies the current state of the job. Valid job states are as follows:

    ABEND
    : The job ended with a nonzero exit code.

    ABENP
    : An **abend** confirmation was received, but the job has not completed.

    ADD
    : The job is being submitted.

    DONE
    : The job completed in an unknown state.

    ERROR
    : For internetwork dependencies only, an error occurred while checking for the remote status.

EXEC The job is running. The + flag written beside the EXEC status means that the job is managed by the local **batchman** process.

EXTRN
For internetwork dependencies only, the status is unknown. An error occurred, a rerun action was just performed on the job in the `EXTERNAL` job stream, or the remote job or job stream does not exist.

FAIL Unable to launch the job.

FENCE
The job's priority value is below the fence.

HOLD
The job is awaiting dependency resolution.

INTRO
The job is introduced for launching by the system. The + flag written beside the INTRO status means that the job is managed by the local **batchman** process.

PEND The job completed, and is awaiting confirmation.

READY
The job is ready to launch, and all dependencies are resolved.

SCHED
The job's **at** time has not arrived.

SUCC The job completed with an exit code of zero.

SUCCP
A **succ** confirmation was received, but the job has not completed.

WAIT The job is in the WAIT state (extended agent only).

until[=*time* **|** *lowtime***,** **|** **,***hightime* **|** *lowtime***,***hightime* **]**
Selects or excludes jobs based on their scheduled end time.

time Specifies the scheduled end time expressed as follows:

hhmm[+*n* **days** | *date*] [**timezone|tz** *tzname*]

hhmm The hour and minute.

+*n* **days**
The next occurrence of *hhmm* in *n* number of days.

date The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*
The name of the time zone of the job. See Chapter 16, "Managing time zones," on page 531 for valid names.

lowtime
Specifies the lower limit of a time range, expressed in the same format as *time*. Jobs are selected that have scheduled end times on or after this time.

hightime
: Specifies the upper limit of a time range, expressed in the same format as *time*. Jobs are selected that have scheduled end times on or before this time.

If **until** is used alone and it is preceded by + then the jobs are selected if they have an **until** time specified.

If **until** is used alone and it is preceded by ~ then the jobs are selected if they have no **until** time specified.

Selecting job streams in commands

For commands that operate on job streams, the target job streams are selected by specifying attributes and qualifiers.

Because ***scheddateandtime*** is specified in minutes, the combination of the **jobstreamname** and the *scheddateandtime* time might not be unique. For this reason the **jobstream_id** has been made available to the user, either for display purposes or to perform actions against a specific instance of a job stream.

The job stream selection syntax is shown below, and described on the following pages. You can choose one of the two syntaxes described.

Syntax

[*workstation*#]*jobstreamname*(*hhmm*[*date*]) [{+ | ~}*jobstreamqualifier*[...]]

[*workstation*#]*jobstream_id* **;schedid**

Arguments

workstation
: Specifies the name of the workstation on which the job stream runs. Except when also using `schedid`, wildcard characters are permitted.

jobstreamname
: Specifies the name of the job stream. Wildcard characters are permitted.

(*hhmm* **[***date***])**
: Indicates the time and date the job stream instance is located in the preproduction plan. This value corresponds to the value assigned to the **schedtime** keyword in the job stream definition if no **at** time constraint was set. After the job stream instance started processing the value of ***hhmm*** **[*date*]** is set to the time the job stream started. The use of wildcards in this field is not allowed. When issuing in line **conman** commands from the shell prompt enclose the **conman** command in double quotes **" "**. For example, run this command as follows:

```
conman "ss my_workstation#my_js(2101 02/23)"
```

jobstreamqualifier
: See "Job Stream Qualifiers" below.

jobstream_id
: Specifies the unique alphanumerical job stream identifier automatically generated by the planner and assigned to that job stream. It is mainly used by internal processes to identify that instance of the job stream within the

production plan, but it can often be used also when managing the job stream from the **conman** command-line program by specifying the **;schedid** option.

schedid

Indicates that the job stream identifier is used in selecting the job stream.

Note: Tivoli Workload Scheduler helps you to identify the correct job stream instance when the job stream selection provides an ambiguous result if more than one instance satisfy your selection criteria. For example when more than one instances of WK1#J1 are included in the production plan and so the job stream selection provides an ambiguous result the following prompt is automatically generated to allow you to choose the correct instance:

```
Process WK1#J1[(0600 03/04/06),(0AAAAAAAAAAAAABTB)]
       (enter "y" for yes, "n" for no)?y
Command forwarded to batchman for WK1#J1[(0600 03/04/06),(0AAAAAAAAAAAAABTB)]
Process WK1#J1[(1010 03/04/06),(0AAAAAAAAAAAAABTC)]
       (enter "y" for yes, "n" for no)?n
```

In the output only the job stream instance scheduled on (0600 03/04/06) and with identifier 0AAAAAAAAAAAAABTB is selected to run the command.

Job stream qualifiers

at[=*time* **|** *lowtime***, | ,***hightime* **|** *lowtime***,***hightime* **]**

Selects or excludes job streams based on the scheduled start time.

time Specifies the scheduled start time expressed as follows:

hhmm[+*n* **days** | *date*] [**timezone|tz** *tzname*]

hhmm The hour and minute.

+*n* **days**

The next occurrence of *hhmm* in *n* number of days.

date The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*

The name of the time zone of the job stream. See Chapter 16, “Managing time zones,” on page 531 for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Job streams are selected that have scheduled start times on or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Job streams are selected that have scheduled start times at or before this time.

If **at** is used alone and it is preceded by + then the job streams selected are those containing an **at** dependency.

If **at** is used alone and it is preceded by ~ then the job streams selected are those not containing an **at** dependency.

carriedforward

Selects job streams that were carried forward if preceded by +, excludes job streams that were carried forward if preceded by ~.

carryforward

If preceded by + selects job streams that were scheduled using the **carryforward** keyword; if preceded by ~ excludes job streams that were scheduled using the **carryforward** keyword.

finished[=*time* **|** *lowtime***,** **|** **,***hightime* **|** *lowtime***,***hightime***]**

Selects or excludes job streams based on whether or not they have finished.

time Specifies the exact time the job streams finished, expressed as follows:

hhmm [*date*] [**timezone|tz** *tzname*]

hhmm The hour and minute.

date The next occurrence of *hhmm* on date, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*

The name of the time zone of the job stream. See Chapter 16, "Managing time zones," on page 531 for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Job streams are selected that finished at or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Job streams are selected that finished at or before this time.

If **finished** is used alone and it is preceded by + then the jobs streams selected are the jobs that have finished running.

If **finished** is used alone and it is preceded by ~ then the jobs streams selected are the jobs that have not finished running.

follows=[*netagent***::][***workstation***#]{***jobstreamname***(***hhmm* **[***mm***/***dd***[/***yy***]])[.***job* **| @] |** *jobstream_id***.***job***;schedid}|** *job***[;nocheck] [,...]**

Selects or excludes job streams based on whether or not they have a `follows` dependency.

netagent

Specifies the name of the network agent that interfaces with the remote Tivoli Workload Scheduler network containing the prerequisite job or job stream. Wildcard characters are permitted. For more information about network agents, refer to Chapter 18, "Managing internetwork dependencies," on page 549.

workstation

Specifies the name of the workstation on which the prerequisite job or job stream runs. Wildcard characters are permitted.

jobstreamname

Specifies the name of the prerequisite job stream. Wildcard characters are permitted.

jobname

Specifies the name of the prerequisite job. Wildcard characters are permitted.

jobstream_id
Specifies the unique job stream identifier. See "Arguments" on page 305 for more information on job stream identifiers.

schedid
This keyword, if present, applies to all the job streams identifiers specified in the clause and indicates that for all the job streams specified you are using the *jobstream_id*s and not the *jobstreamname*s. If you want to select some job streams using the *jobstream_id* and some job streams using the *jobstreamname*, you must use two different **follows** clauses, one containing the job streams identified by the *jobstreamname* without the **schedid** keywords, and the other containing the job streams identified by the *jobstream_id* with the **schedid** keyword.

nocheck
Is valid only for the submission commands and used in conjunction with the**schedid** keyword. The **nocheck** keyword indicates that **conman** does not have to check for the existence of the prerequisite job *jobstream_id.job* in the `Symphony` file. It is assumed that *jobstream_id.job* exists, in case it does not exist **batchman** prints a warning message in the `stdlist`.

If **follows** is used alone and it is preceded by + then the jobs streams are selected if they contain `follows` dependencies.

If **follows** is used alone and it is preceded by ~ then the jobs streams are selected if they contain no `follows` dependency.

limit[=*limit* **|** *lowlimit*, **|** ,*highlimit* **|** *lowlimit*,*highlimit***]**
Selects or excludes job streams based on whether or not they have a job limit.

limit Specifies the exact job limit value.

lowlimit
Specifies the lower limit of range. Job streams are selected that have job limits equal to or greater than this limit.

highlimit
Specifies the upper limit of a range. Job streams are selected that have job limits less than or equal to this limit.

If **limit** is used alone and it is preceded by + then the jobs streams are selected if they have any job limit.

If **limit** is used alone and it is preceded by ~ then the jobs streams are selected if they have no job limit.

needs[=[*workstation***#]***resourcename***]**
Selects or excludes job streams based on whether or not they have a resource dependency.

workstation
Specifies the name of the workstation on which the resource is defined. Wildcard characters are permitted.

resourcename
Specifies the name of the resource. Wildcard characters are permitted.

If **needs** is used alone and it is preceded by + then the jobs streams are selected if they contain `resource` dependencies.

If **needs** is used alone and it is preceded by ~ then the jobs streams are selected if they contain no `resource` dependency.

opens[**=**[*workstation***#**]*filename*[**(***qualifier***)**]]
: Selects or excludes job streams based on whether or not they have a file dependency. A file dependency occurs when a job or job stream is dependent on the existence of one or more files before it can begin running.

workstation
: Specifies the name of the workstation on which the file exists. Wildcard characters are permitted.

filename
: Specifies the name of the file. The name must be enclosed in quotes (") if it contains special characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted.

qualifier
: A valid test condition. If omitted, job streams are selected or excluded without regard to a qualifier.

If **opens** is used alone and it is preceded by + then the jobs streams are selected if they contain `file` dependencies.

If **opens** is used alone and it is preceded by ~ then the jobs streams are selected if they contain no `file` dependency.

priority=*pri* | *lowpri***,** | **,***highpri* | *lowpri***,***highpri*
: Selects or excludes job streams based on their priorities.

pri Specifies the priority value. You can enter **0** through **99**, **hi** or **go**.

lowpri Specifies the lower limit of a priority range. Job streams are selected with priorities equal to or greater than this value.

highpri Specifies the upper limit of a priority range. Job streams are selected with priorities less than or equal to this value.

prompt[**=***promptname* | *msgnum*]
: Selects or excludes job streams based on whether or not they have a prompt dependency.

promptname
: Specifies the name of a global prompt. Wildcard characters are permitted.

msgnum
: Specifies the message number of a local prompt.

If **prompt** is used alone and it is preceded by + then the jobs streams are selected if they contain `prompt` dependencies.

If **prompt** is used alone and it is preceded by ~ then the jobs streams are selected if they contain no `prompt` dependency.

started[**=***time* | *lowtime***,** | **,***hightime* | *lowtime***,***hightime*]
: Selects or excludes job streams based on whether or not they have started.

time Specifies the exact time the job stream started, expressed as follows:

hhmm [*date*] [**timezone**|**tz** *tzname*] *hhmm* The hour and minute.

date The next occurrence of *hhmm* on date, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*
The name of the time zone of the job stream. See Chapter 16, "Managing time zones," on page 531 for valid names.

lowtime
Specifies the lower limit of a time range, expressed in the same format as *time*. Job streams are selected that started at or after this time.

hightime
Specifies the upper limit of a time range, expressed in the same format as *time*. Job streams are selected that started at or before this time.

If **started** is used alone and it is preceded by + then the jobs streams selected are the jobs that have started running.

If **started** is used alone and it is preceded by ~ then the jobs streams selected are the jobs that have not started running.

state=*state***[,...]**
Selects or excludes job streams based on their states.

state Specifies the current state of the job stream. Valid job stream states are as follows:

ABEND
The job stream ended abnormally.

ADD The job stream has just been submitted.

EXEC The job stream is running.

HOLD
The job stream is awaiting dependency resolution.

READY
The job stream is ready to launch, and all dependencies are resolved.

STUCK
Execution is interrupted. No jobs are launched without operator intervention.

SUCC The job stream completed successfully.

until[=*time* | *lowtime***,** | **,***hightime* | *lowtime***,***hightime* **]**
Selects or excludes job streams based on the scheduled end time.

time Specifies the scheduled end time expressed as follows:

hhmm[**+***n* **days** | *date*] [**timezone|tz** *tzname*]

hhmm The hour and minute.

+*n* **days**
The next occurrence of *hhmm* in *n* number of days.

date The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

timezone|tz *tzname*

The name of the time zone of the job stream. See Chapter 16, “Managing time zones,” on page 531 for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Job streams are selected that have scheduled end times on or after this time.

hightime

Specifies the upper limit of a time range, expressed in the same format as *time*. Job streams are selected that have scheduled end times on or before this time.

If **until** is used alone and it is preceded by + then the jobs streams selected are those containing any scheduled end time.

If **until** is used alone and it is preceded by ~ then the jobs streams selected are those not containing any scheduled end time.

Managing jobs and job streams from back-level agents

The change in the job stream instance naming convention introduced with Tivoli Workload Scheduler version 8.3 requires you to apply the following workaround when issuing command-line commands against a plan generated on a Tivoli Workload Scheduler version 8.3 (or later) master domain manager from Tivoli Workload Scheduler version 8.1, 8.2, or 8.2.1 agents:

- You must use the @ symbol as the first character for the job stream instance identifier. For example the job stream running on `CPU1` workstation with identifier `0AAAAAAAAAAAAY3` must be identified in the **conman** command line as follows:

  ```
  CPU1#@AAAAAAAAAAAAY3
  ```
- You cannot use the **follows** keyword when adding a dependency to a job or a job stream or when submitting as a job a command or a file.
- You cannot use the **into** keyword to specify the job stream where the job must be added when submitting as a job a command or a file.

For example, to display the information about the job `job2` contained in the job stream instance with identifier `0AAAAAAAAAAAAT1` running on `CPU1` workstation, run the following command on Tivoli Workload Scheduler version 8.1, 8.2, or 8.2.1 agents:

```
sj CPU1#@AAAAAAAAAAAAT1.job2
```

These changes will also be seen in reports and logs, and any other places where job stream names are printed or displayed.

Conman commands

Table 54 on page 312 lists the **conman** commands. Command names and keywords can be entered in either uppercase or lowercase characters, and can be abbreviated to as few leading characters as are needed to uniquely distinguish them from each other. Some of the command names also have specific short forms.

Note: The workstation types in the following table have these meanings:
- M - master domain managers and backup masters
- F - domain managers and fault-tolerant agents

- T - for fault-tolerant agents
- S - standard agents (you can only display files on a standard agent)

s

Table 54. List of conman commands

Command	Short Form	Description	Type	Page
adddep { job | sched }	**adj | ads**	Adds job or job stream dependencies.	F	"adddep job" on page 314 "adddep sched" on page 316
altpass		Alters a user object definition password.	F	"altpass" on page 317
altpri	**ap**	Alters job or job stream priorities.	F	"altpri" on page 318
bulk_discovery	**bulk**	Performs a bulk discovery. For use with IBM Tivoli Monitoring 6.1 (Tivoli Enterprise Portal).	F	"bulk_discovery" on page 319
cancel { job | sched }	**cj | cs**	Cancels a job or a job stream.	F	"cancel job" on page 319 "cancel sched" on page 321
checkhealthstatus	**chs**	Invokes **chkhltst** service to check if mailbox can be successfully read by **mailman** or if there are errors in the mailbox header.	MFS	"checkhealthstatus" on page 322
confirm	**conf**	Confirms job completion.	F	"confirm" on page 322
console	**cons**	Assigns the Tivoli Workload Scheduler console.	FS	"console" on page 324
continue	**cont**	Ignores the next error.	FS	"continue" on page 325
deldep { job | sched }	**ddj | dds**	Deletes job or job stream dependencies.	F	"deldep job" on page 325 "deldep sched" on page 326
deployconf	**deploy**	Gets the latest monitoring configuration for the event monitoring engine on the workstation.	FS	"deployconf" on page 328
display { file | job | sched }	**df | dj | ds**	Displays files, jobs, and job streams.	FS	"display" on page 328
exit	**e**	Exits **conman**.	FS	"exit" on page 331
fence	**f**	Sets Tivoli Workload Scheduler job fence.	F	"fence" on page 331
help(1)	**h**	Displays command information.	FS	"help" on page 332
kill	**k**	Stops an executing job.	F	"kill" on page 333
limit { cpu | sched }	**lc | ls**	Changes a workstation or job stream job limit.	F	"limit cpu" on page 334 "limit sched" on page 335
link	**lk**	Opens workstation links.	FS	"link" on page 336
listsym	**lis**	Displays a list of `Symphony` log files.	F	"listsym" on page 338
recall	**rc**	Displays prompt messages.	F	"recall" on page 340
redo	**red**	Edits the previous command.	FS	"redo" on page 341
release { job | sched }	**rj | rs**	Releases job or job stream dependencies.	F	"release job" on page 342 "release sched" on page 343
reply	**rep**	Replies to prompt message.	F	"reply" on page 345

Table 54. List of conman commands (continued)

Command	Short Form	Description	Type	Page
rerun	**rr**	Reruns a job.	F	"rerun" on page 346
resetFTA	**N/A**	Recovers a corrupt Symphony file on the specified fault-tolerant agent	T	"resetFTA" on page 348
resource	**res**	Changes the number of resource units.	F	"resource" on page 349
setsym	**set**	Selects a Symphony log file.	F	"setsym" on page 350
showcpus	**sc**	Displays workstation and link information.	FS	"showcpus" on page 351
showdomain	**showd**	Displays domain information.	FS	"showdomain" on page 357
showfiles	**sf**	Displays information about files.	F	"showfiles" on page 358
showjobs	**sj**	Displays information about jobs.	F	"showjobs" on page 360
showprompts	**sp**	Displays information about prompts.	F	"showprompts" on page 375
showresources	**sr**	Displays information about resources.	F	"showresources" on page 378
showschedules	**ss**	Displays information about job streams.	F	"showschedules" on page 380
shutdown	**shut**	Stops Tivoli Workload Scheduler production processes.	FS	"shutdown" on page 384
start		Starts Tivoli Workload Scheduler production processes.	FS	"start" on page 385
startappserver		Starts the embedded WebSphere Application Server process	FS	"startappserver" on page 387
starteventprocessor	**startevtp**	Starts the event processing server.	M(2)	"starteventprocessor" on page 388
startmon	**startm**	Starts the `monman` process that turns on the event monitoring engine on the agent.	FS	"startmon" on page 389
status	**stat**	Displays Tivoli Workload Scheduler production status.	FS	"status" on page 389
stop		Stops Tivoli Workload Scheduler production processes.	FS	"stop" on page 390
stop ;progressive		Stops Tivoli Workload Scheduler production processes hierarchically.		"stop ;progressive" on page 392
stopappserver	**stopapps**	Stops the embedded WebSphere Application Server process	FS	"stopappserver" on page 393
stopeventprocessor	**stopevtp**	Stops the event processing server.	M(2)	"stopeventprocessor" on page 394
stopmon	**stopm**	Stops the event monitoring engine on the agent.	FS	"stopmon" on page 395
submit { docommand | file | job | sched }	**sbd | sbf | sbj | sbs**	Submits a command, file, job, or job stream.	FS(3)	"submit docommand" on page 396 "submit file" on page 399 "submit job" on page 402 "submit sched" on page 405

Table 54. List of conman commands (continued)

Command	Short Form	Description	Type	Page
switcheventprocessor	**switchevtp**	Switches the event processing server from master domain managers to backup masters or vice versa.	M	"switcheventprocessor" on page 409
switchmgr	**switchm**	Switches the domain manager.	F	"switchmgr" on page 410
system-command		Sends a command to the system.	FS	"system command" on page 411
tellop	**to**	Sends a message to the console.	FS	"tellop" on page 411
unlink		Closes workstation links.	FS	"unlink" on page 412
version	**v**	Displays **conman**'s command line program banner.	FS	"version" on page 414

1. Not available on supported Windows operating system.
2. Includes workstations installed as backup masters but used as ordinary fault-tolerant agents.
3. You can use **submit job** (**sbj**) and **submit sched** (**sbs**) on a standard agent by using the connection parameters or specifying the settings in the `useropts` file when invoking the **conman** command line.

Note: In the commands, the terms *sched* and *schedule* refer to *job streams*, and the term *CPU* refers to *workstation*.

adddep job

Adds dependencies to a job.

You must have ***adddep*** access to the job. To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

Syntax

{**adddep job** | **adj**} *jobselect*
 ;*dependency*[**;**...]
 [**;noask**]

Arguments

jobselect
: See "Selecting jobs in commands" on page 296.

dependency
: The type of dependency. Specify one of the following. Wildcard characters are not permitted.

at=*hhmm*[**timezone** | **tz** *tzname*][+*n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

confirmed

deadline=*time* [**timezone**|**tz** *tzname*][+*n* **day**[**s** | *mm/dd[/yy]*]

every=*rate*

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job***;schedid**}| *job*[**,**...]

needs=[*num*] [*workstation***#**]*resource*[**,**...]

opens=[*workstation***#**]"*filename*"[**(***qualifier***)**][**,**...] **priority**[**=***pri* | **hi** | **go**]

prompt="[**:** | **!**]*text***"** | *promptname*[**,**...]

until *time* [**timezone|tz** *tzname*][**+***n* **day**[**s**]] | [**absolute** | **abs**] [**;onuntil** *action*]

noask Specifies not to prompt for confirmation before taking action on each qualifying job.

Note:

1. If you add twice a dependency on a job stream to a job, both dependencies are treated.
2. When using the **deadline** keyword, ensure the **bm check deadline** option is set to a value higher than 0 in the `localopts` configuration file on the workstation you are working on. You can define the **bm check deadline** option on each workstation on which you want to be aware of the deadline expiration, or, if you want to obtain up-to-date information about the whole environment, define the option on the master domain manager. Deadlines for critical jobs are evaluated automatically, independently of the **bm check deadline** option. For more information about the **bm check deadline** option, see Localopts details.

Comments

If you do not specify a value for `priority`, the job reverts to its original scheduled priority. If you do not specify a workstation in `follows`, `needs`, or `opens`, the default is the workstation on which the job runs.

You cannot use this command to add a `resource` or a `prompt` as dependencies unless they are already referenced by a job or a job stream in the `Symphony` file.

Examples

To add a resource dependency to job `job3` in job stream `sked9(0900 02/19/06)`, run the following command:

```
adj sked9(0900 02/19/06).job3 ; needs=2 tapes
```

To add an external follows dependency from to job `JOB022` in job stream `MLN#SCHED_02(0600 02/24)` to `JOBA` in job stream `MLN#NEW_TEST(0900 02/19/06)`, run the following command:

```
adj MLN#NEW_TEST(0900 02/19/06).JOBA ; follows MLN#SCHED_02(0600 02/24/06).JOB022
```

To add a file dependency, and an **until** time to job `j6` in job stream `JS2(0900 02/19/06)`, run the following command:

```
adj WK1#JS2(0900 02/19/06).j6 ; opens="/usr/lib/prdata/file5"(-s %p) ; until=2330
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Workload**→**Monitor**→**Monitor Jobs**
2. Select **All Jobs in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job and click the **Dependencies...** tab.

adddep sched

Adds dependencies to a job stream.

You must have ***adddep*** access to the job stream. To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

Syntax

{**adddep sched** | **ads**} *jstreamselect*
  ;*dependency*[**;**...]
  [**;noask**]

Arguments

jstreamselect
: See "Selecting job streams in commands" on page 305.

dependency
: The type of dependency. Specify one of the following. Wildcard characters are not permitted.

at=*hhmm*[**timezone** | **tz** *tzname*][+*n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

carryforward

deadline=*time* [**timezone|tz** *tzname*][+*n* **day**[**s** | *mm/dd[/yy]*]

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | **@**] | *jobstream_id*.*job***;schedid**}| *job*[**,**...]

limit=*limit*

needs=[*num*] [*workstation***#**]*resource*[**,**...]

opens=[*workstation***#**]"*filename*"[**(***qualifier***)**][**,**...] **priority**[=*pri* | **hi** | **go**]

prompt="[**:** | **!**]*text***"** | *promptname*[**,**...]

until *time* [**timezone|tz** *tzname*][+*n* **day**[**s**] | [**absolute** | **abs**]] [**;onuntil** *action*]

noask Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Note:

1. If you add twice a dependency on a job stream to another job stream, only one dependency is considered.
2. When using the **deadline** keyword, ensure the **bm check deadline** option is set to a value higher than 0 in the `localopts` configuration file on the workstation you are working on. You can define the **bm check deadline** option on each workstation on which you want to be aware of the deadline expiration, or, if you want to obtain up-to-date information about the whole environment, define the option on the master domain manager. Deadlines for critical jobs are evaluated automatically, independently of the **bm check deadline** option. For more information about the **bm check deadline** option, see Localopts details.

Comments

- If you do not specify a value for `priority`, the job stream reverts to its original scheduled priority.

- If you do not specify a value for **limit**, the value defaults to 0.
- If you do not specify a workstation in `follows`, `needs`, or `opens`, the default is the workstation on which the job stream runs.
- You cannot use this command to add a `resource` or a `prompt` as dependencies unless they already exists in the production plan. To see which resource and prompts exist in the plan refer to "showresources" on page 378 and "showprompts" on page 375.

Examples

To add a prompt dependency to job stream `sked9(0900 02/19/06)`, run the following command:

```
ads sked9(0900 02/19/06) ; prompt=msg103
```

To add an external follows dependency from to job `JOBB` in job stream `CPUA#SCHED_02(0600 02/24/06)` and a job limit to job stream `CPUA#TEST(0900 02/19/06)`, run the following command:

```
ads CPUA#TEST(0900 02/19/06) ; follows CPUA#SCHED_02(0600 02/24/06).JOBB; limit=2
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Workload**→**Monitor**→**Monitor Job Streams**
2. Select **All Job Streams in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job stream and click the **Dependencies...** tab.

altpass

Alters the password of a user object in the current production plan.

You must have ***altpass*** access to the user object.

Syntax

altpass
[*workstation*#]
username
[;"*password*"]

Arguments

workstation
Specifies the workstation on which the user is defined. Use the upper case for this field even though you used the mixed case when specifying the *workstation* in the user definition. For more information refer to "User definition" on page 170. Do not specify this field if the user belongs to a Windows domain managed by active directory. The default is the workstation on which you are running **conman**.

username
Specifies the name of a user. Use the same user name specified in the Tivoli Workload Scheduler database and nothe that they are case-sensitive. For more information, see "User definition" on page 170.

password
: Specifies the new password. It must be enclosed in double quotes. To indicate no password for the user, use two consecutive double quotes ("").

Comments

If you do not specify a *password*, **conman** prompts for a password and a confirmation. The password is not displayed as it is entered and should not be enclosed in quotes. Note that the change is made only in the current production plan, and is therefore temporary. To make a permanent change see "User definition" on page 170.

Examples

To change the password of user `Jim` on workstation `mis5` to `mynewpw`, run the following command:

```
altpass MIS5#JIM;"mynewpw"
```

To change the password of user `jim` on workstation `Mis5` to `mynewpw` without displaying the password, run the following command:

```
altpass MIS5#JIM
password: xxxxxxxx
confirm: xxxxxxxx
```

To change the password of user `Jim`, defined in an active directory managed Windows domain named `twsDom`, to `mynewpw`, run the following command:

```
altpass TWSDOM\JIM;"mynewpw"
```

altpri

Alters the priority of a job or job stream.

You must have ***altpri*** access to the job or job stream.

Syntax

{**altpri** | **ap**} *jobselect* | *jstreamselect*
  [;*pri*]
  [**;noask**]

Arguments

jobselect
: See "Selecting jobs in commands" on page 296.

jstreamselect
: See "Selecting job streams in commands" on page 305.

pri Specifies the priority level. You can enter a value of **0** through **99**, **hi**, or **go**.

noask Specifies not to prompt for confirmation before taking action on each qualifying job or job stream.

Examples

To change the priority of the `balance` job in job stream `glmonth(0900 02/19/06)`, run the following command:

```
ap glmonth(0900 02/19/06).balance;55
```

To change the priority of job stream `glmonth(0900 02/19/06)`, run the following command:

```
ap glmonth(0900 02/19/06);10
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor**
2. Select **Monitor Jobs** or **Monitor Job Streams**
3. For jobs, select **All Jobs in plan** or another predefined task name; for job streams, select **All Job Streams in plan** or another predefined task name
4. Choose an engine name, or specify connection properties, and click **OK**
5. Select a job or a job stream and click **More Actions→Priority**.

bulk_discovery

Requests a bulk discovery to update the current status of monitored objects. It is used for the integration with IBM Tivoli Monitoring 6.1 (Tivoli Enterprise Portal).

You must have ***display*** access to the file object.

Syntax

{**bulk_discovery** | **bulk**}

Comments

When the integration with IBM Tivoli Monitoring 6.1 is enabled, the **bulk_discovery** command checks the status of all monitored jobs and job streams within the plan and writes the corresponding events in the event log file.

By default, events are written in the **event.log** file.

Messages indicating the start and end of the bulk discovery activity are logged in the `twsmerge.log`file.

cancel job

Cancels a job.

You must have ***cancel*** access to the job.

Syntax

{**cancel job** | **cj**} *jobselect*
  [;pend]
  [;noask]

Arguments

jobselect
: See "Selecting jobs in commands" on page 296.

pend
: Cancels the job only after its dependencies are resolved.

noask
: Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

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

If you do not use the **;pend** option, jobs and job streams that are dependent on the cancelled job are released immediately from the dependency.

If you include the **;pend** option, and the job has not been launched, cancellation is deferred until all of the dependencies, including an **at** time, are resolved. Once all the dependencies are resolved, the job is cancelled and any jobs or job streams that are dependent on the cancelled job are released from the dependency. During the period the cancel is deferred, the notation **[Cancel Pend]** is listed in the Dependencies column of the job in a **showjobs** display.

If you include the **;pend** option and the job has already been launched, the option is ignored, and any jobs or job streams that are dependent on the cancelled job are immediately released from the dependency.

You can use the **rerun** command to rerun jobs that have been cancelled, or that are marked **[Cancel Pend]**. You can also add and delete dependencies on jobs that are marked **[Cancel Pend]**.

To immediately cancel a job that is marked **[Cancel Pend]**, you can either enter a **release** command for the job or enter another **cancel** command without the **;pend** option.

For jobs with expired **until** times, the notation **[Until]** is listed in the Dependencies column in a **showjobs** display, and their dependencies are no longer evaluated. If such a job is also marked **[Cancel Pend]**, it is not cancelled until you release or delete the **until** time, or enter another **cancel** command without the **;pend** option.

To stop evaluating dependencies, set the priority of a job to zero with the **altpri** command. To resume dependency evaluation, set the priority to a value greater than zero.

Note: In the case of internetwork dependencies, cancelling a job in the `EXTERNAL` job stream releases all local jobs and job streams from the dependency. Jobs in the `EXTERNAL` job stream represent jobs and job streams that have been specified as internetwork dependencies. The status of an internetwork dependency is not checked after a **cancel** is performed. For more information see "Managing internetwork dependencies in the plan" on page 553.

Examples

To cancel job `report` in job stream `apwkly(0900 02/19/06)` on workstation `site3`, run the following command:

```
cj site3#apwkly(0900 02/19/06).report
```

To cancel job `setup` in job stream `mis5(1100 02/10/06)`, if it is not in the ABEND state, run the following command:

```
cj mis5(1100 02/10/06).setup~state=abend
```

To cancel job `job3` in job stream `sked3(0900 02/19/03)` only after its dependencies are resolved, run the following command:

```
cj sked3(0900 02/19/06).job3;pend
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Monitor Jobs**
2. Select **All Jobs in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job and click **More Actions→Cancel**.

cancel sched

Cancels a job stream.

You must have ***cancel*** access to the job stream.

Syntax

{**cancel sched** | **cs**} *jstreamselect*
    [**;pend**]
    [**;noask**]

Arguments

jstreamselect
    See “Selecting job streams in commands” on page 305.

pend Cancels the job stream only after its dependencies are resolved.

noask Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Comments

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

If you do not use the **;pend** option, jobs and job streams that are dependent on the cancelled job stream are released immediately from the dependency.

If you use the **;pend** option and the job stream has not been launched, cancellation is deferred until all of its dependencies, including an **at** time, are resolved. Once all dependencies are resolved, the job stream is cancelled and any dependent jobs or job streams are released from the dependency. During the period the **cancel** is deferred, the notation **[Cancel Pend]** is listed in the Dependencies column of a **showschedules** display.

If you include the **;pend** option and the job stream has already been launched, any remaining jobs in the job stream are cancelled, and any dependent jobs and job streams are released from the dependency.

To immediately cancel a job stream marked **[Cancel Pend]**, either enter a **release** command for the job stream or enter another **cancel** command without the **;pend** option.

To stop evaluating dependencies, set the job stream's priority to zero with the **altpri** command. To resume dependency evaluation, set the priority to a value greater than zero.

If the cancelled job stream contains jobs defined with the **every** option, only the last instance of such jobs is listed as canceled in a **showjobs** display.

Examples

To cancel job stream `sked1(1200 02/17/06)` on workstation `site2`, run the following command:

```
cs site2#sked1(1200 02/17)
```

To cancel job stream `mis2(0900 02/19/06)` if it is in the STUCK state, run the following command:

```
cs mis2(0900 02/19)+state=stuck
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Monitor Job Streams**
2. Select **All Job Streams in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job stream and click **More Actions→Cancel**.

checkhealthstatus

Invokes **chkhltst** service to verify the connectivity between the domain manager and workstations. It checks that the Symphony file is not corrupted, the mailbox files can be successfully read by **mailman**, without errors in the mailbox header, and that the mailbox is not full. This command can be useful to diagnose the reason for an unlinked workstation and to get suggestions about how to recover the problem.

Syntax

{**checkhealthstatus** | **chs**} [*workstation*]

Comments

If *workstation* is not specified, the service is launched locally.

Examples

To check the health status of the *site1* workstation, launch the following command:

```
checkhealthstatus site1
```

confirm

Confirms the completion of a job that was scheduled with the **confirmed** keyword.

You must have ***confirm*** access to the job.

Syntax

{**confirm** | **conf**} *jobselect*
;{**succ** | **abend**}
[;**noask**]

Arguments

jobselect
See "Selecting jobs in commands" on page 296.

succ Confirms that the job ended successfully.

abend
Confirms that the job ended unsuccessfully.

noask Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

Changing the state of a job from ABEND to SUCC does not require that the **confirmed** keyword be used to schedule the job. For more information about job confirmation, see "confirmed" on page 191. For more information about `EXTERNAL` jobs, see "Managing internetwork dependencies in the plan" on page 553.

Table 55 shows the effect of the **confirm** command on the various states of jobs:

Table 55. State change after confirm command

Initial Job State	State after confirm ;succ	State after confirm ;abend
READY	no effect	no effect
HOLD	no effect	no effect
EXEC	**SUCCP**	**ABENP**
ABENP	**SUCCP**	no effect
SUCCP	no effect	no effect
PEND	**SUCC**	**ABEND**
DONE	**SUCC**	**ABEND**
SUCC	no effect	no effect
ABEND	**SUCC**	no effect
FAIL	no effect	no effect
SCHED	no effect	no effect
ERROR (for shadow jobs only)	**SUCC**	**ABEND**
any job in the **`EXTERNAL`** job stream	**SUCC**	**ABEND**

Examples

To issue a **succ** confirmation for job `job3` in job stream `misdly(1200 02/17/06)`, run the following command:

```
confirm misdly(1200 02/17/06).job3;succ
```

To issue an **abend** confirmation for job number **234**, run the following command:

```
confirm 234;abend
```

console

Assigns the Tivoli Workload Scheduler console and sets the message level.

You must have ***console*** access to the workstation.

Syntax

{**console** | **cons**}
  [**sess** | **sys**]
  [**;level=***msglevel*]

Arguments

sess
: Sends Tivoli Workload Scheduler console messages and prompts to standard output.

sys
: Stops sending Tivoli Workload Scheduler console messages and prompts to standard output. This occurs automatically when you exit **conman**.

msglevel
: The level of Tivoli Workload Scheduler messages that are sent to the console. Specify one of the following levels:

  -1 This is the value the product automatically assigns if you modify any of the arguments for the console and you do not reassign any value to `msglevel`. With this value the product sends all the messages generated by all agents and for all operations to the console.

  0 No messages. This is the default on fault-tolerant agents.

  1 Exception messages such as operator prompts and job abends.

  2 Level 1, plus job stream successful messages.

  3 Level 2, plus job successful messages. This is the default on the master domain manager.

  4 Level 3, plus job launched messages.

Comments

If you enter a **console** command with no options, the current state of the console is displayed.

By default, Tivoli Workload Scheduler control processes write console messages and prompts to standard list files. In UNIX, you can also have them sent to the **syslog** daemon.

Examples

To begin writing console messages and prompts to standard output and change the message level to **1**, run the following command:

```
console sess;level=1
```

To stop writing console messages and prompts to standard output and change the message level to **4**, run the following command:

```
cons sys;l=4
```

To display the current state of the console, run the following command:

```
cons
Console is #J675, level 2, session
```

675 is the process ID of the user's shell.

continue

Ignores the next command error.

Syntax

{**continue** | **cont**}

Comments

This command is useful when commands are entered non-interactively. It instructs **conman** to continue running commands even if the next command, following **continue**, results in an error.

Examples

To have **conman** continue with the **rerun** command even if the **cancel** command fails, run the following command:

```
conman "cont&cancel=176&rerun job=sked5(1200 02/17/06).job3"
```

deldep job

Deletes dependencies from a job.

You must have ***deldep*** access to the job.

Syntax

{**deldep job** | **ddj**} *jobselect*
    ;*dependency*[**;**...]
    [**;noask**]

Arguments

jobselect
: See "Selecting jobs in commands" on page 296.

dependency
: The type of dependency. Specify at least one of the following. You can use wildcard characters in *workstation*, *jstream*, *job*, *resource*, *filename*, and *promptname*.

at[**=***time* | *lowtime* | *hightime* | *lowtime*,*hightime*]

confirmed

deadline[**=***time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]]

every

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*(*hhmm* [*mm/dd[/yy]*]) [*.job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**,**...]

needs[**=**[*num*] [*workstation***#**]*resource*[**,**...]]

opens[**=**[*workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]]

priority

prompt[**="**[**:** | **!**]*text***"** | *promptname*[**,**...]]

until[**=***time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**;onuntil** *action*]]

noask Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

If you delete `priority`, the job reverts to its original scheduled priority. When you delete an `opens` dependency, you can include only the base file name and **conman** performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are deleted.

Deleted dependencies no longer remain in effect when running the **rerun** command.

To delete all the `follows` dependencies from the jobs contained in a specific job stream, specify the `follows` keyword as:

```
follows=job_stream_name
```

Do not use a wildcard in this case (such as `follows=job_stream_name.@` because the command will be rejected.

Examples

To delete a resource dependency from job `job3` in job stream `sked9(0900 02/19/06)`, run the following command:

```
ddj sked9(0900 02/19/06).job3 ; needs=2 tapes
```

To delete all external follows dependency from job stream `CPUA#TEST(0900 02/19/06)`, run the following command:

```
ddj CPUA#TEST(0900 02/19/06).JOBA ; follows
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Monitor Jobs**
2. Select **All Jobs in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job and click the **Dependencies...** tab.

deldep sched

Deletes dependencies from a job stream.

You must have ***deldep*** access to the job stream.

Syntax

{**deldep sched** | **dds**} *jstreamselect*
  ;*dependency*[**;**...]
  [**;noask**]

Arguments

jstreamselect

See "Selecting jobs in commands" on page 296.

dependency

The type of dependency. Specify at least one of the following. You can use wildcard characters in *workstation, jstreamname, jobname, resource, filename,* and *promptname*.

at[=*time* | *lowtime* | *hightime* | *lowtime,hightime*]

carryforward

deadline[=*time*[**timezone** | **tz** *tzname*][+*n* **days** | *mm/dd[/yy]*]]

follows=[*netagent*::][*workstation*#]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | @] | *jobstream_id.job*;**schedid**}| *job*[,...]

limit

needs[=[*num*] [*workstation*#]*resource*[,...]]

opens[=[*workstation*#]"*filename*"[(*qualifier*)][,...]]

priority

prompt[="[: | !]*text*" | *promptname*[,...]]

until[=*time* [**timezone**|**tz** *tzname*][+*n* **day**[**s**]] [;**onuntil** *action*]]

noask Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Comments

If you delete `priority` , the job reverts to its original scheduled priority. When you delete an `opens` dependency, you can include only the base file name, and **conman** performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are deleted.

Deleted dependencies no longer remain in effect when running the **rerun** command.

Examples

To delete a `resource` dependency from job stream `sked5(0900 02/19/06)`, run the following command:

```
dds sked5(0900 02/19/06);needs=2 tapes
```

To delete all `follows` dependencies from job stream `sked3(1000 04/19/06)`, run the following command:

```
dds sked3(1000 04/19/06);follows
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Monitor Job Streams**
2. Select **All Job Streams in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job stream and click the **Dependencies...** tab.

deployconf

Gets the latest monitoring configuration for the event monitoring engine on the workstation.

Syntax

{**deployconf** | **deploy**} [*domain*!]*workstation*

Arguments

domain Specifies the name of the destination domain for the operation. Wildcard characters are permitted.

This argument is useful when deploying to more than one workstation in a domain. For example, to deploy the latest monitoring configuration to all the agents in the `AURORABU` domain , use the following command:

```
deploy AURORABU!@
```

The domain is not needed if you do not include wildcard characters in *workstation*.

If you do not include *domain*, and you include wildcard characters in *workstation*, the default domain is the one in which **conman** is running.

workstation
Specifies the name of the workstation where the monitoring engine runs. Wildcard characters are not permitted.

Comments

If the existing configuration is already up-to-date, the command has no effect.

Permission to `start` actions on `cpu` objects is required in the security file to be enabled to run this command.

display

Displays a job file or a job stream definition.

If you specify a file by name, you must have read access to the file. For job files and job stream definitions, you must have ***display*** access to the job or job stream.

Syntax

{**display file** | **df**} *filename* [**;offline**]

{**display job** | **dj**} *jobselect* [**;offline**]

{**display sched** | **ds**} *jstreamselect*
[**valid** {**at** *date* | **in** *date date*}
[**;offline**]

Arguments

filename
Specifies the name of the file, usually a job script file. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumeric characters, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted. The file must be

accessible from your login workstation. Use this option is you want to show only the content of the job script file.

jobselect
: The job whose job file is displayed. See "Selecting jobs in commands" on page 296. The job file must be accessible from your login workstation. This keyword applies only to path and filename of the script file of jobs defined with the **scriptname** option.

jstreamselect
: The job stream whose definition is displayed. See "Selecting job streams in commands" on page 305.

valid
: Specifies the day or the interval of days during which the job stream instances to be displayed must be active. This means that the validity interval of those job stream instances must contain the time frame specified in **valid** argument. The format used for *date* depends on the value assigned to the ***date format*** variable specified in the `localopts` file. If not specified the selected instance is the one valid today.

offline
: Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

Examples

To display the file `c:\maestro\jclfiles\arjob3`, run the following command:

```
df c:\apps\maestro\jclfiles\arjob3
```

To display the script file for job `createpostreports` in job stream `final` offline, run the following command:

```
dj FINAL(0559 03/06/06).CREATEPOSTREPORTS
```

This is a sample output of this command:

```
M235062_99#FINAL(0559 03/06/06).CREATEPOSTREPORTS /home/tws83/CreatePostReports
  #!/bin/sh
  ####################################################################
  # Licensed Materials - Property of IBM
  # ?Restricted Materials of IBM?
  # 5698-WSH
  # (C) Copyright IBM Corp. 1998, 2006 All Rights Reserved.
  # US Government Users Restricted Rights - Use, duplication or
  # disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
  ###################################################################
  #@(#) $Id: CreatePostReports.sh,v 1.0

  ##
  ## CreatePostReports message catalog definitions.
  ##

  ##
  ## message set id
  ##
  MAE_CREATEPOSTREPORTS_SET=226
  MAE_COPYRIGHT_SET=234

  ##
  ....
  ...
  ....
  #
  # End
  #
```

To display the job stream definition for job stream mod, run the following command:

```
ds mod
```

This is a sample output of this command:

```
Job Stream Name   Workstation       Valid From  Updated On  Locked By
 ----------------  ----------------  ----------  ----------  ----------------
MOD               M235062_99        06/30/2007  03/04/2006  -

SCHEDULE M235062_99#MOD VALIDFROM 06/30/2007
ON RUNCYCLE SCHED1_PREDSIMPLE VALIDFROM 07/18/2007 "FREQ=DAILY;INTERVAL=1"
( AT 1111 )
CARRYFORWARD
FOLLOWS M235062_99#SCHED_FIRST1.@
FOLLOWS M235062_99#SCHED_FIRST.JOB_FTA
PRIORITY 66
:
M235062_99#JOBMDM
 SCRIPTNAME "/usr/acct/scripts/gl1" STREAMLOGON root
 DESCRIPTION "general ledger job1"
 TASKTYPE UNIX
 RECOVERY STOP
 PRIORITY 30
NEEDS 16 M235062_99#JOBSLOTS
PROMPT PRMT3

B236153_00#JOB_FTA
FOLLOWS M235062_99#SCHED_FIRST1.@
FOLLOWS M235062_99#SCHED_FIRST.JOB_FTA
PRIORITY 66
:
M235062_99#JOBMDM
 SCRIPTNAME "/usr/acct/scripts/gl1" STREAMLOGON root
 DESCRIPTION "general ledger job1"
 TASKTYPE UNIX
 RECOVERY STOP
 PRIORITY 30
NEEDS 16 M235062_99#JOBSLOTS
PROMPT PRMT3

B236153_00#JOB_FTA
 DOCOMMAND "echo pippo" STREAMLOGON root
 DESCRIPTION "general ledger job1"
 TASKTYPE UNIX
 RECOVERY STOP
 FOLLOWS JOBMDM
END
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Design→Create Workload Definitions**
2. Select an engine name and click **OK**
3. In the Quick Open panel of the ensuing pop-up window, click the **Job Definition** or **Job Stream** button
4. Enter the job or job stream name, or use filters to find it

5. Click **View**.

exit

Exits the **conman** command line program.

Syntax

{**exit** | **e**}

Comments

When you are in help mode in UNIX, this command returns **conman** to command-input mode.

Examples

To exit the **conman** command-line program, run the following command:

```
exit
```

or

```
e
```

fence

Changes the job fence on a workstation. Jobs are not launched on the workstation if their priorities are less than or equal to the job fence.

You must have ***fence*** access to the workstation.

Syntax

{**fence** | **f**} *workstation*
    ;*pri*
    [**;noask**]

Arguments

workstation
: Specifies the workstation name. The default is your login workstation.

pri
: Specifies the priority level. You can enter **0** through **99**, **hi**, **go**, or **system**. Entering **system** sets the job fence to zero.

noask
: Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

The job fence prevents low priority jobs from being launched, regardless of the priorities of their job streams. It is possible, therefore, to hold back low priority jobs in high priority job streams, while allowing high priority jobs in low priority job streams to be launched.

When you first start Tivoli Workload Scheduler following installation, the job fence is set to zero. When you change the job fence, it is carried forward during pre-production processing to the next day's production plan.

To display the current setting of the job fence, use the **status** command.

Examples

To change the job fence on workstation `site4`, run the following command:

```
fence site4;20
```

To change the job fence on the workstation on which you are running **conman**, run the following command:

```
f ;40
```

To prevent all jobs from being launched by Tivoli Workload Scheduler on workstation `tx3`, run the following command:

```
f tx3;go
```

To change the job fence to zero on the workstation on which you are running **conman**, run the following command:

```
f ;system
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **More Actions→Fence...**.

help

Displays help information about commands. Not available in Windows.

Syntax

{**help** | **h**} {*command*|*keyword*}

Arguments

command
: Specifies the name of a **conman** or system command. For **conman** commands, enter the full command name; abbreviations and short forms are not supported. For commands consisting of two words, enter the first word, and help for all versions of the command is displayed. For example, entering **help display** displays information about the **display file**, **display job**, and **display sched** commands.

keyword
: You can also enter the following keywords:

  COMMANDS
  : Lists all conman commands.

  SETUPCONMAN
  : Describes how to setup to use conman.

  RUNCONMAN
  : How to run conman.

SPECIALCHAR
Describes the wildcards, delimiters and other special characters you can use.

JOBSELECT
Lists information about selecting jobs for commands.

JOBSTATES
Lists information about job states.

JSSELECT
Lists information about selecting job streams for commands.

JSSTATES
Lists information about job stream states.

MANAGEBACKLEVEL
Managing jobs and job streams from back-level agents.

Examples

To display a list of all **conman** commands, run the following command:

```
help commands
```

To display information about the **fence** command, run the following command:

```
help fence
```

To display information about the **altpri job** and **altpri sched** commands, run the following command:

```
h altpri
```

To display information about the special characters you can use, run the following command:

```
h specialchar
```

kill

Stops a job that is running. In UNIX, this is accomplished with a UNIX **kill** command. You must have ***kill*** access to the job.

Syntax

{**kill** | **k**} *jobselect*
[**;noask**]

Arguments

jobselect
See “Selecting jobs in commands” on page 296.

noask Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

The **kill** operation is not performed by **conman**; it is run by a Tivoli Workload Scheduler production process, so there might be a short delay.

Killed jobs end in the ABEND state. Any jobs or job streams that are dependent on a killed job are not released. Killed jobs can be rerun.

Examples

To kill the job `report` in job stream `apwkly(0600 03/05/06)` on workstation `site3`, run the following command:

```
kill site3#apwkly(0600 03/05/06).report
```

To kill job number `124` running on workstation `geneva`, run the following command:

```
kill geneva#124
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Monitor Jobs**
2. Select **All Jobs in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job and click **More Actions→Kill**.

limit cpu

Changes the limit of jobs that can run simultaneously on a workstation. You must have ***limit*** access to the workstation.

Syntax

{**limit cpu** | **lc** } *workstation*
    ;*limit*
    [**;noask**]

Arguments

workstation
: Specifies the name of the workstation. Wildcard characters are permitted. The default is your login workstation.

limit
: Specifies the how many jobs can run concurrently on the workstation. Supported values are from **0** to **1024** and **system**.

  If you set limit cpu to **0**:
  - For a job stream in the **READY** state, only jobs with **hi** and **go** priority values can be launched on the workstation.
  - For a job stream with a **hi** or **go** priority value, all jobs with a priority value other than 0 can be launched on the workstation.

  If you set limit cpu to **system**, there is no limit to the number of concurrent jobs on the workstation.

noask
: Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

To display the current job limit on your login workstation, use the **status** command.

When you first start Tivoli Workload Scheduler following installation, the workstation job limit is set to zero, and must be increased before any jobs are launched. When you change the limit, it is carried forward during preproduction processing to the next day's production plan.

Tivoli Workload Scheduler attempts to launch as many jobs as possible within the job limit. There is a practical limit to the number of processes that can be started on a workstation. If this limit is reached, the system responds with a message indicating that system resources are not available. When a job cannot be launched for this reason, it enters the **fail** state. Lowering the job limit can prevent this from occurring.

Examples

To change the job limit on the workstation on which you are running **conman**, run the following command:

```
lc ;12
```

To change the job limit on workstation `rx12`, run the following command:

```
lc rx12;6
```

To set to 10 the job limit on all the workstations belonging to the domain and to child domains, run the following command:

```
lc @!@;10
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **More Actions→Limit...**.

limit sched

Changes the **limit** set in the definition of a job stream. For additional information on setting a limit in a job stream definition, refer to “limit” on page 204. You must have ***limit*** access to the job stream.

Syntax

{**limit sched** | **ls** } *jstreamselect*
  ;*limit*
  [**;noask**]

Arguments

jstreamselect
: See “Selecting job streams in commands” on page 305.

limit
: Specifies the job limit. You can enter **0** through **1024**.

noask
: Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Examples

To change the job limit on all job streams that include `sales` in their name, run the following command:

```
ls sales@;4
```

To change the job limit on job stream `CPUA#Job1`, run the following command:

```
ls CPUA#apwkly;6
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Workload**→**Monitor**→**Monitor Job Streams**
2. Select **All Job Streams in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job stream and click **More Actions**→**Limit...**.

link

Opens communication links between workstations. In a Tivoli Workload Scheduler network, fault-tolerant and standard agents are linked to their domain managers, and domain managers are linked to their parent domain managers. Extended agents are not linked; they communicate through a host.

You must have ***link*** access to the target workstation.

Syntax

{**link** | **lk**} [*domain*!]*workstation*
    [**;noask**]

Arguments

domain Specifies the name of the domain in which links are opened. Wildcard characters are permitted.

This argument is useful when linking more than one workstation in a domain. For example, to link all the agents in domain `stlouis`, use the following command:

```
lk stlouis!@
```

The domain is not needed if you do not include wildcard characters in *workstation*.

If you do not include *domain*, and you include wildcard characters in *workstation*, the default domain is the one in which **conman** is running.

workstation
Specifies the name of the workstation to be linked. Wildcard characters are permitted.

This command is not supported on remote engine workstations.

noask Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

If the **autolink** option is set to **on** in a workstation definition, its link is opened automatically each time Tivoli Workload Scheduler is started. If **autolink** is set to **off**, you must use **link** and **unlink** commands to control linking. For information about **autolink** see "Workstation definition" on page 127.

Assuming that a user has **link** access to the workstations being linked, the following rules apply:

- A user running **conman** on the master domain manager can link any workstation in the network.
- A user running **conman** on a domain manager other than the master can link any workstation in its own domain and subordinate domains. The user cannot link workstations in peer domains.
- A user running **conman** on an agent can link any workstation in its local domain provided that the workstation is a domain manager or host. A peer agent in the local domain cannot be linked.
- To link a subordinate domain while running **conman** in a higher domain, it is not necessary that the intervening links be open.

Examples

Figure 22 and Table 56 on page 338 show the links opened by **link** commands run by users in various locations in the network.
DM*n* are domain managers and **A***nn* are agents.

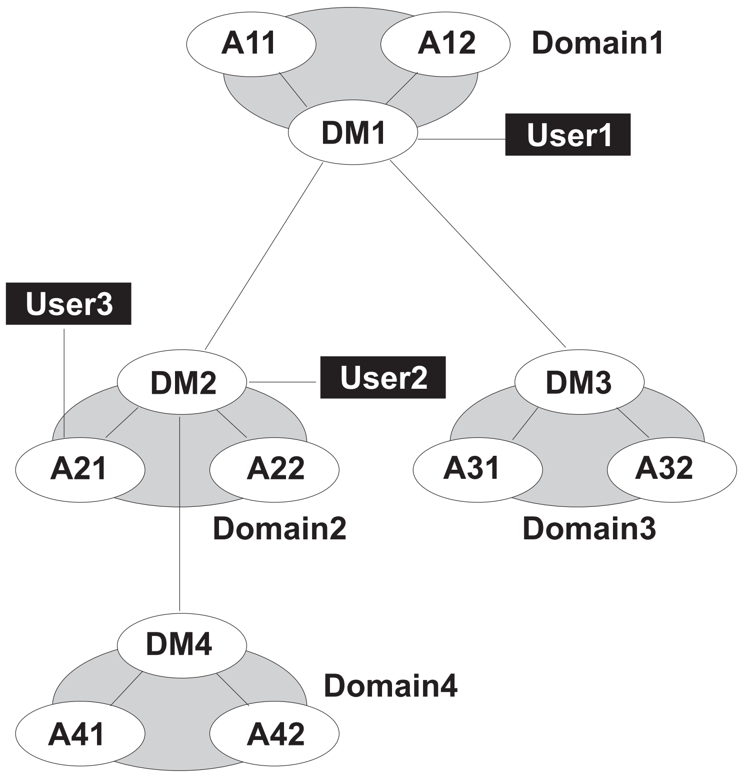


Figure 22. Network links

Table 56. Opened links

Command	**Links Opened by User1**	**Links Opened by User2**	**Links Opened by User3**
link @!@	All links are opened.	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4 DM4-A41 DM4-A42	DM2-A21
link @	DM1-A11 DM1-A12 DM1-DM2 DM1-DM3	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4	DM2-A21
link DOMAIN3!@	DM3-A31 DM3-A32	Not allowed.	Not allowed.
link DOMAIN4!@	DM4-A41 DM4-A42	DM4-A41 DM4-A42	Not allowed.
link DM2	DM1-DM2	Not applicable.	DM2-A21
link A42	DM4-A42	DM4-A42	Not allowed.
link A31	DM3-A31	Not allowed.	Not allowed.

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **Link**.

listsym

Lists the production plan (Symphony files) already processed.

Syntax

{**listsym** | **lis**} [**trial** | **forecast**]
    [**;offline**]

Arguments

trial Lists trial plans.

forecast
Lists forecast plans.

offline
Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

Results

Schedule Date
The date used to select the job streams to run.

Actual Date
The date **batchman** began running the Symphony file.

Start Time
The time **batchman** began running the Symphony file.

Log Date
The date the plan (Symphony file) was logged by the **stageman** command.

Run Num
The run number assigned to the plan (Symphony file). This is used internally for Tivoli Workload Scheduler network synchronization.

Size The number of records contained in the Symphony file.

Log Num
The log number indicating the chronological order of log files. This number can be used in a **setsym** command to switch to a specific log file.

Filename
The name of the log file assigned by the **stageman** command.

Examples

To list the production plan files, run the following command:

```
listsym
```

this is a sample output for the command:

```
Job Stream   Actual Start      Log      Run          Log
Date         Date   Time       Date     Num    Size   Num   Filename
03/05/06   03/05/06 21:06   03/05/06     42    534     1  M200603052111    Exp
03/04/06   03/04/06 15:59   03/05/06     41    463     2  M200603052106    Exp
03/04/06   03/04/06 15:51   03/04/06     40    362     3  M200603041559    Exp
03/04/06   03/04/06 14:31   03/04/06     39    460     4  M200603041551    Exp
03/04/06   03/04/06 14:26   03/04/06     38    436     5  M200603041431    Exp
03/04/06   03/04/06 14:24   03/04/06     37    436     6  M200603041426    Exp
03/04/06   03/04/06 14:19   03/04/06     36    436     7  M200603041424    Exp
03/04/06   03/04/06 14:17   03/04/06     35    436     8  M200603041419    Exp
03/04/06   03/04/06 14:17   03/04/06     34    364     9  M200603041417    Exp
```

To list files containing trial plans, run the following command:

```
listsym trial
```

this is a sample output for the command:

```
Job Stream   Actual Start      Log      Run          Log
Date         Date   Time       Date     Num    Size   Num   Filename
03/03/06                    03/03/06      0    126     1  Tpippo           Exp
03/03/06                    03/03/06      0   1850     2  Tangelo2         Exp
03/03/06                    03/03/06      0   1838     3  Tangelo1         Exp
```

To list the files containing the forecast plans, run the following command:

```
listsym forecast
```

This is a sample output for the command:

```
Job Stream   Actual Start      Log      Run          Log
Date         Date   Time       Date     Num    Size   Num   Filename
03/03/06                    03/03/06      0     62     1  Fpluto           Exp
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Forecast→List Available Plans**
2. Select an engine name and click a plan type or write a plan filename
3. Click **Display Plans List**.

recall

Displays prompts that are waiting for a response.

You must have ***display*** access to the prompts.

Syntax

{**recall** | **rc**} [*workstation*]
  [**;offline**]

Arguments

workstation
: Specifies the name of the workstation on which the prompt was issued. If you do not specify a workstation, only prompts for the login workstation and global prompts are displayed.

offline
: Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

Results

State The state of the prompt. The state of pending prompts is always ASKED.

Message or Prompt
: For named prompts, the message number, the name of the prompt, and the message text. For unnamed prompts, the message number, the name of the job or job stream, and the message text.

Examples

To display pending prompts issued on the workstation on which you are running **conman**, run the following command:

```
recall
```

or:

```
rc
```

To display pending prompts on workstation `site3`, run the following command:

```
rc site3
```

To display pending prompts on all workstations and have the output sent to **conman**'s offline device, run the following command:

```
rc @;offline
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Workload Dependencies**
2. Select **Monitor Prompts**
3. You can either select **All Prompts in plan**, which will list all prompts regardless of their status, or create and select another predefined task that lists only prompts in ASKED status
4. Choose an engine name, or specify connection properties, and click **OK**.

redo

Edits and reruns the previous command.

Syntax

{**redo** | **red**}

Context

When you run the **redo** command, **conman** displays the previous command, so that it can be edited and rerun. Use the spacebar to move the cursor under the character to be modified, and enter the following directives.

Directives

d[*dir***]**
Deletes the character above the **d**. This can be followed by other directives.

i*text*
Inserts text before the character above the **i**.

r*text*
Replaces one or more characters with text, beginning with the character above the **r**. Replace is implied if no other directive is entered.

>*text*
Appends text to the end of the line.

>d[*dir* | *text***]**
Deletes characters at the end of the line. This can be followed by another directive or text.

>r*text*
Replaces characters at the end of the line with text.

Directive Examples

ddd
Deletes the three characters above the **d**s.

iabc
Inserts **abc** before the character above the **i**.

rabc
Replaces the three characters, starting with the one above the **r**, with **abc**.

abc
Replaces the three characters above **abc** with **abc**.

d diabc
Deletes the character above the first **d**, skips one character, deletes the character above the second **d**, and inserts **abc** in its place.

>abc
Appends **abc** to the end of the line.

>ddabc
Deletes the last two characters in the line, and inserts **abc** in their place.

>rabc Replaces the last three characters in the line with **abc**.

Examples

To insert a character, run the following command:

```
redo
setsm 4
    iy
setsym 4
```

To replace a character, run the following command:

```
redo
setsym 4
      5
setsym 5
```

release job

Releases jobs from dependencies.

You must have ***release*** access to the job.

Syntax

{**release job** | **rj**} *jobselect*
[**;***dependency*[**;**...]]
[**;noask**]

Arguments

jobselect
Specifies the job or jobs to be released. See "Selecting jobs in commands" on page 296.

dependency
The type of dependency. You can specify one of the following. You can use wildcard characters in *workstation, jstreamname, jobname, resource, filename,* and *promptname*.

at[**=***time* | *lowtime* | *hightime* | *lowtime*,*hightime*]

confirmed

deadline[**=***time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]]

every

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**,**...]

needs[**=**[*num*] [*workstation***#**]*resource*[**,**...]]

opens[**=**[*workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]]

priority

prompt[**="**[**:** | **!**]*text***"** | *promptname*[**,**...]]

until[**=***time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**;onuntil** *action*]]

noask Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

When you release an `opens` dependency, you can include only the base file name, and **conman** performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are released.

For `needs` dependencies, the released job is given the required number of units of the resource, even though they might not be available. This can cause the available units in a **showresources** to display a negative number.

When you release a job from a `priority` dependency, the job reverts to its original scheduled priority.

Released dependencies remain in effect when running the **rerun** command.

Examples

To release job `job3` in job stream `ap(1000 03/05/06)` from all of its dependencies, run the following command:

```
rj ap(1000 03/05/06).job3
```

To release all jobs on workstation `site4` from their dependencies on a prompt named `glprmt`, run the following command:

```
rj site4#@.@;prompt=glprmt
```

See also

In the Tivoli Dynamic Workload Console:
1. Click **Tivoli Workload Scheduler→Workload→Monitor→Monitor Jobs**
2. Select **All Jobs in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job and click **More Actions→Release**.

release sched

Releases job streams from dependencies.

You must have ***release*** access to the job stream.

Syntax

{**release sched** | **rs**} *jstreamselect*
 [**;***dependency*[**;**...]]
 [**;noask**]

Arguments

jstreamselect
: See "Selecting job streams in commands" on page 305.

dependency
: The type of dependency. Specify one of the following. You can use wildcard characters in *workstation, jstream, job, resource, filename,* and *promptname*.

 at[=*time* | *lowtime* | *hightime* | *lowtime,hightime*]

carryforward

deadline[**=***time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]]

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**,**...]

limit

needs[**=**[*num*] [*workstation***#**]*resource*[**,**...]]

opens[**=**[*workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]]

priority

prompt[**="**[**:** | **!**]*text***"** | *promptname*[**,**...]]

until[**=***time* [**timezone**|**tz** *tzname*][**+***n* **day**[**s**]] [**;onuntil** *action*]]

noask Specifies not to prompt for confirmation before taking action on each qualifying job stream.

Comments

When deleting an `opens` dependency, you can include only the base file name, and **conman** performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are released.

For `needs` dependencies, the released job stream is given the required number of units of the resource, even though they might not be available. This can cause the available units in a **showresources** to display a negative number.

When you release a job stream from a `priority` dependency, the job stream reverts to its original priority.

In certain circumstances, when you have submitted a **deldep** command, the command might have succeeded even though it is again forwarded to **batchman**. For more information, see "Conman commands processing" on page 296.

Examples

To release job stream instance with *jobstream_id* `0AAAAAAAAAAAAABSE` from all of its dependencies, run the following command:

```
rs 0AAAAAAAAAAAAABSE; schedid
```

To release job stream `sked5(1105 03/07/06)` from all of its `opens` dependencies, run the following command:

```
rs sked5(1105 03/07/06);opens
```

To release all job streams on workstation `site3` from their dependencies on job stream `main#sked23`, run the following command:

```
rs site3#@;follows=main#sked23
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Workload**→**Monitor**→**Monitor Job Streams**
2. Select **All Job Streams in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**

4. Select a job stream and click **More Actions→Release**.

reply

Replies to a job or job stream prompt.

You must have ***reply*** access to the named or global prompt. To reply to an unnamed prompt, you must have ***reply*** access to prompts, and ***reply*** access to the associated job or job stream.

Syntax

{**reply** | **rep**}
  { *promptname* | [*workstation*#]*msgnum*}
  ;*reply*
  [;**noask**]

Arguments

promptname
: Specifies the name of a global prompt. Wildcard characters are permitted.

workstation
: Specifies the name of the workstation on which an unnamed prompt was issued.

msgnum
: Specifies the message number of an unnamed prompt. You can display message numbers with the **recall** and **showprompts** commands.

reply
: Specifies the reply, either **Y** for yes or **N** for no.

noask
: Specifies not to prompt for confirmation before taking action on each qualifying prompt.

Comments

If the reply is **Y**, dependencies on the prompt are satisfied. If the reply is **N**, the dependencies are not satisfied and the prompt is not reissued.

Prompts can be replied to before they are issued. You can use the **showprompts** command to display all prompts, whether or not they have been issued.

Examples

To reply **Y** to the global prompt `arpmt`, run the following command:

```
reply arprmt;y
```

To reply **N** to message number **24** on workstation `site4`, run the following command:

```
rep site4#24;n
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Workload Dependencies→Monitor Prompts**
2. Select **All Prompts in plan** or another predefined task name

3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a prompt and click **Reply Yes** or **Reply No**.

rerun

Reruns a job.

You must have ***rerun*** access to the job.

Syntax

{**rerun** | **rr**} *jobselect*
  [;from=[*wkstat***#**]*job*[
  ;at=*time*]
  [**;pri=***pri*]]
  [;noask]

{**rerun**|**rr**} *jobselect*
  [;step=*step*]
  [;noask]

Arguments

jobselect
: Specifies the name of one or more jobs. Wildcard characters are permitted.

from=[*wkstat***#]***job*
: Specifies the name of a job defined in the database whose job file or command will be run in place of the job specified by *jobselect*.

  *wkstat***#**
  : Specifies the name of the workstation on which the **from** job runs. The default is the workstation on which **conman** is running.

  job
  : Specifies the name of the **from** job definition The following types of job names are not permitted:
    - The names of jobs submitted using the **submit file** and **submit docommand** commands.
    - The alias names of jobs submitted using the **submit job** command.

  To use the **from** argument, you must have access to the database from the computer on which you are running **conman**

at=*time*
: Specifies the rerun job's start time, expressed as follows:

  *hhmm [***timezone|tz** *tzname]* [**+***n* **days** | *date*]

  where:

  hhmm
  : The hour and minute.

  +*n* **days**
  : The next occurrence of *hhmm* in *n* number of days.

  date
  : The next occurrence of *hhmm* on *date*, expressed as *mm/dd[/yy]*.

  timezone|tz *tzname*
  : The name of the time zone of the job. See Chapter 16, "Managing time zones," on page 531 for valid names.

pri=*pri*
: Specifies the priority to be assigned to the rerun job. If you do not specify a priority, the job is given the same priority as the original job.

step=*step*
: Specifies that the job is rerun using this name in place of the original job name. See “Usage Notes®” for more information.

noask Specifies not to prompt for confirmation before taking action on each qualifying job.

Comments

You can rerun jobs that are in the SUCC, FAIL, or ABEND state. A rerun job is placed in the same job stream as the original job, and inherits the original job's dependencies. If you rerun a repetitive (**every**) job, the rerun job is scheduled to run at the same rate as the original job.

Note: You can issue **rerun** for jobs in the `EXTERNAL` job stream that are in the ERROR state. Jobs in the `EXTERNAL` job stream represent jobs and job streams that have been specified as internetwork dependencies. The job state is initially set to **extrn** immediately after a **rerun** is run, and **conman** begins checking the state.

When **;from** is used, the name of the rerun job depends on the value of the Global Option **retain rerun job names**. If the option is set to **Y**, rerun jobs retain the original job names. If the option is set to **N**, rerun jobs are given the **from** job names. For more information, refer to the Tivoli Workload Scheduler *Administration Guide*.

In **conman** displays, rerun jobs are displayed with the notation **>>rerun as**. To refer to a rerun job in another command, such as **altpri**, you must use the original job name.

When a job is rerun with the **;step** option, the job runs with *step* in place of its original name. Within a job script, you can use the **jobinfo** command to return the job name and to run the script differently for each iteration. For example, in the following UNIX script, the **jobinfo** command is used to set a variable named **STEP** to the name that was used to run the job. The **STEP** variable is then used to determine how the script is run.

```
...
MPATH=`maestro`
STEP=`$MPATH/bin/jobinfo job_name`
if [$STEP = JOB3]
  then
  ...
  STEP=JSTEP1
  fi
if [$STEP = JSTEP1]
  then
  ...
  STEP=JSTEP2
  fi
if [$STEP = JSTEP2]
  then
  ...
  fi
...
```

In **conman** displays, jobs rerun with the **;step** option are displayed with the notation **>>rerun step**.

For information about **jobinfo**, see "jobinfo" on page 449.

Examples

To rerun job `job4` in job stream `sked1` on workstation `main`, run the following command:

```
rr main#sked1.job4
```

To rerun job `job5` in job stream `sked2` using the job definition for job `jobx` where the job's **at** time is set to 6:30 p.m. and its priority is set to **25**, run the following command:

```
rr sked2.job5;from=jobx;at=1830;pri=25
```

To rerun job `job3` in job stream `sked4` using the job name `jstep2`, run the following command:

```
rr sked4.job3;step=jstep2
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Workload**→**Monitor**→**Monitor Jobs**
2. Select **All Jobs in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a job and click **Rerun...**.

resetFTA

Generates an updated `Sinfonia` file and sends it to a fault-tolerant agent on which the `Symphony` file has corrupted.

Note: Complete removal and replacement of the `Symphony` file causes some loss of data, for example events on job status, or the contents of the Mailbox.msg message and the tomaster.msg message queues. If state information about a job was contained in those queues, that job is rerun. It is recommended that you apply this command with caution.

In the process, the following files are moved to the *TWA_home*/TWS/tmp directory:

- Appserverbox.msg
- clbox.msg
- Courier.msg
- Intercom.msg
- Mailbox.msg
- Monbox.msg
- Moncmd.msg
- Symphony
- Sinfonia

Before the command is performed, an information message is displayed to request confirmation and ensure the command is not issued by mistake. If one of the target files cannot be moved because it is being used by another process (for example, the mailman process is still running) the operation is not performed and an error message is displayed.

Authorization

You must have **RESETFTA** access to the fault-tolerant agent you want to reset.

Syntax

resetFTA cpu

Arguments

cpu Is the fault-tolerant agent to be reset.

This command is not available in the Dynamic Workload Console.

Examples

To reset the fault-tolerant agent with name omaha, run the following command:

```
resetFTA omaha
```

See also

For more information about the fault-tolerant agent recovery procedure, see the section about the recovery procedure on a fault-tolerant agent in *Tivoli Workload Scheduler: Troubleshooting Guide*..

resource

Changes the number of total units of a resource.

You must have ***resource*** access to the resource.

Syntax

{**resource** | **reso**} [*workstation*#]
resource;*num*
[;**noask**]

Arguments

workstation
Specifies the name of the workstation on which the resource is defined. The default is the workstation on which **conman** is running.

resource
Specifies the name of the resource.

num Specifies the total number of resource units. Valid values are **0** through **1024**.

noask Specifies not to prompt for confirmation before taking action on each qualifying resource.

Examples

To change the number of units of resource `tapes` to **5**, run the following command:

```
resource tapes;5
```

To change the number of units of resource `jobslots` on workstation `site2` to **23**, run the following command:

```
reso site2#jobslots;23
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Workload**→**Monitor**→**Workload Dependencies**→**Monitor Resources**
2. Select **All Resources in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a resource and click **Change Units...**.

setsym

Selects a production plan archive file. Subsequent display commands show the contents of the archived production plan. You cannot modify the information in a production plan archive file.

Syntax

{**setsym** | **set**} [**trial** | **forecast**] [*filenum*]

Arguments

trial Lists trial plans.

forecast
Lists forecast plans.

filenum
Specifies the number of the production plan archive file. If you do not specify a log file number, the pointer returns to zero, the current production plan (`Symphony`). Use the **listsym** command to list archive file numbers.

Examples

To select production plan archive file **5**, run the following command:

```
setsym 5
```

To select the current production plan (`Symphony` file), run the following command:

```
set
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Workload**→**Forecast**→**List Available Plans**
2. Select an engine name and either click **Archived plans** or provide a plan filename
3. Click **Display Plans List**.

showcpus

Displays information about workstations and links.

The displayed information is updated only while Tivoli Workload Scheduler (batchman) is running on the workstations. If **`batchman`** is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showcpus** | **sc**} [[*domain*!]*workstation*]
 [**;info**|**;link**]
 [**;offline**]

{**showcpus** | **sc**} [[*domain*!]*workstation*] [**;getmon**]

Arguments

domain Specifies the name of a domain. The default is the domain in which the command is run.

workstation
Specifies the name of a workstation. The default is the workstation where the command is run. When no domain and no workstation are specified, the output can be the following:

- The following command displays all the workstations that are in the domain of the workstation where the command was run, plus all the connected domain managers if the workstation is a domain manager.
```
conman "sc"
```
- The following command displays all the workstations that are in the domain of the workstation where the command was run, without the connected domain managers.
```
conman "sc @"
```

info Displays information in the **info** format.

link Displays information in the **link** format.

offline
Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

getmon
Returns the list of event rules defined for the monitor running on the specified workstation in the following format:
```
<rule_name>::<eventProvider>#<eventType>:<scope>
```

The rule scope is automatically generated information about rule attributes, such as the workstations where it is used, a job or file name, and so on.

The header of the output contains also the time stamp of when the rule configuration package was last generated.

Note: This option is not valid on dynamic workload broker workstations (or dynamic domain managers). In this case, you can retrieve the information about the active rules defined in these workstations in the `TWA_home\TWS\monconf\TWSObjectsMonitor.cfg` file on the master domain manager.

Results

When the `getmon` parameter is not used, the output of the command is produced in three formats, **standard**, **info**, and **link**. The default value is **standard**. The meaning of the characters displayed depends on the type of format you select.

When run on a workstation with a version of Tivoli Workload Scheduler earlier than version 8.6, the **sc** command shows as FTA the workstation types introduced with Tivoli Workload Scheduler version 8.6, pool, dynamic pool, agent and remote engine.

When the `getmon` parameter is used, the list of rules is provided as separate output.

Examples

1. To display information about the workstation on which you are running **conman** in the **info** format, run the following command:

   ```
   showcpus ;info
   ```

   A sample output for this command is:

   ```
   CPUID         VERSION   TIME ZONE               INFO
   MASTER         8.6.0.0  US/Pacific              Linux 2.6.5-7.191-s390 #1 SM
   FTA1           8.6.0.0                          Linux 2.4.9-e.24 #1 Tue May
   FTA2           8.6.0.0                          HP-UX B.11.11 U 9000/785
   ```
2. To display link information for all workstations, run the following command:

   ```
   sc @!@;link
   ```

   A sample output is the following:

   ```
   CPUID       HOST      FLAGS  ADDR  NODE
   MASTER     MASTER     AF T  51099  9.132.239.65
   FTA1       FTA1       AF T  51000  CPU235019
   FTA2       FTA2       AF T  51000  9.132.235.42
   BROKER1    MASTER     A  T  51111  9.132.237.17
   ```
3. To display information about the workstation, run the following command:

   ```
   showcpus
   ```

   If you run this command in an environment when the primary connection of the workstation with its domain or higher manager is not active, you receive the following output:

   ```
   CPUID   RUN    NODE     LIMIT FENCE  DATE     TIME    STATE    METHOD DOMAIN
   MASTER  360 *WNT  MASTER   10    0 03/05/2010 1348   I J   E          MASTERDM
   FTA1    360  WNT  FTA      10    0 03/05/2010 1348 FTI JW M           MASTERDM
   FTA2    360  WNT  FTA      10    0 03/05/2010 1348 FTI JW M           MASTERDM
   FTA3    360  WNT  MANAGER  10    0 03/05/2010 1348 LTI JW M           DOMAIN1
   FTA4    360  WNT  FTA      10    0 03/05/2010 1348 F I J  M           DOMAIN1
   FTA5    360  WNT  FTA      10    0 03/05/2010 1348   I J  M           DOMAIN1
   SA1     360  WNT  S-AGENT  10    0 03/05/2010 1348 F I J  M           DOMAIN1
   XA_FTA4 360  OTHR X-AGENT  10    0 03/05/2010 1348 L I J  M           DOMAIN1
   FTA6    360  WNT  MANAGER  10    0 03/05/2010 1348 F I J  M           DOMAIN2
   FTA7    360  WNT  FTA      10    0 03/05/2010 1349 F I J  M           DOMAIN2
   FTA7    360  WNT  FTA      10    0 03/05/2010 1349 F I J  M           DOMAIN2
   BROKER  360  OTHR BROKER   10    0 03/05/2010 1349 LTI JW             MASTERDM
   ```

If you run this command in an environment when the primary connection of the workstation with its domain or higher manager is active and at least one secondary connection is not active, you receive the following output:

```
CPUID   RUN     NODE     LIMIT FENCE   DATE    TIME    STATE     METHOD DOMAIN
MASTER  360 *WNT  MASTER   10    0  03/05/2010 1348    I J   E          MASTERDM
FTA1    360  WNT  FTA      10    0  03/05/2010 1348  FTI JW M           MASTERDM
FTA2    360  WNT  FTA      10    0  03/05/2010 1348  FTI JW M           MASTERDM
FTA3    360  WNT  MANAGER  10    0  03/05/2010 1348  FTI JW M           DOMAIN1
FTA4    360  WNT  FTA      10    0  03/05/2010 1348  F I J  M           DOMAIN1
FTA5    360  WNT  FTA      10    0  03/05/2010 1348  L I    M           DOMAIN1
SA1     360  WNT  S-AGENT  10    0  03/05/2010 1348  F I J  M           DOMAIN1
XA_FTA4 360  OTHR X-AGENT  10    0  03/05/2010 1348  L I J  M           DOMAIN1
FTA6    360  WNT  MANAGER  10    0  03/05/2010 1348  F I J  M           DOMAIN2
FTA7    360  WNT  FTA      10    0  03/05/2010 1349  F I J  M           DOMAIN2
```

If you run this command in an environment when the primary connection of the workstation with its domain or higher manager and all secondary connections are active, you receive the following output:

```
CPUID   RUN     NODE     LIMIT FENCE   DATE    TIME     STATE    METHOD DOMAIN
MASTER  360 *WNT  MASTER   10    0  03/05/2010 1348    I J    E         MASTERDM
FTA1    360  WNT  FTA      10    0  03/05/2010 1348  FTI JW M           MASTERDM
FTA2    360  WNT  FTA      10    0  03/05/2010 1348  FTI JW M           MASTERDM
FTA3    360  WNT  MANAGER  10    0  03/05/2010 1348  FTI JW M           DOMAIN1
FTA4    360  WNT  FTA      10    0  03/05/2010 1348  F I J  M           DOMAIN1
FTA5    360  WNT  FTA      10    0  03/05/2010 1348  F I    M           DOMAIN1
SA1     360  WNT  S-AGENT  10    0  03/05/2010 1348  F I J  M           DOMAIN1
XA_FTA4 360  OTHR X-AGENT  10    0  03/05/2010 1348  L I J  M           DOMAIN1
FTA6    360  WNT  MANAGER  10    0  03/05/2010 1348  F I J  M           DOMAIN2
FTA7    360  WNT  FTA      10    0  03/05/2010 1349  F I J  M           DOMAIN2
```

4. To get a list of active rule monitors on the workstation named CPU1, run this command:

   ```
   sc CPU1 getmon
   ```

   You get the following output:

   ```
   Monitoring configuration for CPU1:
   *********************************************
   *** Package Date : 04/22/2009 12:00 GMT ***
   *********************************************
   Rule1::FileMonitor#FileCreated:Workstation=CPU1,CPU2;File="\tmp\filename"
   Rule2::FileMonitor#ModificationCompleted:Workstation=CPU1,CPU3;File="\staging\orders"
   Rule3::TWSObjectsMonitor#JobSubmit:JobKey=CPU1#JS1.Job1
   Rule5::TWSObjectsMonitor#JobLate:JobKey=CPU1#JS1.Job1
   ```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**

Standard format

CPUID
: The name of the workstation to which this information applies.

RUN The run number of the Symphony file .

NODE
: The node type and workstation type. Node types are as follows:
  - UNIX
  - WNT

- OTHER
- ZOS
- IBM i

Workstation types are as follows:
- MASTER
- MANAGER
- FTA
- S-AGENT
- X-AGENT
- AGENT
- POOL
- D-POOL
- REM-ENG

LIMIT
The Tivoli Workload Scheduler job limit.

FENCE
The Tivoli Workload Scheduler job fence.

DATE TIME
The date and time Tivoli Workload Scheduler started running the current production plan (`Symphony` file).

STATE
Displays the following information:

- The state of the workstation's links and processes. Up to five characters are displayed as follows. The explanation of the characters is divided based on the character scope:

  ```
  [L|F] [T|H|X|B] [I] [J] [W|H|X] [M] [E|e] [D] [A|R]
  ```

  where:

  L The primary link is open (linked) to its domain or upper manager.

  If the workstation is of type dynamic agent or remote engine, this flag indicates that the workstation is connected to the workload broker server.

  If the workstation is of type pool or dynamic pool, this flag indicates that the workload broker workstation the pool or dynamic pool is registered to is linked to its domain or upper manager.

  F The workstation is fully linked through primary and all secondary connections. This flag appears only if the ***enSwfaultTol*** global option is set to *YES* using the **optman** command line on the master domain manager and it indicates that the workstation is directly linked to its domain manager and to all its full backup domain managers. For information on how to use the **optman** command line, refer to *IBM Tivoli Workload Scheduler Administration Guide*.

  T This flag is displayed if the fault-tolerant agent is directly linked to the domain manager from where you run the command.

  H The workstation is linked through its host.

  X The workstation is linked as an extended agent (x-agent).

  B The workstation communicates through the workload broker server.

  I If the workstation is of type agent MASTER, MANAGER, FTA, S-AGENT, X-AGENT, this flag indicates that **jobman** program has completed startup initialization.

If the workstation is of type dynamic agent, pool or dynamic pool, this flag indicates that the agent is correctly initialized.

If the workstation is of type remote engine, this flag indicates that the communication between the remote engine workstation and the remote engine is correctly initialized.

J If the workstation is of type agent MASTER, MANAGER, FTA, S-AGENT, X-AGENT, this flag indicates that **jobman** program is running.

If the workstation is of type dynamic agent, this flag indicates that JobManager is running. Because no monitoring is performed on dynamic pool workstations, for this workstation type the J character is always shown.

If the workstation is of type pool, this flag indicates that the JobManager process is running on at least one agent registered to the pool.

If the workstation is of type remote engine, this flag indicates that the ping command to the remote engine is successful.

| **W** The workstation is linked via TCP/IP using the **writer** process.

| If the workstation running **conman** is directly linked to the
| remote workstation, you see the flag W because the local
| mailman is linked to the remote writer process.

```
LTI JW
```

| If the workstation running **conman** is not directly linked to the
| remote workstation, you do not see the flag W because the local
| mailman is not directly linked to the remote writer process.

```
L I J
```

| For more details about the **writer** process, see Network
| processes.

Note: If the workstation running **conman** is the extended agent's host, the state of the extended agent is

```
LXI JX
```

If the workstation running **conman** is not the extended agent's host, the state of the extended agent is

```
LHI JH
```

- The state of the monitoring agent. Up to three characters are displayed as follows:

```
[M] [E|e] [D]
```

where:

M The `monman` process is running. This flag is displayed for all the workstations in the network when the event-driven workload automation feature is enabled (global option `enEventDrivenWorkloadAutomation` is set to `yes`), with the exception of those workstations where `monman` was manually stopped (using either **conman** or the Tivoli Dynamic Workload Console).

E The event processing server is installed and running on the workstation.

e The event processing server is installed on the workstation but is not running.

D The workstation is using an up-to-date package monitoring configuration. This flag is displayed for the workstations on which the latest package of event rules was deployed (either manually with the `planman deploy` command or automatically with the frequency specified by the `deploymentFrequency` global option).

- The state of the WebSphere Application Server. A one-character flag is displayed, if the application server is installed:

  ```
  [A|R]
  ```

  where:

  A The WebSphere Application Server was started.

  R The WebSphere Application Server is restarting.

  The flag is blank if the application server is down or if it was not installed.

METHOD
The name of the access method specified in the workstation definition. For extended agents only.

DOMAIN
The name of the domain in which the workstation is a member.

Info format

CPUID
The name of the workstation to which this information applies.

VERSION
The version of the Tivoli Workload Scheduler agent installed on the workstation.

TIMEZONE
The time zone of the workstation. It is the same as the value of the TZ environment variable. For an extended agent, this is the time zone of its host. For a remote engine workstation, this is the time zone of the remote engine.

INFO An informational field. For all the workstation types except the extended agent and the broker workstations it contains the operating system version and the hardware model. For extended agents and remote engine workstations, no information is listed. For remote engine workstation it shows Remote Engine.

Link format

CPUID
The name of the workstation to which this information applies.

HOST The name of the workstation acting as the host to a standard agent or extended agent. For domain managers and fault-tolerant agents, this is the same as CPUID. For standard agent and broker workstations, this is the name of the domain manager. For extended agents, this is the name of the host workstation.

FLAGS
The state of the workstation properties. Up to five characters are displayed as follows:

`[A] [B] [F] [`*s*`] [T]`

A	Autolink is turned on in the workstation definition.
B	This flag is used only in end-to-end environment and it indicates if the **deactivate job launching** flag is disabled.
F	Full Status mode is turned on in the workstation definition.
s	The ID of **mailman** server for the workstation.
T	The link is defined as TCP/IP.

ADDR
The TCP/IP port number for the workstation.

NODE
The node name of the workstation.

showdomain

Displays domain information.

The displayed information is updated only as long as Tivoli Workload Scheduler (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showdomain** | **showd**} [*domain*]
  [**;info**]
  [**;offline**]

Arguments

domain Specifies the name of the domain. The default is the domain in which **conman** is running. Wildcard characters are permitted.

info Displays information in the **info** format.

offline
Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

Results

The output of the command is produced in two formats, **standard**, and **info**.

Examples

To display information about the domain `masterdm`, run the following command:

```
showdomain masterdm
```

A sample output is the following:

```
DOMAIN          MANAGER         PARENT
*MASTERDM      *MASTER
```

To display the member workstations in all domains in the **info** format, run the following command:

```
showdomain @;info
```

a sample output is the following:

```
DOMAIN           MEMBER-CPUs      CPU-Type
 MASTERDM        *MASTER           MASTER
 DOM1             FTA1             MANAGER
 DOM2             FTA2             MANAGER
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Domains**
2. Select **All Domains in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**

Standard format

DOMAIN
The name of the domain to which this information applies.

MANAGER
The name of the domain manager.

PARENT
The name of the parent domain.

Info format

DOMAIN
The name of the domain to which this information applies.

MEMBER-CPUS
The names of the workstations in the domain.

CPU-TYPE
The type of each workstation: MASTER, MANAGER, FTA, S-AGENT, X-AGENT, or BROKER.

showfiles

Displays information about file dependencies. A file dependency occurs when a job or job stream is dependent on the existence of one or more files before it can begin running.

The displayed information is updated only as long as Tivoli Workload Scheduler (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

Syntax

{**showfiles** | **sf**} [[*workstation*#]*file*]
[;*state*[;...]]
[**;keys**]
[**;offline**]

{**showfiles** | **sf**} [[*workstation*#]*file*]
    [;*state*[;...]]
    [;**deps**[;**keys** | **info** | **logon**]]
    [;**offline**]

Arguments

workstation
: Specifies the name of the workstation on which the file exists. The default is the workstation on which **conman** is running. Wildcard characters are permitted.

file
: Specifies the name of the file. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). The default is to display all file dependencies. Wildcard characters are permitted.

state
: Specifies the state of the file dependencies to be displayed. The default is to display file dependencies in all states. The states are as follows:

yes File exists and is available.

no File is unavailable, or does not exist.

? Availability is being checked.

<blank>
The file has not yet been checked, or the file was available and used to satisfy a job or job stream dependency.

keys Displays a single column list of the objects selected by the command.

deps Displays information in the **deps** format. Use **keys**, **info**, or **logon** to modify the display.

offline
Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

Results

The output of the command is produced in three formats: **standard**, **keys**, and **deps**. The arguments **keys**, **info**, and **logon** modify the **deps** display.

Examples

To display the status of a file dependency for `d:\apps\mis\lib\data4`, run the following command:

```
showfiles d:\apps\mis\lib\data4
```

To display **offline** the status of all file dependencies on all workstations in the `deps` format, run the following command:

```
sf @#@;deps;offline
```

To display the status of all file dependencies on all workstations in the `deps` format, run the following command:

```
sf @#@;deps
```

A sample output is the following:

```
                                                     (Est) (Est)
Workstation Job Stream SchedTime  Job      State Pr Start Elapse ReturnCode Dependencies

MASTER#/test/^LFILEJOB^ Dependencies are:
 MASTER     #LFILEJOB  0600 11/26 ******** READY 10
                                  LFILEJOB HOLD  10 (11/26)                ^LFILEJOB^

MASTER#/usr/home/me10_99/`/usr/home/me10_99/bin/parms FILE_JS1` Dependencies are:
 MASTER     #FILE_JS1  0600 11/26 ******** HOLD  10 (11/26)                parms FILE_JS1`
                                  FILE_JS1 HOLD  10 (11/26)

MASTER#/usr/home/me10_99/`/usr/home/me10_99/bin/parms FILE_JOB1` Dependencies are:
 MASTER     #FILE_JOB1 0600 11/26 ******** READY 10
                                  FILE_JB1 HOLD  10 (11/26)                parms FILE_JB1`
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Workload Dependencies→Monitor Files**
2. Select **All Files in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**

Standard format

Exists The state of the file dependency.

File Name
The name of the file.

Keys format

Files are listed with one file on each line. Directory names are not included. Each file is listed in the following format:

```
workstation#file
```

Deps format

Files are listed followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Jobs and job streams that have `file` dependencies are listed with one on each line, in the following format:

```
workstation#jstream[.job]
```

Deps;info format

Files are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Files are listed followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

showjobs

Displays information about jobs.

The displayed information is updated only as long as Tivoli Workload Scheduler (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showjobs** | **sj**} [*jobselect*]
  [**;keys** | **info** | **step** | **logon** | **crit** | **keys retcod**]
  [**;short** | **single**]
  [**;offline**]
  [**;showid**]

{**showjobs** | **sj**} [*jobselect*]
  [**;deps**[**;keys** | **info** | **logon**]]
  [**;short** | **single**]
  [**;offline**]
  [**;showid**]
  [**;props**]

{**showjobs** | **sj**} [*jobselect* |
  [**workstation#**]*jobnumber.hhmm*]
  [**;stdlist**[**;keys**]]
  [**;short** | **single**]
  [**;offline**]
  [**;showid**]
  [**;props**]

Arguments

crit Displays information in the **crit** format.

deps Displays information in the deps format; that is, the jobs used in *follows* dependencies are listed followed by the dependent jobs and job streams. Jobs are listed in the basic `showjobs` format. Job streams are listed in the basic `showschedules` format. Use "keys", "info", or "logon" to modify the "deps" display.

hhmm The time the job started. Use this, together with the **stdlist** and **single** arguments, to display a specific instance of the job.

info Displays information in the **info** format.

jobnumber
The job number.

jobselect
See “Selecting jobs in commands” on page 296.

keys Displays a single column list of the objects selected by the command.

logon Displays information in the **logon** format.

offline
Sends the output of the command to the **conman** output device. For information about this device, see “Offline output” on page 292.

props Displays the following information about the specified job instance, you must have display access to the props of the specified job instance being shown:

General Information

- Job
- Workstation
- Task
- Task Type
- Job Stream
- Job Stream Workstation
- Scheduled Time
- Priority
- Login
- Monitored
- Requires Confirmation
- Interactive
- Critical

Runtime Information

- Status
- Internal Status
- Not Satisfied Dependencies
- Job Number
- Rerun Options
- Information
- Promoted
- Return Code
- Return Code Mapping Expression

Time Information

- Actual Start
- Earliest Start
- Latest Start Action
- Critical Latest Start
- Deadline
- Repeat Range
- Actual Duration
- Estimated Duration

Recovery Information

- Action
- Message
- Job Definition
- Workstation

Extra Information

This section shows additional properties specific for shadow jobs and jobs defined by JSDL. For shadow jobs it contains the following information:

For distributed shadow jobs:
- Remote Job Scheduled Time
- Remote Job
- Remote Job Stream
- Remote Job Stream Workstation

For z/OS shadow jobs:
- Remote Job Scheduled Time
- Remote Job
- Remote Job Workstation
- Remote Job Error Code

For more information, see “How the shadow job status changes after the bind is established” on page 571.

Note: Information on archived jobs is not retrievable using the **props** option.

retcod Displays the return code for the job. This argument must be used in conjunction with the **keys** argument, for example:

```
%sj @; keys retcod
```

short Shortens the display for **every** and **rerun** jobs to include only the following:
- The first iteration
- Jobs in different states
- Exactly matched jobs

Note: This field shows the specific properties if the job is a shadow job or a job defined by JSDL.

showid
Displays for each job stream the job stream identifier.

single Selects only the parent job in a chain that can include reruns, repetitions, and recovery jobs. The job must be identified by job number in *jobselect*. This is useful with the **stdlist** option.

stdlist Displays information in the **stdlist** format. Use the **keys** argument to modify the display.

Note: Information on archived jobs is not retrievable using the **stdlist** option.

step Displays information in the **step** format.

workstation
The name of the workstation on which the job runs. Wildcard characters are permitted.

Results

The output of the **showjobs** command is produced in eight formats: **standard**, **keys**, **info**, **step**, **logon**, **deps**, **crit**, and **stdlist**. The **keys**, **info**, **crit**, and **logon** arguments modify the displays.

Examples

To display the status of all jobs in the acctg job stream on workstation site3, you can run the showjobs command in one of these two formats:

```
showjobs site3#acctg.@
```

or:

```
showjobs site3#acctg
```

To display the status of job JBA belonging to job stream TEST1 on workstation CPUA, on which you are running **conman**, and ask to show the job stream identifier for the job stream, run the following command:

```
sj CPUA#TEST1(0900 02/19/06).JBA
```

A sample output for this command is the following:

```
Workstation Job Stream SchedTime Job State Pr Start Elapse ReturnCode Dependencies

CPUA        #TEST1    0900 02/19 *** HOLD   0(02/19)                {02/20/06}; -TEST-
                                 JBA HOLD  66(14:30)                J2(0600 02/24/06).JB1
```

The **at** dependency is shown as (14:30) in the Start column and the follows dependency from the job J2(0600 02/24/06).JB1 for job JOBA is shown in the Dependencies column.

In the Dependencies column the date enclosed in braces, {02/20/06}, indicates that the job stream instance has been carried forward and the date indicates the day when the job stream instance was added to the production plan for the first time.

To display the status of jobs belonging to job stream JSDOC on workstation site3, on which you are running **conman**, and ask to show the job stream identifier for the job stream, run the following command:

```
%sj JSDOC.@;showid
```

A sample output for this command is the following:

```
Workstation Job Stream SchedTime Job State Pr Start Elapse ReturnCode Dependencies

site3        #JSDOCOM 0600 11/26 *** SUCC  10 11/26 00:01            {0AAAAAAAAAAAACRZ}
                                JDOC SUCC  10 11/26 00:01       0    #J25565
```

The job stream identifier 0AAAAAAAAAAAACRZ for job stream JDOCOM is shown in the Dependencies column.

Note: The time or date displayed in the **Start** column is converted in the time zone set on the workstation where the job stream is to run.

To display the status of jobs belonging to job stream JSDOCOM on workstation site3, and ask to show the information about the user ID under which the job runs, run the following command:

```
sj site3#JSDOCOM.@;logon
```

A sample output for this command is the following:

```
Workstation Job Stream SchedTime  Job    State Job#    Logon   ReturnCode
site3       #JSDOCOM  0600 11/26
                                  JDOCOM SUCC  #J25565 me10_99     0
```

To display the status of all jobs in the HOLD state on all workstations, in the **deps** format, run the following command:

```
sj @#@.@+state=hold;deps
```

a sample output is the following:

```
Workstation Job Stream SchedTime Job  State Pr Start Elapse RetCode Dependencies

CPUA#JS2.JOBB Dependencies are:


CPUA        #JS21     0900 02/19 ***** HOLD  0(02/19)              {02/20/06}; -TEST- JOBA HOLD 66(14:30)
                                                                                JS22(0600 02/24/06).JOBB

CPUA#JS25.JOBC Dependencies are:


CPUA       #JS25      0600 02/24 ***** HOLD 10(02/24)              {02/20/06}
                                 jobaa HOLD 10(02/24)(00:01)        TEST1; JOBC          TEST2; JOB1
                                                                                JS18(0600 02/24/06).@

CPUA#JS25.JOB1 Dependencies are:


CPUA       #JS25      0600 02/24 ***** HOLD 10(02/24)               {02/20/06}
                                 JOBC  HOLD 10(02/24)(00:01)         JOB1
                                 jobaa HOLD 10(02/24)(00:01)         TEST1; JOBC         TEST2; JOB1
JS18(0600 02/24/06).@
```

To display the log from the standard list files for the job J25 in the job stream JS25(0600 02/19/06) on workstation CPUA, running in a UNIX environment, run the following command:

```
sj CPUA#JS25(0600 02/19/06).J25;stdlist
```

The output is the following:

```
===============================================================
= JOB       : CPUA#JS25[(0600 02/19/06),(0AAAAAAAAAAAABQM)].J25
= USER      : tme10_99
= JCLFILE   : ls
= Job Number: 28630
= Mon 02/20/06 07:57:37 PST
===============================================================
Tivoli Workload Scheduler (UNIX)/JOBMANRC
AWSBIS307I Starting /usr/home/tme10_99/jobmanrc ls

Tivoli Workload Scheduler (UNIX)/JOBINFO 8.4 (9.5)
Installed for user "tme10_99".
Locale LANG set to the following: "en"
...
... <stdout, stderr, and echoed commands>
...
AWSBIS308I End of job
===============================================================
= Exit Status           : 0
= System Time (Seconds) : 0     Elapsed Time (Minutes) : 0
= User Time (Seconds)   : 0
= Mon 02/20/06 07:57:38 PST
===============================================================
```

where:

Exit Status
Is the status of the job when it completed.

Elapsed Time
Is the elapsed time for the job.

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 **System Time** and **User Time** fields are used only in UNIX. Their values in Windows are always set to **0**. This is because, in Windows, the **joblnch.exe** process runs in a very short time, which can be considered null.

To display the properties of the shadow job with job number 546863237, run the following command:

```
sj 546863237;props
```

A sample output for this command is the following:

```
General Information
  Job = D_SHADOW_JOB
  Workstation = REMENG1
  Task =
    <jsdl:jobDefinition
          xmlns:jsdl="http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl"
          xmlns:dshadow="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow"
          xsi:schemaLocation="http://www.ibm.com/xmlns/prod/scheduling/1.0/dshadow">
       <jsdl:application name="distributedShadowJob">
          <dshadow:DistributedShadowJob>
             <dshadow:JobStream>JS1</dshadow:JobStream>
             <dshadow:Workstation>MYWKST</dshadow:Workstation>
             <dshadow:Job>JOBDEF1</dshadow:Job>
             <dshadow:matching>
                <dshadow:previous/>
             </dshadow:matching>
          </dshadow:DistributedShadowJob>
       </jsdl:application>
    </jsdl:jobDefinition>
  Task Type = distributedShadowJob
  Job Stream = JSDIST
  Job Stream Workstation = MYWKST
  Scheduled Time = 2010/08/11 06:00 TZ CEST
  Priority = 10
  Login =
  Monitored = No
  Requires Confirmation = No
  Interactive = No
  Critical = No
Runtime Information
  Status = Undecided
  Internal Status = DONE
  Not Satisfied Dependencies = 0
  Job Number = 546863237
  Rerun Options =
  Information =
  Promoted = No
  Return Code =
  Return Code Mapping Expression =
Time Information
  Actual Start = 2010/08/11 12:00 TZ CEST
  Earliest Start =
  Latest Start Action =
  Critical Latest Start =
  Deadline =
  Repeat Range =
  Actual Duration =
  Estimated Duration = 00:02 (hh:mm)
Recovery Information
  Action =
  Message =
  Job Definition =
  Workstation =
Extra Information
  Remote Job Scheduled Time = 08/11/2010 06:00 TZ CEST
  Remote Job = JOBDEF1
  Remote Job Stream = JS1
  Remote Job Stream Workstation = MYWKST
```

The following example displays the status of the job `dbseload` with a return code of **7** and a state of SUCCESSFUL:

```
$conman sj workstation#DAILY_DB_LOAD
Tivoli Workload Scheduler (UNIX)/CONMAN 8.4 (1.36.2.22) Licensed Materials -
Property of IBM(R)
5698-WSH
(C) Copyright IBM Corp 1998, 2007 All rights reserved.
US Government User Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
IBM is a registered trademark of International Business Machines
Corporation in the United States, other countries, or both.
Installed for user "tme10_99".
Locale LANG set to the following: "en"
Scheduled for (Exp) 02/20/06 (#35) on CPUA.
Batchman LIVES. Limit:50,Fence:0,Audit Level:0
sj workstation#DAILY_DB_LOAD
(Est) (Est)
CPU Schedule Job State Pr Start
Elapse Dependencies Return Code
WORKSTATION #DAILY_DB_LOAD ****************************** SUCC 10 22:11
00:04
DATASPLT SUCC 10 22:11
00:01 #J17922 0
DATAMRGE ABEND 10 22:12
00:01 #J17924 1
CHCKMRGE SUCC 10 22:12
00:01 #J17926 0
DATACLNS SUCC 10 22:12
00:01 #J17932 0
DATARMRG SUCC 10 22:13
00:01 #J18704 0
DBSELOAD SUCC 10 22:13
00:01 #J18706 7
DATAREPT SUCC 10 22:13
00:01 #J18712 0
DATARTRN SUCC 10 22:14
00:01 #J18714 0
$
```

The following example displays the return code for a specific job named `workstation#daily_db_load.dbseload`:

```
$ conman sj workstation#daily_db_load.dbseload\;keys\;retcod

Tivoli Workload Scheduler (UNIX)/CONMAN 8.4 (1.36.2.22) Licensed Materials -
Property of IBM(R)
5698-WSH
(C) Copyright IBM Corp 1998, 2007 All rights reserved.
US Government User Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
IBM is a registered trademark of International Business Machines
```

```
Corporation in the United States, other countries, or both.
Installed for user "tme10_99".
Locale LANG set to the following: "en"
Scheduled for (Exp) 02/20/06 (#35) on CPUA.
Batchman LIVES. Limit:50,Fence:0,Audit Level:0
sj workstation#daily_db_load.dbseload;keys;retcod 8
$
```

The *retcod* feature when integrated into a script can become quite powerful.

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Monitor Jobs**
2. Select **All Jobs in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**

Standard format

CPU The workstation on which the job runs.

Schedule
The name of the job stream.

SchedTime
The time and date when the job was scheduled to run in the plan.

Job The name of the job. The following notation may precede a job name:

>> rerun as
A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> rerun step
A job that was rerun with the **rerun ;step** command.

>> every run
The second and subsequent runs of an every job.

>> recovery
The run of a recovery job.

State The state of the job or job stream. Job states are as follows:

ABEND
The job ended with a non-zero exit code.

ABENP
An **abend** confirmation was received, but the job is not completed.

ADD The job is being submitted.

DONE
The job completed in an unknown state.

ERROR
For internetwork dependencies only, an error occurred while checking for the remote status.

EXEC The job is running.

EXTRN
For internetwork dependencies only, the status is unknown. An error occurred, a rerun action was just performed on the job in the `EXTERNAL` job stream, or the remote job or job stream does not exist.

FAIL Unable to launch the job.

FENCE
The priority of the job is below the fence.

HOLD
The job is awaiting dependency resolution.

INTRO
The job is introduced for launching by the system.

PEND The job completed, and is awaiting confirmation.

READY
The job is ready to launch, and all dependencies are resolved.

SCHED
The **at** time set for the job has not been reached.

SUCC The job completed with an exit code of zero.

SUCCP
A SUCC confirmation was received, but the job is not completed.

WAIT The job is in the WAIT state (extended agent).

Job stream states are as follows:

ABEND
The job stream ended with a nonzero exit code.

ADD The job stream was added with operator intervention.

CANCL
The job stream was cancelled.

CANCELP
The job stream is pending cancellation. Cancellation is deferred until all of the dependencies, including an at time, are resolved.

ERROR
For internetwork dependencies only, an error occurred while checking for the remote status.

EXEC The job stream is running.

EXTRN
For internetwork dependencies only, the job stream is in a remote Tivoli Workload Scheduler network and its state is unknown. An error occurred, a rerun action was just performed on the `EXTERNAL` job stream, or the INET job or job stream does not exist.

HOLD
The job stream is awaiting dependency resolution.

READY
The job stream is ready to launch and all dependencies are resolved.

STUCK
Execution of the job stream was interrupted. No jobs are launched without operator intervention.

SUCC The job stream completed successfully.

Pr The priority of the job stream or job. A plus sign (+) preceding the priority means the job has been launched.

(Est)Start

The start time of the job stream or job. Parentheses indicate an estimate of the start time. If the command is performed on the same day when the job is scheduled to run, the **Start** parameter displays a time as (Est)Start. If the command is performed on a day different from the day when the job is scheduled to run, the **Start** parameter displays a date as (Est)Start. For example if you have the following job whose start time occurs on the same day when the job is scheduled to run:

```
SCHEDULE MASTERB1#JS_B
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 1700
:
MASTERB1#JOB1
 AT 1800
END
```

You receive the following output:

```
%sj @#@
                                             (Est)  (Est)
CPU      Schedule SchedTime  Job   State Pr Start  Elapse RetCode Deps
MASTERB1#JS_B     1700 08/18 ***** HOLD  10(17:00)
                             JOB1  HOLD  10(18:00)
```

For example if you have the following job whose start time occurs on a day different from the day when the job is scheduled to run:

```
SCHEDULE MASTERB1#JS_A
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 0400
:
MASTERB1#JOB_A
 AT 0500
END
```

You receive the following output:

```
%sj @#@
                                             (Est)  (Est)
CPU      Schedule SchedTime  Job   State Pr Start  Elapse RetCode Deps
MASTERB1#JS_A     0400 08/19 ***** HOLD  10(08/19)
                             JOB_A HOLD  10(08/19)
```

(Est)Elapse

The run time of the job stream or job. Parentheses indicate an estimate based on logged statistics.

dependencies

A list of job dependencies and comments. Any combination of the following can be listed:

- For a `follows` dependency, a job stream or job name is displayed.

  If the job or job stream is a pending predecessor, its name is followed by a `[P]`.

  In case of an orphaned dependency an `[O]` is displayed.

  For more information on pending predecessors and orphaned dependencies refer to "Managing external follows dependencies for jobs and job streams" on page 52.
- For an `opens` dependency, the file name is displayed. If the file resides on an extended agent and its name is longer than 25 characters, only the last 25 characters are displayed.

- For a `needs` dependency, a resource name enclosed in hyphens (-) is displayed. If the number of units requested is greater than one, the number is displayed before the first hyphen.
- For a **deadline** time, the time preceded by an angle bracket (<) is displayed.
- For an **every** rate, the repetition rate preceded by an ampersand (&) is displayed.
- For an **until** time, the time preceded by an angle bracket (<) is displayed.
- For a `prompt` dependency, the prompt number is displayed in the format #*num*. For global prompts, the prompt name follows in parentheses.
- For running jobs, the process identification number (PID) is displayed in the format **#J***nnnnn*.
- Jobs submitted on UNIX using the Tivoli Workload Scheduler **at** and **batch** commands are labeled **[Userjcl]**.
- When reporting time dependencies the **showjobs** command shows in the **Start** column:
  - Only the time *hh:mm* if the day when the time dependencies is set matches with the day when the **showjobs** command is run.
  - Only the date *MM/DD* if the day when the time dependencies is set does not match with the day when the **showjobs** command is run.
- Cancelled jobs are labeled **[Cancelled]**.
- Jobs cancelled with **;pend** option are labeled **[Cancel Pend]**.
- Jobs with expired **until** times, including jobs cancelled with **;pend** option, are labeled **[Until]**.
- **[Recovery]** means that operation intervention is required.
- **[Confirmed]** means that confirmation is required because the job was scheduled using the **confirm** keyword.
- **[Script]** applies to end-to-end networks only; it means that this job has a centralized script and that Tivoli Workload Scheduler for z/OS has not yet downloaded it to the agent.

Keys format

Job names are listed one on each line in the following format:

```
workstation#jstream hhmm mm/dd.job
```

for example:

```
CPU    Schedule SchedTime  Job     State Pr Start Elapse RetCode Deps

MYCPU+#SCHED_F+ 0600 03/04 ******* HOLD  55(03/04)              [03/04/06]; #33
               (M235062+#)JOBMDM  HOLD  30(03/04)              #1(PRMT3);-16 JOBSLOTS-

MYCPU+#SCHED_F+ 1010 03/04 ******* HOLD  55(03/04)              [03/04/06]; #34
               (M235062+#)JOBMDM  HOLD  30(03/04)              #1(PRMT3);-16 JOBSLOTS-
```

Info format

CPU The workstation on which the job runs.

Schedule
The name of the job stream.

SchedTime
The time and date when the job was scheduled to run in the plan.

Job The name of the job. The following notation might precede a job name:

>> rerun as
A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> rerun step
A job that was rerun with the **rerun ;step** command.

>> every run
The second and subsequent runs of an every job.

>> recovery
The run of a recovery job.

Job File
The name of the script or executable file of the job. Long file names might wrap, causing incorrect paging. To avoid this, pipe the output to **more**.

Opt The job recovery option, if any. The recovery options are **RE** for rerun, **CO** for continue, and **ST** for stop.

Job The name of the recovery job, if any.

Prompt
The number of the recovery prompt, if any.

For example:

```
conman "sj;info | more
```

produces a sample output like the following:

```
 --------Restart---------
CPU      Schedule SchedTime  Job      JobFile                Opt  Job Prompt
M235062+#SCHED_22 1010 03/06
                             JOBMDM   /usr/acct/scripts/gl1
                 (B236153+#)JOB_FTA  echo job12
M235062+#SCHED_22 0600 03/07
                             JOBMDM   /usr/acct/scripts/gl1
                 (B236153+#)JOB_FTA  echo job12
M235062+#FINAL    0559 03/07
                             MAKEPLAN /home/tws84/MakePlan TWSRCMAP:(RC=0) OR (RC=4)
                             SWITCHP+ /home/tws84/SwitchPlan
                             CREATEP+ /home/tws84/CreatePostReports
                             UPDATES+ /home/tws84/UpdateStats
M235062+#SCHED12  1010 03/06
                             JOBMDM   /usr/acct/scripts/gl1
                 (B236153+#)JOB_FTA  echo job12
```

Step format

This format is not supported in Windows.

CPU The workstation on which the job runs.

Schedule
The name of the job stream.

SchedTime
The time and date when the job was scheduled to run in the plan.

Job The name of the job. The following notation might precede a job name:

>> rerun as
A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> repeated as
The second and subsequent runs of an **every** job.

State The state of the job or job stream. See “Standard Format” for information about state.

Return code
The return code of the job.

Job# The process identification number displayed as **#J***nnnnn*.

Step A list of descendant processes that are associated with the job. For extended agent jobs, only host processes are listed.

Logon format

CPU The workstation on which the job runs.

Schedule
The name of the job stream.

SchedTime
The time and date when the job was scheduled to run in the plan.

Job The name of the job. The following notation might precede a job name:

>> rerun as
A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> repeated as
The second and subsequent runs of an **every** job.

State The state of the job or job stream. See “Standard Format” for information about state.

Return code
The return code of the job.

Job# The process identification number displayed as **#J***nnnnn*.

Logon The user name under which the job runs.

Stdlist format

A standard list file is created automatically by **jobmon** in Windows or **jobman** in UNIX, for each job that **jobmon** and **jobman** launches. You can display the contents of the standard list files using **conman**. A standard list file contains:

- Header and trailer banners.
- Echoed commands.
- The stdout output of the job.
- The stderr output of the job.

To specify a particular date format to be used in the standard list files, change the IBM Tivoli Workload Scheduler date format before creating the standard list files. You do this by modifying the date locale format.

Depending on your environment, change the date locale format by performing the steps listed below:

- In UNIX, set the LANG variable in the environment when **netman** starts. If the LANG variable is not set, the operating system locale is set by default to "C".
- In Windows, perform the following steps:
  1. Go to Control Panel→Regional Options and set your locale (location).

2. Right-click on "My Computer", go to Properties, click on "Advanced", go to Environment Variables and set the LANG variable as a system variable.
3. Shut down and restart the system.

The standard list files for the selected jobs are displayed.

Stdlist;keys format

The names of the standard list files for the selected jobs are listed, one on each line.

Crit format

CPU The workstation on which the job runs.

Schedule
The name of the job stream.

SchedTime
The time and date when the job was scheduled to run in the plan.

Job The name of the job. The following notation might precede a job name:

>> rerun as
A job that was rerun with the **rerun** command, or as a result of automatic recovery.

>> repeated as
The second and subsequent runs of an **every** job.

State The state of the job or job stream. See “Standard Format” for information about state.

Pr The priority of the job stream or job. A plus sign (+) preceding the priority means the job has been launched.

(Est)Start
The start time of the job stream or job. Parentheses indicate an estimate of the start time. If the start time is more than 24 hours in the past or future, the date is listed instead of the time.

(Est)Elapse
The run time of the job stream or job. Parentheses indicate an estimate based on logged statistics.

CP Indicates if the job is flagged as critical (**C**) and/or promoted (**P**).

CritStart
The latest time a job can start without impacting the deadlines of mission critical successors.

For example, the result of the following generic command:

```
%sj @#@;crit
```

is:

```
                                              (Est)  (Est)        Crit
CPU     Schedule SchedTime  Job      State Pr Start  Elapse  CP   Start

MYCPU_F+#JSA     1600 03/05 ******** HOLD  10
                            JOBA1    HOLD  10                CP   1759 03/05
                            JOBA2    HOLD  10                     1758 03/05
                            JOBA3    HOLD  10                     1757 03/05
                            JOBA4    HOLD  10                C    1659 03/05
```

Note that:

- The **C** flag applies only to jobs defined as critical in their job stream definition. It is set at plan or **`submit`** time.
- The **P** flag applies to both critical jobs and to their predecessors (which are jobs that are not defined as critical but might nonetheless impact the timely completion of a successor critical job). It is set at execution time if the job was promoted.
- Both critical jobs and critical predecessors have a critical start time.

  The scheduler calculates the critical start time of a critical job by subtracting its estimated duration from its deadline. It calculates the critical start time of a critical predecessor by subtracting its estimated duration from the critical start time of its next successor. Within a critical network the scheduler calculates the critical start time of the critical job first and then works backwards along the chain of predecessors. These calculations are reiterated as many times as necessary until the critical job has run.

Deps format

Jobs used in `follows` dependencies are listed followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Jobs and job streams that have `follows` dependencies are listed, one on each line.

Deps;info format

Jobs used in `follows` dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Jobs used in `follows` dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

showprompts

Displays information about prompts.

The displayed information is updated only as long as Tivoli Workload Scheduler (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showprompts** | **sp**} [*promptselect*]
    [**;keys**]
    [**;offline**]

{**showprompts** | **sp**} [*promptselect*]
    [**;deps**[**;keys** | **info** | **logon**]][**;offline**]

Arguments

promptselect

[*promptname* | [*workstation#*]*msgnum*][**;***state*[**;**...]]

promptname
: Specifies the name of a global prompt. Wildcard characters are permitted.

workstation
: Specifies the name of the workstation on which an unnamed prompt is issued. The default is the workstation on which **conman** is running.

msgnum
: Specifies the message number of an unnamed prompt.

state
: Specifies the state of prompts to be displayed. The states are as follows:

  YES The prompt was replied to with **y**.

  NO The prompt was replied to with **n**.

  ASKED
  The prompt was issued, but no reply was given.

  INACT
  The prompt has not been issued.

keys Displays a single column list of the objects selected by the command.

deps Displays information in the **deps** format. Use **keys**, **info**, or **logon** to modify the display.

info Displays information in the **info** format.

logon Displays information in the **logon** format.

offline
Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

Note: Prompt numbers assigned to both global and local prompts change when the production plan is extended.

Results

The output of the command is produced in three formats: **standard**, **keys**, and **deps**. The arguments **keys**, **info**, and **logon** modify the **deps** display.

Examples

To display the status of all `prompt` issued on the workstation on which you are running **conman**, run the following command:

```
showprompts
```

a sample is the following:

```
State Message or Prompt
ASKED 1(PRMT3) !continue?
INACT 3(CPUA#SCHED_12[(0600 03/12/06),
      (0AAAAAAAAAAAAABST)]) Are you ready to process job1?
```

```
INACT 5(CPUA#SCHED_12[(1010 03/12/06),(0AAAAAAAAAAAABSU)])
      Are you ready to process job2?
INACT 7(CPUA#SCHED_22[(0600 03/12/06),(0AAAAAAAAAAAABTR)])
      Are you ready to process job3?
```

To display the status of all mis prompts that have been issued, in the **deps** format, run the following command:

```
sp mis@;asked;deps
```

To display the status of prompt number 7 on workstation CPUA, run the following command:

```
sp CPUA#7
```

The output of the command is:

```
INACT 7(CPUA#SCHED_22[(0600 03/12/06),(0AAAAAAAAAAAABTR)])
      Are you ready to process job3?
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Workload Dependencies→Monitor Prompts**
2. Select **All Prompts in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**

Standard format

State The state of the prompt.

Message or Prompt
For named prompts, the message number, the name, and the text of the prompt. For unnamed prompts, the message number, the name of the job or job stream, and the text of the prompt.

Keys format

The prompts are listed one on each line. Named prompts are listed with their message numbers and names. Unnamed prompts are listed with their message numbers, and the names of the jobs or job streams in which they appear as dependencies.

Deps format

Prompts used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Jobs and job streams that have prompt dependencies are listed one on each line.

Deps;info format

Prompts used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Prompts used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

showresources

Displays information about resources.

The displayed information is updated only as long as Tivoli Workload Scheduler (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showresources** | **sr**} [[*workstation*#]*resourcename*]
[**;keys**]
[**;offline**]

{**showresources** | **sr**} [[*workstation*#]*resourcename*]
[**;deps**[**;keys** | **info** | **logon**]]
[**;offline**]

Arguments

workstation
Specifies the name of the workstation on which the resource is defined. The default is the workstation on which **conman** is running.

resourcename
Specifies the name of the resource. Wildcard characters are permitted.

keys Displays a single column list of the objects selected by the command.

deps Displays information in the **deps** format. Use **keys**, **info**, or **logon** to modify the display.

info Displays information in the **info** format.

logon Displays information in the **logon** format.

offline
Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

Results

The output of the command is produced in three formats: **standard**, **keys**, and **deps**. The arguments **keys**, **info**, and **logon** modify the **deps** display.

Examples

To display information about all resources on the workstation on which you are running **conman**, run the following command:

```
showresources
```

A sample output is:

```
CPU#Resource      Total Available    Qty UsedBy
CPUA #JOBSLOTS    16       16        No holders of this resource
```

To display information about the `jobslots` resource on workstation `CPUA` in the **deps** format, run the following command:

```
sr CPUA#JOBSLOTS;deps
```

A sample output is the following:

```
                                             (Est)  (Est)
Workstation Job Stream SchedTime Job  State Pr Start Elapse RetCode Dependencies

CPUA       #JOBSLOTS Dependencies are:


  FTAA     #SCHED_F+ 0600 03/04 ****** HOLD 55(03/04)             [03/04/06];#33
                  (CPUA#)JOBMDM HOLD 30(03/04)             #1(PRMT3);-16 JOBSLOTS-

  FTAA     #SCHED_F+ 1010 03/04 ****** HOLD 55(03/04)             [03/04/06];#34
                  (CPUA#)JOBMDM HOLD 30(03/04)             #1(PRMT3);-16 JOBSLOTS-

  FTAA     #SCHED_F+ 0600 03/05 ****** HOLD 55(03/05)             [03/04/06];#35
                  (CPUA#)JOBMDM HOLD 30(03/05)             #1(PRMT3);-16 JOBSLOTS-
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Workload Dependencies→Monitor Resources**
2. Select **All Resources in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**

Standard format

CPU The workstation on which the resource is defined.

Resource
The name of the resource.

Total The total number of defined resource units.

Available
The number of resource units that have not been allocated.

Qty The number of resource units allocated to a job or job stream.

Used By
The name of the job or job stream.

Keys format

The resources are listed one on each line.

Deps format

Resources used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Jobs and job streams that have `resource` dependencies are listed one on each line.

Deps;info format

Resources used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Resources used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

showschedules

Displays information about job streams.

The displayed information is updated only as long as Tivoli Workload Scheduler (batchman) is running. Whether batchman is up or down is confirmed on screen by the `Batchman LIVES` or `Batchman down` message when you issue the `conman start` command.

You must have ***list*** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

Syntax

{**showscheds** | **ss**} [*jstreamselect*]
    [**;keys**]
    [**;offline**]
    [**;showid**]

{**showscheds** | **ss**} [*jstreamselect*]
    [**;deps**[**;keys** | **info** | **logon**]]
    [**;offline**]
    [**;showid**]

Arguments

jstreamselect
: See "Selecting job streams in commands" on page 305.

keys Displays a single column list of the objects selected by the command.

deps Displays information in the deps format; that is, the job streams used in *follows* dependencies are listed followed by the dependent jobs and job streams. Jobs are listed in the basic `showjobs` format. Job streams are listed in the basic `showschedules` format. Use "keys", "info", or "logon" to modify the "deps" display.

info Displays information in the **info** format.

logon Displays information in the **logon** format.

offline
: Sends the output of the command to the **conman** output device. For information about this device, see "Offline output" on page 292.

showid
: Displays for each job stream the job stream identifier.

Results

The output of the command is produced in three formats: standard, **keys**, and **deps**. The arguments **keys**, **info**, and **logon** modify the **deps** display. The list displayed in the output of the command does not include jobs that were rerun in previous scheduling processes, but the total shown at the end does.

Examples

To display the status of job stream CLEM_DOCOM on workstation site3, and ask for the job stream identifier run the following command:

```
%ss @#JS_DOCOM ;showid
```

A sample output of this command is the following:

```
                                          (Est) (Est)  Jobs  Sch
Workstation Job Stream SchedTime State Pr Start Elapse #  OK Lim
site3      #JS_DOCOM  0600 11/26 SUCC  10 11/26 00:01  1   1      {0AAAAAAAAAAAACRZ}
```

To display the status of all job streams in the HOLD state on the workstation on which you are running **conman**, run the following command:

```
showschedules @+state=hold
```

A sample output for this command is the following:

```
                                            (Est)  (Est)  Jobs   Sch
Workstation Job Stream SchedTime  State Pr  Start  Elapse #  OK  Lim
site3       #FILE_JS1  0600 11/26 HOLD  10 (11/26)        1   0       parms FILE_JS1`
```

To display the status of all job streams with name beginning with sched on workstation CPUA in the **deps;info** format, run the following command:

```
ss CPUA#sched@;deps;info
```

A sample output is the following:

```
   --------Restart---------
  CPU   Schedule SchedTime Job    JobFile              Opt   Job       Prompt

CPUA  #JS_FIRST1[(0600 03/10/06),(0AAAAAAAAAAAABVY)] Dependencies are:
  CPUA#MOD 0212 03/10
                        JOBMDM /usr/scripts/gl1(B236153+#)JOB_FTA1  echo Start gl1?
  CPUA#MOD 0251 03/10
                        JOBMDM /usr/scripts/gl2(B236153+#)JOB_FTA2  echo Start gl2?
```

To display **offline** the status of all job streams in the ABEND state on all workstations, run the following command:

```
ss @#@+state=abend;off
```

To display the status of all job streams on all workstations, run the following command:

```
%ss @#@
```

This is a sample output for the command:

```
                                             (Est)  (Est)    Jobs  Sch
Workstation Job Stream  SchedTime  State Pr  Start  Elapse  #  OK  Lim
site3       #JS_DOCOM   0600 11/26 SUCC  10  11/26  00:01   1   1
site3       #JS_SCRIPT  0600 11/26 SUCC  10  11/26  00:03   1   1
site2       #JS_PRED1   1000 11/26 SUCC  10  11/26  00:01   1   1
site3       #JS_SCRIPT1 0600 11/26 ABEND 10  11/26  00:01   1   0
site3       #LFILEJOB   0600 11/26 READY 10                 1   0
site1       #RES_100    0600 11/26 SUCC  10  11/26  00:09   1   1
site3       #FILE_JS1   0600 11/26 HOLD  10 (11/26)         1   0       parms FILE_JS1`
site3       #FILE_JOB   0600 11/26 SUCC  10  11/26  00:01   1   1
```

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Monitor→Monitor Job Streams**

2. Select **All Job Streams in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**

Standard format

CPU The workstation on which the job stream runs.

Schedule
The name of the job stream.

SchedTime
The time and date when the job stream was scheduled to run in the plan.

State The state of the job stream. The states are as follows:

ADD The job stream was added with operator intervention.

ABEND
The job stream ended with a nonzero exit code.

CANCL
The job stream was cancelled.

CANCELP
The job stream is pending cancellation. Cancellation is deferred until all of the dependencies, including an at time, are resolved.

ERROR
For internetwork dependencies only, an error occurred while checking for the remote status.

EXEC The job stream is running.

EXTRN
For internetwork dependencies only, the job stream is in a remote Tivoli Workload Scheduler network and its status is unknown. An error occurred, a rerun action was just performed on the `EXTERNAL` job stream, or the INET job or job stream does not exist.

HOLD
The job stream awaiting dependency resolution.

READY
The job stream is ready to launch and all dependencies are resolved.

STUCK
Job stream execution was interrupted. No jobs are launched without operator intervention.

SUCC The job stream completed successfully.

Pr The priority of the job stream.

(Est)Start
The start time of the job stream or job. Parentheses indicate an estimate of the start time. If the command is performed on the same day when the job stream is scheduled to run, the **Start** parameter displays a time as (Est)Start. If the command is performed on a day different from the day when the job stream is scheduled to run, the Start parameter displays a date as (Est)Start. For example if you have the following job stream whose start time occurs on the same day when the job stream is scheduled to run:

```
SCHEDULE MASTERB1#JS_B
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 1800
:
MASTERB1#JOB1
END
```

You receive the following output:

```
%ss @#@
                                      (Est)  (Est)    Jobs   Sch
CPU      Schedule SchedTime  State Pr Start  Elapse   #  OK  Lim
MASTERB1#JS_B     1800 08/18 HOLD  10(18:00)          1   0
```

For example if you have the following job stream whose start time occurs on a day different from the day when the job stream is scheduled to run:

```
SCHEDULE MASTERB1#JS_A
ON RUNCYCLE RULE1 "FREQ=DAILY;"
AT 0500
:
MASTERB1#JOB1
END
```

You receive the following output:

```
%ss @#@
                                      (Est)  (Est)  Jobs  Sch
CPU      Schedule SchedTime  State Pr Start  Elapse #  OK Lim
MASTERB1#JS_A     0500 08/19 HOLD  10(08/19)        1   0
```

(Est)Elapse
The run time of the job stream. Parentheses indicate an estimate based on logged statistics.

Jobs # The number of jobs in the job stream.

Jobs OK
The number of jobs that have completed successfully.

Sch Lim
The job stream's job limit. If one is not listed, no limit is in effect.

dependencies
A list of job stream dependencies and comments. Any combination of the following may be listed:

- For a `follows` dependency, a job stream or job name is displayed. If the job or job stream is a pending predecessor, its name is followed by a `[P]`.
- For an `opens` dependency, the file name is displayed. If the file resides on an extended agent, and its name is longer than 25 characters, only the last 25 characters are displayed.
- For a `needs` dependency, a resource name enclosed in hyphens (-) is displayed. If the number of units requested is greater than one, the number is displayed before the first hyphen.
- For an **until** time, the time preceded by an angled bracket (<).
- For a `prompt` dependency, the prompt number displayed as #*num*. For global prompts, the prompt name in parentheses follows.
- Cancelled job streams are labeled **[Cancelled]**.
- Job streams cancelled with the **;pend** option are labeled **[Cancel Pend]**.
- For a **deadline** time, the time preceded by an angle bracket (<) is displayed.

- Job streams that contain the **carryforward** keyword are labeled **[Carry]**.
- For job streams that were carried forward from the previous production plan, the original name and date are displayed in brackets.
- When reporting time dependencies the **showschedules** command shows in the **Start** column:
  - Only the time *hh:mm* if the day when the time dependencies is set matches with the day when the **showschedules** command is run.
  - Only the date *mm/dd* if the day when the time dependencies is set does not match with the day when the **showschedules** command is run.

  Note: The time or date displayed in the **Start** column is converted in the time zone set on the workstation where the job stream is to run.

Keys format

The job streams are listed one on each line.

Deps format

Job streams used as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the standard **showjobs** format. Job streams are listed in the standard **showschedules** format.

Deps;keys format

Job streams that have `follows` dependencies are listed one on each line.

Deps;info format

Job streams used in as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;info** format. Job streams are listed in the standard **showschedules** format.

Deps;logon format

Job streams used in as dependencies are listed, followed by the dependent jobs and job streams. Jobs are listed in the **showjobs;logon** format. Job streams are listed in the standard **showschedules** format.

shutdown

Unconditionally stops all the Tivoli Workload Scheduler production processes and services, including **batchman**, **jobman**, **netman**, **mailman**, **appservman**, all **mailman** servers, and all **writer** processes.

Even though this command does stop the **appservman** service, it does not stop the WebSphere Application Server services. To stop WebSphere Application Server services, run the **stopappserver** command. For more information, see "stopappserver" on page 393.

On Windows workstations, the **shutdown** command does not stop the **tokensrv** service.

Note: This command is not supported on remote engine workstations.

You must have ***shutdown*** access to the workstation.

Syntax

{**shutdown** | **shut**} [**;wait**]

Arguments

wait Waits until all processes have stopped before prompting for another command.

Comments

The **shutdown** command stops the processes only on the workstation on which **conman** is running. To restart **netman** only, run the **StartUp** command. For information about the **StartUp** command, see "StartUp" on page 464. To restart the entire process tree, run the following **conman** commands:

```
start
startappserver
startmon
```

You must run a **conman unlink @** command before executing a **shutdown** command.

Examples

To shut down production on the workstation on which you are running **conman**, run the following command:

```
unlink @
shutdown
```

To shut down production on the workstation on which you are running **conman** and wait for all processes to stop, run the following command:

```
unlink@;noask
shut ;wait
```

start

Starts Tivoli Workload Scheduler production processes, except for the event monitoring engine and WebSphere Application Server (see "startappserver" on page 387 and "startmon" on page 389 to learn about the commands that start these processes).

Note: Make sure **conman start** is not issued while either **JnextPlan** or **stageman** runs.

You must have ***start*** access to the workstation.

Syntax

start [*domain*!]*workstation*
  [**;mgr**]
  [**;noask**]
  [**;demgr**]

Arguments

domain Specifies the name of the domain in which workstations are started. Wildcard characters are permitted.

This argument is useful when starting more than one workstation in a domain. For example, to start all the agents in domain **stlouis**, use the following command:

```
start stlouis!@
```

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

workstation
: Specifies the name of the workstation to be started. Wildcard characters are permitted.

  This command is not supported on remote engine workstations.

mgr
: This can be entered only on the workstation on which **conman** is running. It starts the local workstation as the domain manager. The workstation becomes the new domain manager and the current domain manager becomes a fault-tolerant agent. This form of the command usually follows a **stop** command.

  Note: The preferred method of switching a domain manager is to use a **switchmgr** command. See "switchmgr" on page 410 for more information.

noask
: Specifies not to prompt for confirmation before taking action on each qualifying workstation.

demgr
: This option prevents the opening of external connections during the transition time between when an agent starts as an old domain manager, and when the **switchmgr** command is run, depriving the agent of the domain manager function. This option is run automatically, but until the old domain manager has processed the **switchmgr** event (in the case, for example, of delayed restart or restart after repairing a damaged agent), the *demgr* option **must** be used to start the old domain manager from the local command line. For more details on this option, see the *Tivoli Workload Scheduler Administration Guide*.

Comments

The **start** command is used at the start of each production period to restart Tivoli Workload Scheduler following preproduction processing. At that time it causes the autolinked fault-tolerant agents and standard agents to be initialized and started automatically. Agents that are not autolinked are initialized and started when you run a **link** command.

Assuming the user has ***start*** access to the workstations being started, the following rules apply:

- A user running **conman** on the master domain manager can start any workstation in the network.
- A user running **conman** on a domain manager other than the master can start any workstation in that domain and subordinate domains. The user cannot start workstations in peer domains.
- A user running **conman** on an agent can start workstations that are hosted by that agent.

Examples

Figure 23 on page 387 and Table 57 on page 387 below show the workstations started by **start** commands run by users in various locations in the network.

DM*n* are domain managers and **A***nn* are agents.

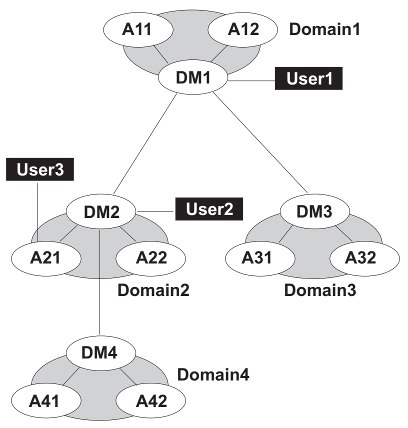


Figure 23. Example network

Table 57. Started workstations

Command	Started by User1	Started by User2	Started by User3
start @!@	All workstations are started	DM2 A21 A22 DM4 A41 A42	A21
start @	DM1 A11 A12	DM2 A21 A22	A21
start DOMAIN3!@	DM3 A31 A32	Not allowed	Not allowed
start DOMAIN4!@	DM4 A41 A42	DM4 A41 A42	Not allowed
start DM2	DM2	DM2	Not allowed
start A42	A42	A42	Not allowed
start A31	A31	Not allowed	Not allowed

startappserver

Starts the embedded WebSphere Application Server on the workstation.

Syntax

startappserver [*domain***!**]*workstation*
[;wait]

Arguments

domain Specifies the name of the domain of the workstation. Because workstations have unique names, the domain is not needed when starting the WebSphere Application Server on a specific workstation. Wildcard characters are permitted.

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

workstation
Specifies the name of the workstation where you want to start the monitoring engine. Wildcard characters are permitted. If no domain and workstations are specified, the action is on the local workstation.

wait Waits until WebSphere Application Server has started before prompting for another command.

Comments

Permission to `start` actions on `cpu` objects is required in the security file to be enabled to run this command.

WebSphere Application Server can also be started with the `StartUp` utility command.

starteventprocessor

Starts the event processing server on the master domain manager, backup master, or on a workstation installed as a backup master that functions as a plain fault-tolerant agent.

Syntax

{**starteventprocessor** | **startevtp**} [*domain*!]*workstation*

Arguments

domain Specifies the name of the domain of the workstation.

workstation
Specifies the name of the workstation where you want to start the event processing server. Wildcard characters are not permitted.

Comments

You can omit the workstation name if you run the command locally.

Permission to `start` actions on `cpu` objects is required in the security file to be enabled to run this command.

See also

In the Tivoli Dynamic Workload Console:
1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **More Actions→Start Event Processor**.

startmon

Starts the monman process that turns on the event monitoring engine on the workstation.

Syntax

{**startmon** | **startm**} [*domain*!]*workstation*
 [**;noask**]

Arguments

domain Specifies the name of the domain of the workstation. Because workstations have unique names, the domain is not needed when starting the monitoring engine on a specific workstation. Wildcard characters are permitted.

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

workstation
Specifies the name of the workstation where you want to start the monitoring engine. Wildcard characters are permitted.

noask Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

Permission to `start` actions on `cpu` objects is required in the security file to be enabled to run this command.

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **More Actions→Start Event Monitoring**.

status

Displays the **conman** banner and the Tivoli Workload Scheduler production status.

Syntax

{**status** | **stat**}

Results

Following the word **schedule** on the second line of output, the production plan (`Symphony` file) mode is shown in parentheses. The **Def** or **Exp** information can appear. **Def** means that the production plan is in non-expanded mode, and **Exp** means it is in expanded mode. The mode of the production plan is determined by the setting of the global option ***expanded version***. With Tivoli Workload Scheduler, Version 8.2, databases and plans are always expanded, but this information appears for backward compatibility with previous versions.

Examples

The following example displays the status of the current production plan.

```
%status
TWS for UNIX/CONMAN 8.4 (1.36.2.22)
Licensed Materials  Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998, 2007
US Government User Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
Job stream (Exp) 11/26/05 (#34) on site3.
Batchman LIVES. Limit:19, Fence:0, Audit Level:0
```

stop

Stops Tivoli Workload Scheduler production processes. To stop the **netman** process, use the **shutdown** command. You must have ***stop*** access to the workstation.

Syntax

stop [*domain***!**]*workstation*
[**;wait**]
[**;noask**]

Arguments

domain Specifies the name of the domain in which workstations are stopped. Because workstations have unique names, the domain is not needed when stopping a specific workstation. Wildcard characters are permitted.

This argument is useful when stopping more than one workstation in a domain. For example, to stop all the agents in domain **stlouis**, use the following command:

```
stop stlouis!@
```

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

workstation
Specifies the name of the workstation to be stopped. Wildcard characters are permitted.

This command is not supported on remote engine workstations.

wait Specifies not to accept another command until all processes have stopped.

noask Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

If the **stop** command cannot be applied to a distant workstation (for example, if the TCP/IP path is not available), the command is stored locally in a `pobox` file, and is sent to the workstation when it becomes linked.

Assuming the user has ***stop*** access to the workstations being stopped, the following rules apply:

- A user running **conman** on the master domain manager can stop any workstation in the network.

- A user running **conman** on a domain manager other than the master can stop any workstation in that domain and subordinate domains. The user cannot stop workstations in peer domains.
- A user running **conman** on an agent can stop any workstation in the local domain.

When you issue a **stop @** command on a domain manager, a local **conman stop** command runs on the remote CPUs. The command starts running on the lowest stations in the network hierarchy, then finally runs on the domain manager. However, the `Symphony` file is not updated before the CPUs go down. Therefore, if you issue a **conman sc@!@** command from any CPU, the resulting information might be an up to date picture of the states of the CPUs, even of the domain manager.

Examples

Figure 24 and Table 58 below show the workstations stopped by different **stop** commands run by users in different locations in the network.

DM*n* are domain managers and **A***nn* are agents.

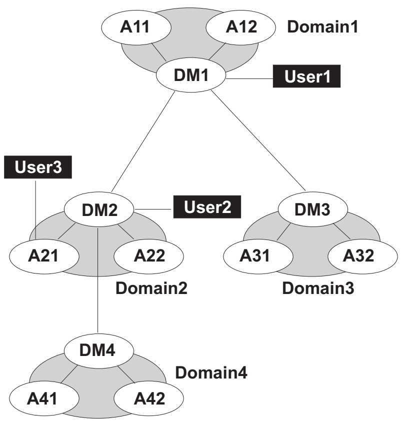


Figure 24. Example network

Table 58. Stopped workstations

Command	Stopped by: User1	Stopped by User2	Stopped by User3
stop @!@	All workstations are stopped	DM2 A21 A22 DM4 A41 A42	DM2 A21 A22
stop @	DM1 A11 A12	DM2 A21 A22	DM2 A21 A22
stop DOMAIN3!@	DM3 A31 A32	Not allowed	Not allowed

Table 58. Stopped workstations (continued)

Command	Stopped by: User1	Stopped by User2	Stopped by User3
stop DOMAIN4!@	DM4 A41 A42	DM4 A41 A42	Not allowed
stop DM2	DM2	DM2	DM2
stop A42	A42	A42	Not allowed
stop A31	A31	Not allowed	Not allowed

stop ;progressive

Stops Tivoli Workload Scheduler production processes hierarchically when you have defined at least one workstation as BEHINDFIREWALL in an Tivoli Workload Scheduler network. Similar to the stop @!@ command, but more effective in improving plan performance. The command does not run from the domain in which the command was initially issued for each subordinate domain, but runs at each hierarchical level.

Note: This command is not supported on remote engine workstations.

You must have ***stop*** access to the workstation.

Syntax

stop ;progressive

Comments

When you issue the command on a domain manager, all workstations in that domain are stopped and then the domain manager itself is stopped and the command continues to run on any subordinate domains. The command continues to run in this hierarchical manner, the domain manager stops workstations in the same domain, stops itself, and then continues to run on subordinate domains.

Examples

Figure 25 on page 393 and Table 59 on page 393 show the workstations stopped by issuing the **stop ;progressive** command on `DM2` and `DM4`.

DM*n* are domain managers and **A***nn* are agents.

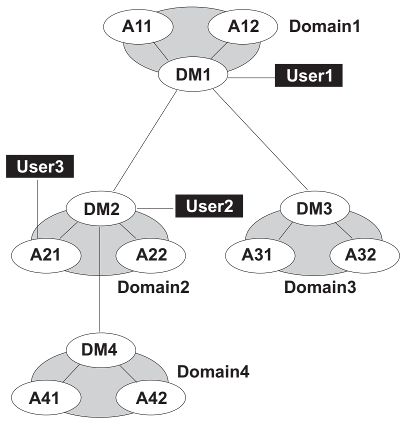


Figure 25. Example network

Table 59. Stopped workstations with stop ;progressive

Command	**Stopped by DM2**	**Stopped by DM4**
stop ;progressive	A21 A22 DM2	A41 A42 DM4

stopappserver

Stops the embedded WebSphere Application Server on the workstation.

Syntax

{**stopappserver** | **stopapps**} [*domain*!]*workstation*
[**;wait**]

Arguments

domain Specifies the name of the domain of the workstation. Because workstations have unique names, the domain is not needed when stopping the WebSphere Application Server on a specific workstation. Wildcard characters are permitted.

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

workstation
Specifies the name of the workstation where you want to stop the monitoring engine. Wildcard characters are permitted. If no domain and workstations are specified, the action is on the local workstation.

wait Waits until WebSphere Application Server has stopped before prompting for another command.

Comments

Permission to `stop` actions on `cpu` objects is required in the security file to be enabled to run this command.

On Windows systems refrain from using Windows services to stop WebSphere Application Server. If you use Windows services, the *appserverman* process, which continues to run, will start WebSphere Application Server again. Use this command or the **stopWas** command (without the **-direct** option) instead.

When you run the command, the *appserverman* process first checks if WebSphere Application Server can retrieve the user's credentials (username and password) from the `soap.client.props` file located in the WebSphere Application Server profile. If the check is negative, *appserverman* reads them from the `useropts` file of the user and runs the `stopServer.sh (bat)` script to pass them to WebSphere Application Server.

To be able to run the command, you must therefore complete one of the following two customization procedures to provide the user credentials to WebSphere Application Server:

- Customize the user name (`com.ibm.SOAP.loginUserid`) and password (`com.ibm.SOAP.loginPassword`) properties in the `soap.client.props` file located in:

```
TWS_home/appserver/profiles/twsprofile/properties/     (Version 8.4 and earlier master)
TWS_home/appserver/profiles/twsconnprofile/properties/ (Version 8.4 and earlier agents)
TWA_home/eWAS/profiles/TIPProfile/properties/          (Version 8.5 and up master and agents)
```

  You must also:
  1. Set property `com.ibm.SOAP.securityEnabled` to `true` in the same file to enable the SOAP client security
  2. Run the **encryptProfileProperties.sh** script to encrypt the password. See the Tivoli Workload Scheduler *Administration Guide* for more information on this application server tool.
- Customize the `Attributes for conman (CLI in version 8.4) connections` section in the `localopts` file by specifying the details of the connector or of the master domain manager.

  You must also:
  1. Create (or customize if already present) the `useropts` file manually, adding the `USERNAME` and `PASSWORD` attributes for the user who will run `stopappserver` . Make sure the `useropts` file name is entered in the `USEROPTS` key in the `Attributes for conman (CLI) connections` section. See the *Administration Guide* for further details.
  2. Encrypt the password in the `useropts` file simply by running `conman`.

stopeventprocessor

Stops the event processing server.

Syntax

{**stopeventprocessor** | **stopevtp**} [*domain*!][*workstation*]

Arguments

domain Specifies the name of the domain of the workstation.

workstation
Specifies the name of the master domain manager, backup master, or workstation installed as a backup master that functions as a plain fault-tolerant agent where you want to stop the event processing server. Wildcard characters are not permitted.

You can omit the workstation name if you run the command locally.

Comments

This command cannot be issued in an asynchronous way.

If you issue the command from a workstation other than the one where the event processor is configured, the command uses the command-line client, so the user credentials for the command-line client must be set correctly.

Permission to `stop` actions on `cpu` objects is required in the security file to be enabled to run this command.

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **More Actions→Stop Event Processor**.

stopmon

Stops the event monitoring engine on the workstation.

Syntax

{**stopmon** | **stopm**} [*domain***!**]*workstation*
  [**;wait**]
  [**;noask**]

Arguments

domain Specifies the name of the domain of the workstation. Because workstations have unique names, the domain is not needed when stopping the monitoring engine on a specific workstation. Wildcard characters are permitted.

If *domain* is omitted, and *workstation* contains wildcard characters, the default domain is the one in which **conman** is running.

workstation
Specifies the name of the workstation where you want to stop the monitoring engine. Wildcard characters are permitted.

wait Specifies not to accept another command until the monitoring engine has stopped.

noask Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

The monitoring engine is restarted automatically when the next production plan is activated (on Windows also when Tivoli Workload Scheduler is restarted) unless you disable the `autostart monman` local option.

The command is asynchronous, unless you specify the `wait` keyword.

Permission to `stop` actions on `cpu` objects is required in the security file to be enabled to run this command.

See also

In the Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **More Actions→Stop Event Monitoring**.

submit docommand

Submits a command to be launched as a job.

To run this command, in the security file you must have ***submit*** access for the job with the name specified in its database definition and, if you use the ***alias*** keyword, also with the name specified with this keyword. In addition, if you use the ***recoveryjob*** keyword, you must have ***submit*** access for the job specified with that keyword.

To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

If you submit the job from a workstation other than the master domain manager, you must be connecting as a user that:

- has proper credentials defined in the useropts file to connect to the master domain manager through WebSphere Application Server
- is authorized to perform submit commands in the security file stored on the master domain manager

Syntax

{**submit docommand** | **sbd**} [*workstation*#]**"***cmd***"**
[**;alias**[**=***name*]]
[**;into=**[*workstation***#**]
{*jobstream_id***;schedid** |*jobstreamname* (*hhmm*[*date*])}]
[**;***joboption*[**;**...]]

Arguments

workstation
Specifies the name of the workstation on which the job will be launched. Wildcard characters are permitted, in which case, the job is launched on all qualifying workstations. The default is the workstation on which **conman** is running. You cannot specify a domain or workstation class.

Note: Because of a limitation in the way Windows manages the equal (=) sign in the shell environment, you must mask the equal (=) sign as follows '\='\ when submitting Windows commands using **submit docommand**. For example, to set the local variable **var1** to *hello* you must issue the following command:

```
%sbd "set var1\"=\"hello"
```

cmd Specifies a valid system command of up to 255 characters. The entire command must be enclosed in quotes ("). The command is treated as a job, and all job rules apply.

alias=*name*
Specifies a unique name to be assigned to the job. If you enter the **alias** keyword without specifying a name, a name is constructed using up to the first six alphanumeric characters (in upper case) of the command, depending on the number of characters in the command, followed by a ten digit random number. If there are blanks in the command, the name is constructed using up to the first six alphanumeric characters before the blank. For example, if the command is **"rm apfile"**, the generated name will be similar to **RM0123456789**. If the command is longer than six alphanumeric characters such as, **"wlsinst"**, the generated name will be **wlsins0396578515**.

If you do not include **alias** the first time you submit the command, a job name is constructed using up to 255 characters of the command name. If you submit a command a second time from the same workstation, the **alias** keyword is mandatory and must be unique for each command submission.

into=*jobstream_instance*
Identifies the job stream instance into which the job will be placed for launching. Select the job stream instance as follows:

[*workstation***#**]*jobstreamname*[*hhmm*[*date*]]

or

[*workstation***#**]*jobstream_id* ;schedidIf **into** is not used, the job is added to a job stream named **JOBS**.

joboption
Specify any of the following:

at=*hhmm* [**timezone|tz** *tzname*] [**+***n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

confirmed

critical

deadline=*time* [**timezone|tz** *tzname*][**+***n* **day**[**s** | *mm/dd[/yy]*]

every=*rate*

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*[*hhmm* [*mm/dd[/yy]*]][.*job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**;nocheck**][**;wait**=*time*][**,**...]

Note: The **;nocheck** argument is not supported in internetwork dependencies.

interactive

Note: This keyword can be used in Windows environments only.

logon=*user*.

needs=[*num*] [*workstation***#**]*resource*[**,**...]

opens=[*workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]

priority[**=***pri* | **hi** | **go**]

prompt="[**:** | **!**]*text***"** | *promptname*[**,**...]

rccondsucc"*Success Condition*"

recovery=stop | **continue** | **rerun**

recoveryjob=*the name of a recovery job different from the one (if present) specified in the job definition in the database*

after [*workstation#*]*jobname*

abendprompt "*text*"

until *time* [**timezone**|**tz** *tzname*][+*n* **day**[**s**] | [**absolute** | **abs**]] [**;onuntil** *action*]

The default value for ***joboption*** is the user on the workstation from which the command is being run.

Using local parameters

You can use local parameters as values with the following keywords:

- `cmd`
- `opens`
- `logon`
- `prompt`
- `abendprompt`

Local parameters are defined and managed with the parms utility command in a local database on the workstation where the job is run. The parameters are resolved on the workstation while the **`submit`** command is in execution.

Comments

Jobs submitted in production from the **conman** command line are not included in the preproduction plan and so they cannot be taken into account when identifying external follows dependencies predecessors.

If you do not specify a *workstation* with `follows`, `needs`, `opens`, or `into`, the default is the workstation of the job.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *jobStreamName.workstationName.jobName* format.

When you submit the object into a job stream and add a follows dependency that shares the same job stream name (for example, you submit the object into job stream `schedA` and define a follows dependency on `schedA.job2`), the dependency is treated as an *external* follows dependency. Since Version 8.3, unlike in previous versions, because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

Examples

To submit an **rm** command into the job stream `JOBS` with a `follows` dependency, run the following command:

```
submit docommand="rm apfile";follows sked3
```

To submit a **sort** command with the alias **sortit** and place the job in the job stream `reports` with an **at** time of 5:30 p.m., run the following command:

```
sbd "sort < file1 > file2";alias=sortit;into=reports;at=1730
```

To submit **chmod** commands on all workstations with names beginning with `site`, run the following command:

```
sbd site@#"chmod 444 file2";alias
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Submit→Submit Ad Hoc Jobs**
2. Select an engine name and click **OK**
3. Fill in the requested data in the Submit Ad Hoc Job into Plan screens and click **OK**.

submit file

Submits a file to be launched as a job.

To run this command, in the security file you must have ***submit*** access for the job with the name specified in its database definition and, if you use the ***alias*** keyword, also with the name specified with this keyword. In addition, if you use the ***recoveryjob*** keyword, you must have ***submit*** access for the job specified with that keyword.

To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

If you submit the job from a workstation other than the master domain manager, you must be connecting as a user that:

- has proper credentials defined in the useropts file to connect to the master domain manager through WebSphere Application Server
- is authorized to perform submit commands in the security file stored on the master domain manager

Syntax

{**submit file** | **sbf**} **"***filename***"**
 [**;alias**[**=***name*]]
 [**;into=**[*workstation***#**]{*jobstream_id*
 ;schedid |*jobstreamname* (*hhmm*[*date*])}]
 [**;***joboption*[**;**...]]
 [**;noask**]

Arguments

filename
: Specifies the name of the file, up to 255 characters. Wildcard characters are permitted. The name must be enclosed in quotes (") if it contains characters other than alphanumeric characters, dashes (-), slashes (/), and underscores (_). See the examples.

alias=*name*
: Specifies a unique name to be assigned to the job. If you enter the **alias** keyword without specifying a name, a name is constructed using up to the first six alphanumeric characters (in upper case) of the file name, depending on the number of characters in the file name, followed by a ten

digit random number. For example, if the file name is `jclttx5`, the generated name will be similar to `JCLTTX0123456789`.

If you do not include **alias**, a filename is constructed using up to 255 alphanumeric characters of the file's base name, in upper case.

In either of the above cases, if the file name does not start with a letter, you are prompted to use **alias=** *name*.

If you submit a file a second time from the same workstation, the **alias** keyword is mandatory and must be unique for each file submission.

into=*jobstream_instance*
: Identifies the job stream instance into which the job will be placed for launching. Select the job stream instance as follows:

[*workstation***#**]*jobstreamname*[*hhmm*[*date*]]

or

[*workstation***#**]*jobstream_id* ;schedidIf **into** is not used, the job is added to a job stream named **JOBS**.

joboption
: Specify one of the following:

at=*hhmm* [**timezone|tz** *tzname*] [+*n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

confirmed

critical

deadline=*time*[**timezone** | **tz** *tzname*][+*n* **days** | *mm/dd[/yy]*]

every=*rate*

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*(*hhmm* [*mm/dd[/yy]*])[.*job* | **@**] | *jobstream_id.job***;schedid**}| *job*[**;nocheck**][**;wait**=*time*][**,**...]

Note: The **;nocheck** argument is not supported in internetwork dependencies.

interactive

Note: This keyword can be used in Windows environments only.

logon=*user*

needs=[*num*] [*workstation***#**]*resource*[**,**...]

opens=[*workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]

priority[=*pri* | **hi** | **go**]

prompt="[**:** | **!**]*text***"** | *promptname*[**,**...]

rccondsucc"*Success Condition*"

recovery=stop | continue | rerun

recoveryjob=*the name of a recovery job different from the one (if present) specified in the job definition in the database*

after [*workstation***#**]*jobname*

abendprompt "*text*"

until *time* [**timezone|tz** *tzname*][+*n* **day**[**s**] | [**absolute** | **abs**]] [**;onuntil** *action*]

noask Specifies not to prompt for confirmation before taking action against each qualifying file.

Using local parameters

You can use local parameters as values with the following keywords:

- `filename`
- `opens`
- `logon`
- `prompt`
- `abendprompt`

Local parameters are defined and managed with the parms utility command in a local database on the workstation where the job is run. The parameters are resolved on the workstation while the **`submit`** command is running.

Comments

Jobs submitted in production from the **conman** command line are not included in the preproduction plan and so they cannot be taken into account when identifying external follows dependencies predecessors.

If you do not specify a workstation with `follows`, `needs`, `opens`, or `into`, the default is the workstation on which **conman** is running.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *jobStreamName.workstationName.jobName* format.

When you submit the object into a job stream and add a follows dependency that shares the same job stream name (for example, you submit the object into job stream `schedA` and define a follows dependency on `schedA.job2`), the dependency is treated as an *external* follows dependency. Since Version 8.3, unlike in previous versions, because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

Examples

To submit a file into the job stream `jobs` (the job name is `myjcl`), run the following command:

```
submit file=d:\jobs\lib\daily\myjcl
```

where the `;into` sequence was omitted.

To submit a file, with a job name of `misjob4`, into the job stream `missked`, run the following command:

```
sbf /usr/lib/mis/jcl4;alias=misjob4;into=missked ;needs=2 slots
```

The job needs two units of the `slots` resource.

To submit all files that have names beginning with `back` into the job stream `bkup`, run the following command:

```
sbf "/usr/lib/backup/back@";into=bkup
```

To submit file `tws_env.cmd`, whose path contains a blank, on a Windows workstation run:

- In interactive mode:

  ```
  sbf "\"C:\Program Files\IBM\TWS\lucaMDM\tws_env.cmd\"";alias=MYJOB
  ```

  Being in Windows, the double quotes (") must be escaped by the "\ character sequence.
- In command line mode:

  ```
  conman sbf "\"\\\"C:\Program Files\IBM\TWS\lucaMDM\tws_env.cmd\\\"\"";alias=MYJOB
  ```

  Being in Windows, and running the command externally from the conman environment, the escape sequence becomes longer.

where "\" is the escape character for the blank in the file path.

submit job

Submits a job to be launched.

To run this command, in the security file you must have ***submit*** (***submitdb***) access for the job with the name specified in its database definition and, if you use the ***alias*** keyword, also with the name specified with this keyword. In addition, if you use the ***recoveryjob*** keyword, you must have ***submit*** access for the job specified with that keyword.

Note that if you have security ***submitdb*** rights only, you are limited to submit jobs defined in the database. You cannot submit ad-hoc jobs.

To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

If you submit the job from a workstation other than the master domain manager, you must be connecting as a user that:

- Has proper credentials defined in the useropts file to connect to the master domain manager through WebSphere Application Server
- Is authorized to perform **submit** commands in the security file stored on the master domain manager

If you submit a shadow job, see Chapter 19, "Defining and managing cross dependencies," on page 559 for more details.

Syntax

{**submit job** | **sbj**} [*workstation*#]*jobname*
 [**;alias**[=*name*]]
 [**;into=**[*workstation*#]{*jobstream_id*
 ;schedid | *jobstreamname*(*hhmm*[*date*])}]
 [**;***joboption*[**;**...]]
 [**;vartable=***tablename*]
 [**;noask**]

Arguments

workstation
: Specifies the name of the workstation on which the job will be launched. Wildcard characters are permitted, in which case, the job is launched on all

qualifying workstations. The default is the workstation on which `conman` is running. You cannot specify a domain or workstation class.

jobname
: Specifies the name of the job. Wildcard characters are permitted, in which case, all qualifying jobs are submitted. If the job is already in the production plan, and is being submitted into the same job stream, you must use the **alias** argument to assign a unique name.

alias=*name*
: Specifies a unique name to be assigned to the job in place of *jobname*. If you enter the **alias** keyword without specifying a name, a name is constructed using the first two alphanumeric characters of *jobname* followed by a six digit random number. The name is always upshifted. For example, if *jobname* is `jcrttx5`, the generated name will be similar to `JC123456`.

into=*jobstream_instance*
: Identifies the job stream instance into which the job will be placed for launching. Select the job stream instance as follows:

[*workstation***#**]*jobstreamname*[*hhmm*[*date*]]

or

[*workstation***#**]*jobstream_id* ;schedidIf **into** is not used, the job is added to a job stream named **JOBS**.

joboption
: Specify one of the following:

at=*hhmm* [**timezone|tz** *tzname*] [**+***n* **days** | *mm/dd[/yy]*] | [**absolute** | **abs**]

confirmed

critical

deadline=*time*[**timezone** | **tz** *tzname*][**+***n* **days** | *mm/dd[/yy]*]

every=*rate*

follows=[*netagent***::**][*workstation***#**]{*jobstreamname*(*hhmm* [*mm/dd[/yy]*]) [.*job* | **@**] | *jobstream_id*.*job***;schedid**}| *job*[**;nocheck**][**;wait**=*time*][**,**...]

Note: The **;nocheck** argument is not supported in internetwork dependencies.

needs=[*num*] [*workstation***#**]*resource*[**,**...]

opens=[*workstation***#**]"*filename*"[**(***qualifier***)**][**,**...]

priority[=*pri* | **hi** | **go**]

prompt="[**:** | **!**]*text***"** | *promptname*[**,**...]

rccondsucc"*Success Condition*"

recovery=stop | continue | rerun

recoveryjob=*the name of a recovery job different from the one (if present) specified in the job definition in the database*

after [*workstation***#**]*jobname*

abendprompt "*text*"

until *time* [**timezone|tz** *tzname*][+*n* **day**[**s**] | [**absolute** | **abs**]] [**;onuntil** *action*]

vartable=*tablename*
Specifies the name of the variable table, if different than the default one, where the variables you intend to use are defined.

Remember:

- With this command, you can use variable substitution for the following keywords:
  - `opens`
  - `prompt`
  - `abendprompt`
- Enclose the variable between carets (^), and then enclose the entire string between quotation marks. If the variable contains a portion of a path, ensure that the caret characters are not immediately preceded by a backslash (\) because, in that case, the \^ sequence could be wrongly interpreted as an escape sequence and resolved by the parser as caret character. If necessary, move the backslash into the definition of the variable between carets.

If you submit a job containing variables defined in a variable table that is not the default variable table and you do not specify the variable table in the run-cycle, job stream, or workstation, the variables are not resolved. See Chapter 6, "Customizing your workload using variable tables," on page 101.

noask Specifies not to prompt for confirmation before taking action against each qualifying job. This option can be used only with **;schedid**.

Comments

Jobs submitted in production from the **conman** command line are not included in the preproduction plan and so they cannot be taken into account when identifying external follows dependencies predecessors.

If you do not specify a workstation with `follows`, `needs`, `opens`, or `into`, the default is the workstation of the job.

at specifies at which time the job can be submitted. If the **at** keyword is used, then the job cannot start before the time set with this keyword. Note that if the master domain manager of your network runs with the `enLegacyStartOfDayEvaluation` and `enTimeZone` options set to `yes` to convert the `startOfDay` time set on the master domain manager to the local time zone set on each workstation across the network, you must add the **absolute** keyword to make it work.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *jobStreamName.workstationName.jobName* format.

When you submit the object into a job stream and add a follows dependency that shares the same job stream name (for example, you submit the object into job stream `schedA` and define a follows dependency on `schedA.job2`), the dependency is treated as an *external* follows dependency. Since Version 8.3, unlike in previous versions, because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

Examples

To submit the `test` jobs into the job stream `JOBS`, run the following command:

```
sbj test
```

To submit a job with an alias of `rptx4` and place the job in the job stream `reports` with an **at** time of 5:30 p.m., run the following command:

```
sbj rjob4;alias=rptx4;into=reports;at=1730
```

To submit job `txjob3` on all workstations whose names begin with `site`, run the following command:

```
sbj site@#txjob3;alias
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler**→**Workload**→**Submit**→**Submit Predefined Jobs**
2. Select an engine name and click **OK**
3. Fill in the requested data in the Submit Job into Plan screen and click **OK**.

submit sched

Submits a job stream for processing.

To run this command, in the security file you must have ***submit*** access for the job stream with the name specified in its database definition and, if you use the ***alias*** keyword, also with the name specified with this keyword. To include `needs` and `prompt` dependencies, you must have ***use*** access to the resources and global prompts.

The `submit schedule` command uses the credentials set in the `useropts` file belonging to the *TWS_user* who installed that workstation.

If you submit the job stream from a workstation other than the master domain manager, you must be connecting as a user that:

- has proper credentials defined in the useropts file to connect to the master domain manager through WebSphere Application Server
- is authorized to perform submit commands in the security file stored on the master domain manager

Syntax

{**submit sched** | **sbs**} [*workstation*#]*jstreamname*
    [**;alias**[=*name*]]
    [;*jstreamoption*[**;**...]]
    [**;vartable=***tablename*]
    [**;noask**]

Arguments

workstation
    Specifies the name of the workstation on which the job stream will be

launched. Wildcard characters are permitted, in which case, the job stream is launched on all qualifying workstations. The default is the workstation on which **conman** is running. You cannot specify a domain or workstation class.

jstreamname
: Specifies the name of the job stream. Wildcard characters are permitted, in which case, all qualifying job streams are submitted. If the job stream is already in the production plan, you must use the **alias** argument to assign a unique name.

alias=*name*
: Specifies a unique name to be assigned to the job stream in place of *jstreamname*. If set, this value corresponds also to the *jobstream_id*. If you enter the **alias** keyword without specifying a name, a name is constructed using the first alphanumeric characters of *jstreamname* followed by a six digit random number. The name is always upshifted. For example, if *jstreamname* is `sttrom`, the generated name will be similar to `ST123456`.

  The authorization to submit the schedule is checked in the Security file using the original name not the alias name.

jstreamoption
: Enter any of the following (refer to "Job stream definition keyword details" on page 188 to find which options are mutually exclusive):

  [at=*hhmm* **[timezone|tz** *tzname*] [+*n* **days** | *date*] **[absolute | abs]]** | **[schedtime=[***hhmm* [*date*] | [+*n* **days**]]
  : where:

    at specifies at which time the job stream can be submitted. If the at keyword is used, then the job stream cannot start before the time set with this keyword (see the topic on the job stream definition keywords in the chapter on "Defining objects in the database" in "*User's Guide and Reference*" for more information about the "at" keyword). Note that if the master domain manager of your network runs with the `enLegacyStartOfDayEvaluation` and `enTimeZone` options set to `yes` to convert the `startOfDay` time set on the master domain manager to the local time zone set on each workstation across the network, you must add the **absolute** keyword to make it work.

    schedtime represents the day and time when the job stream is positioned in the plan. If by this time the job stream is free from dependencies, and has no defined **at** time restrictions, it is launched. The value assigned to **schedtime** does not represent a dependency for the job stream. Its value is then displayed in the *SchedTime* columns in the output of the show commands. If an **at** restriction is defined, then the value assigned to **schedtime** is overwritten by the **at** value. When the job stream actually starts, the value assigned to **schedtime** is overwritten by the actual start time of the job stream.

    The format used for *date* depends on the value assigned to the ***date format*** variable specified in the `localopts` file.

    If no additional time zone is specified, the time zone set on the workstation running the command is assumed.

  carryforward

deadline=*time***[timezone | tz** *tzname***][+***n* **days** | *date***]**
If no additional time zone is specified, the time zone set on the workstation running the command is assumed.

follows=[*netagent***::][***workstation***#]{***jobstreamname***[***hhmm* **[***mm***/***dd[/yy]***]][.***job* **| @]** **|** *jobstream_id***.***job***;schedid}|** *job***[;nocheck][;wait=***time***][,...]**
The matching criteria used when submitting job streams in production is different from the way **follows** dependencies are resolved in the preproduction plan. When a job stream, for example `JS_A`, containing a **follows** dependency from a job or a job stream, for example `JS_B`, is submitted from the **conman** command line program, the predecessor instance of `JS_B` is defined following this criterion:

1. The closest instance of `JS_B` preceding `JS_A`.
2. If no preceding instance of `JS_B` exists, then the predecessor instance is the closest instance of `JS_B` following `JS_A`.
3. Otherwise an error is displayed and the command fails if the **;nocheck** keyword is not used.

The predecessor job stream instance is searched among the instances added to the production plan when **JnextPlan** was run and the instances submitted in production with the **sbs** command, including those submitted with an alias.

Attention: The **;nocheck** argument is not supported in internetwork dependencies.

limit=*joblimit*

needs=[*num***] [***workstation***#]***resource***[,...]**

opens=[*workstation***#]"***filename***"[(***qualifier***)][,...]**

priority[=*pri* **| hi | go]**

prompt="[: | !]*text***" |** *promptname***[,...]**

until *time* **[timezone|tz** *tzname***][+***n* **day[s] | [absolute | abs]] [;onuntil** *action***]**
If no additional time zone is specified, the time zone set on the workstation running the command is assumed.

vartable=*tablename*
Specifies the name of the variable table, if different than the default one, where the variables you intend to use are defined.

Remember:

- With this command, you can use variable substitution for the following keywords:
  - `opens`
  - `prompt`
- Enclose the variable between carets (^), and then enclose the entire string between quotation marks. If the variable contains a portion of a path, ensure that the caret characters are not immediately preceded by a backslash (\) because, in that case, the \^ sequence could be wrongly interpreted as an escape sequence and resolved by the parser as caret character. If necessary, move the backslash into the definition of the variable between carets.

If you submit a job stream with jobs containing variables defined in a variable table that is not the default variable table and you do not specify the variable table in the run-cycle, job stream, or workstation, the variables are not resolved. See Chapter 6, "Customizing your workload using variable tables," on page 101.

noask Specifies not to prompt for confirmation before taking action against each qualifying job stream. This option can be used only with **;schedid**.

Comments

Job streams submitted in production from the **conman** command line are not included in the preproduction plan and so they cannot be taken into account when identifying external follows dependencies predecessors.

If you do not specify a workstation with `follows`, `needs`, or `opens`, the default is the workstation of the job stream.

The scheduler classifies follows dependencies as *internal* when they are specified only by their job name within the job stream. It classifies them as *external* when they are specified in the *jobStreamName.workstationName.jobName* format.

When you submit a job stream that includes a job with a follows dependency that shares the same job stream name (for example, job stream `schedA` includes a job named `job6` that has a follows dependency on `schedA.job2`), the dependency is added as an *external* follows dependency. Since Version 8.3, unlike in previous versions, because the scheduler uses the `sameday` matching criteria to resolve external dependencies, dependencies originated in this way are never added the first time the object is submitted.

Examples

To submit the `adhoc` job stream on workstation `site1` and flags it as a **carryforward** job stream, run the following command:

```
submit sched=site1#adhoc;carryforward
```

To submit job stream `fox4` with a job limit of **2**, a priority of **23**, and an **until** time of midnight, run the following command:

```
sbs fox4;limit=2;pri=23;until=0000
```

To submit job stream `sched3` on all workstations with names that start with `site`, run the following command:

```
sbs site@#sched3
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Workload→Submit→Submit Predefined Job Streams**
2. Select an engine name
3. Fill in the requested data in the Submit Job Stream into Plan screen and click **OK**.

switcheventprocessor

Switches the event processing server from the master domain manager to the backup master or vice versa.

Note that you can run the event processing server also on a workstation installed as a backup master that runs as a plain fault-tolerant agent.

Syntax

{**switcheventprocessor** | **switchevtp**} *workstation*

Arguments

workstation
: Specifies the name of the master domain manager or of the backup master where you want to switch the event processing server. Wildcard characters are not permitted.

Comments

If you issue the command from a workstation other than the one where the event processor is configured, the command uses the command-line client, so the user credentials for the command-line client must be set correctly.

In case of backup masters the workstation must have the `full-status` attribute set to `on`.

Permission to `start` and `stop` actions on `cpu` objects is required in the security file to be enabled to run this command.

The correlation state of pending correlation rule instances is lost whenever the server is switched off or migrated. If caching of received events is enabled in the configuration file of the EIF listener, the cached events are lost after the event processor is switched.

Important:

- Before running this command, run **`planman deploy`** as a precaution. Do this to make sure that your latest changes or additions to active event rules are deployed before the event processor is switched and so avoid the risk that, because of a time mismatch, the latest updates (sent automatically based on the setup of the **`deploymentFrequency`** global option) are received by the old event processor instead of the new one.
- The master and backup masters designated to run the event processor should have their clocks synchronized at all times to avoid inconsistencies in the calculation of the time interval of running event rules. In fact, if the event processor is switched to a not-synchronized computer, timeout actions in the process of being triggered might undergo unexpected delays. Use a Network Time Protocol (NTP) server to keep all clocks synchronized.

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**

2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **More Actions→Become Event Processor**.

switchmgr

Switches domain management from the current domain manager to a backup domain manager.

You must have ***start*** and ***stop*** access to the backup domain manager.

The **switchmgr** command must only be used as part of specific procedures for switching domain management capabilities from a domain manager to its backup domain manager either permanently or temporarily. For information about these procedures, refer to the *IBMTivoli Workload Scheduler: Administration Guide*.

Syntax

{**switchmgr** | **switchm**} *domain*;*newmgr*

Arguments

domain Specifies the domain in which you want to switch managers.

newmgr
Specifies the name of the new domain manager. This must be a workstation in the same domain, and should be defined beforehand as a fault-tolerant agent with Resolve Dependencies and Full Status enabled.

Comments

The command stops a specified workstation and restarts it as the domain manager. All domain member workstations are informed of the switch, and the old domain manager is converted to a fault-tolerant agent in the domain.

The next time `JnextPlan` is run on the old domain manager, the domain acts as though another **switchmgr** command had been run and the old domain manager automatically resumes domain management responsibilities.

Fault-tolerant agents defined with `securitylevel = on` might fail to use the SSL port to connect to the new master domain manager after the **`switchmgr`** command is run. In this case do either of the following to let the agent start correctly:
- Unlink and then link the agent from the new master domain manager.
- Use the `securitylevel = force` option on the agent.

Examples

To switch the domain manager to workstation `orca` in the `masterdm` domain, run the following command:

```
switchmgr masterdm;orca
```

To switch the domain manager to workstation `ruby` in the `bldg2` domain, run the following command:

```
switchmgr bldg2;ruby
```

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **More Actions→Become Master Domain Manager**.

system command

Runs a system command.

Syntax

[**:** | **!**] *system-command*

Arguments

system-command
: Specifies any valid system command. The prefix (: or !) is required only when a command name has the same spelling as a **conman** command.

Examples

To run a **ps** command in UNIX, run the following command:

```
ps -ef
```

To run a **dir** command in Windows, run the following command:

```
dir \bin
```

tellop

Sends a message to the Tivoli Workload Scheduler console.

Syntax

{**tellop** | **to**} [*text*]

Arguments

text Specifies the text of the message. The message can contain up to 900 characters.

Comments

If **tellop** is issued on the master domain manager, the message is sent to all linked workstations. If issued on a domain manager, the message is sent to all of the linked agents in its domain and subordinate domains. If issued on a workstation other than a domain manager, the message is sent only to its domain manager if it is linked. The message is displayed only if the console message level is greater than zero. See "console" on page 324.

If **tellop** is entered alone, it prompts for the message text. At the prompt, type each line and press the Return key. At the end of the message, type two slashes (//) or a period (.), and press the Return key. You can use the new line sequence (\n) to format messages. Typing **Control+c** at any time will exit the **tellop** command without sending the message.

Examples

To send a message, run the following command:

```
tellop TWS will be stopped at\n4:30 for 15 minutes.
```

To prompt for text before sending a message, run the following command:

```
to
TELLOP>*********************************
TELLOP>*  TWS will be stopped at       *
TELLOP>*  4:30 for 15 minutes.         *
TELLOP>*********************************
TELLOP>//
```

unlink

Closes communication links between workstations.

You must have ***unlink*** access to the target workstation.

Syntax

unlink [*domain*!]*workstation*
    [**;noask**]

Arguments

domain Specifies the name of the domain in which to close links. It is not necessary to specify the domain name of a workstation in the master domain. Wildcard characters are permitted.

Note: You must always specify the domain name when unlinking a workstation not in the master domain.

This argument is useful when unlinking more than one workstation in a domain. For example, to unlink all the agents in domain `stlouis`, use the following command:

```
unlink stlouis!@
```

If you do not specify *domain*, and *workstation* includes wildcard characters, the default domain is the one in which **conman** is running.

workstation
Specifies the name of the workstation to be unlinked. Wildcard characters are permitted.

This command is not supported on remote engine workstations.

noask Specifies not to prompt for confirmation before taking action on each qualifying workstation.

Comments

Assuming that a user has **unlink** access to the workstations being unlinked, the following rules apply:

- A user running **conman** on the master domain manager can unlink any workstation in the network.

- A user running **conman** on a domain manager other than the master can unlink any workstation in its own domain and subordinate domains. The user cannot unlink workstations in peer domains.
- A user running **conman** on an agent can unlink any workstation in its local domain provided that the workstation is either a domain manager or host. A peer agent in the same domain cannot be unlinked.

For additional information see "link" on page 336.

Examples

Figure 26 and Table 60 show the links closed by **unlink** commands run by users in various locations in the network.

DM*n* are domain managers and **A***nn* are agents.

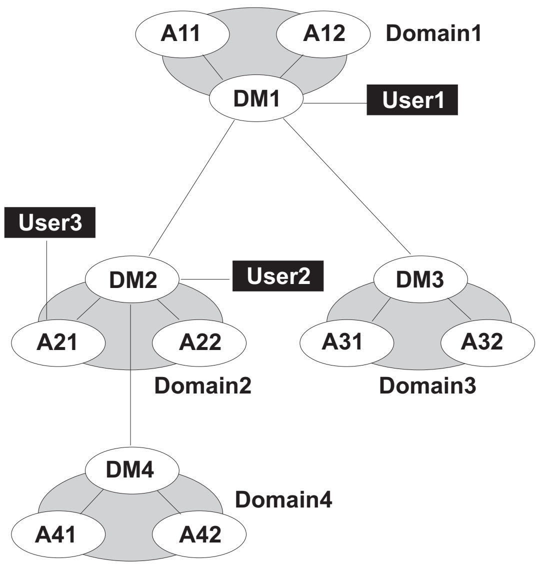


Figure 26. Unlinked network workstations

Table 60. Unlinked workstations

Command	Closed by User1	Closed by User2	Closed by User3
unlink@!@	All links are closed	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4 DM4-A41 DM4-A42	DM2-A21
unlink @	DM1-A11 DM1-A12 DM1-DM2 DM1-DM3	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4	DM2-A21
unlink DOMAIN3!@	DM3-A31 DM3-A32	Not allowed	Not allowed

Table 60. Unlinked workstations (continued)

Command	Closed by User1	Closed by User2	Closed by User3
unlink DOMAIN4!@	DM4-A41 DM4-A42	DM4-A41 DM4-A42	Not allowed
unlink DM2	DM1-DM2	Not applicable	DM2-A21
unlink A42	DM4-A42	DM4-A42	Not allowed
unlink A31	DM3-A31	Not allowed	Not allowed

See also

In the Tivoli Dynamic Workload Console:

1. Click **Tivoli Workload Scheduler→Scheduling Environment→Monitor→Monitor Workstations**
2. Select **All Workstations in plan** or another predefined task name
3. Choose an engine name, or specify connection properties, and click **OK**
4. Select a workstation and click **Unlink**.

version

Displays the **conman** program banner, inclusive of the version up to the installed fix pack level.

Syntax

{**version** | **v**}

Examples

To display the **conman** program banner, run the following command:

```
%version
```

The output is similar to this:

```
TWS for UNIX/CONMAN 8.4 (1.36.2.22)
Licensed Materials  Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998, 2007
US Government User Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
Job stream (Exp) 11/26/06 (#34) on site3.
Batchman LIVES.Limit:19,Fence:0,Audit Level:0
```

Chapter 11. Scheduling jobs on IBM i systems

When scheduling a job on IBM i systems, the job launches a native command that can be either a system or a user command. The native command consists of SBMJOB system command, which launches a batch job. The native command starts one or more batch jobs. The Agent Monitor can monitor a maximum of 130 batch jobs. The batch jobs can be monitored only if they are started by the native command and the maximum number of jobs that the IBM i agent can monitor is 130.

The agent joblog and TWSASPOOLS environment variable

By default, all information about the running of jobs is stored in the agent joblog. Most of this information usually consists of spool files. To select the spool file types that you want included in the agent joblog, use the **TWSASPOOLS** system variable, which works at IBM i agent level for any job to be submitted.

The **TWSASPOOLS** system variable forces the IBM i agent to either ignore all spool files or include one or more of them.

On the IBM i agent, create a new system level environment variable named TWSASPOOLS and set it to a list of the spool file types that are to be included. The list must begin with the **`SPOOLS`**: token.

For example, to force the IBM i agent to ignore all spool files, create the TWSASPOOLS variable as follows.

```
ADDENVVAR ENVVAR(TWSASPOOLS) VALUE(SPOOLS:) LEVEL(*SYS)
```

where the list after the SPOOL: token is empty. In this case, any agent joblog report for the IBM i agent is limited to the activity report that the Agent Monitor produces to trace its submission and monitoring action, and to the IBM i joblog of the Agent Monitor, which is always added at the end of the agent joblog.

To allow the IBM i agent to include only the QPRINT and the QPJOBLOG spool file types, that is, any spool files produced by **`printf`** instructions inside any ILE-C program and any produced joblog, create the TWSASPOOLS as follows:

```
ADDENVVAR ENVVAR(TWSASPOOLS) VALUE('SPOOLS: QPRINT QPJOBLOG') LEVEL(*SYS)
```

If the TWSASPOOLS variable already exists, change it as follows:

```
CHGENVVAR ENVVAR(TWSASPOOLS) VALUE('SPOOLS: QPRINT QPJOBLOG') LEVEL(*SYS)
```

If any VALUE parameter is set to an incorrect string, the IBM i agent ignores the TWSASPOOLS environment variable option. You can create and change the TWSASPOOLS environment variable while with the IBM i agent active, but no workload activity must be running.

The agent return code retrieval

The IBM i programming model was originally based on an early object orientation model in which programs communicated through message passing, rather than using return codes. The introduction of the Integrated Language Programming (ILE) model lead to the definitions of common areas to exchange data as return codes in the same job environment: the user return codes and the system end codes.

For information about user return codes, see “Controlling the job environment with the user return code.”

When the IBM i agent verifies that a submitted command or job is completed, it assigns a return code to the job based on the job status of the completed job. The return code is set depending on the completion message of the command or job. If the command or job completes successfully, the return code is set to 0. If the command or job does not complete successfully, the return code is set to the value of the severity of the message related to the exception that caused the abnormal end of the job. The IBM i agent can also set the return code to the value of the user return code when it is returned by the submitted command. If retrieved, the user return code is used as the value to set the return code.

The return code value assigned to the job is included in the IBM i agent joblog for the job and sent back to the scheduler user interface (WEB UI or z/OS ISPF panels) as return code, for compatibility reasons with agents on other operating systems.

Controlling the job environment with the user return code

With the introduction of the IBM i ILE model, it is possible to retrieve a value returned by a called program inside the same job.

When the Agent Monitor verifies that a submitted command is completed, it retrieves the following end of job codes using an IBM i System API:

End status code or <Status> (0 if successful)
It indicates if the system issued a controlled cancellation of the job. Possible values are:

1	the subsystem or the job itself is canceled.
0	the subsystem or the job itself is not canceled.
blank	the job is not running.

Program return code or <Prc> (0000 if successful)

It specifies the completion code of the last program (such as a data file utility program, or an RPG or COBOL program, invoked by the job).

If the job includes no program, the program return code is 0.

User return code or <Urc> (0000 if successful)

It specifies the user-defined return code set by ILE high-level language constructs. For example, the return code of a program written in C language.

It represents the most recent return code set by any thread within the job.

If the submitted command is a call to a user ILE program returning a value on exiting, this value is found in the Urc end of job code.

You can decide how to control the job environment of your submitted jobs by preparing the commands to be submitted as CALLs to your ILE programs, where the internal flow is controlled and the end status is decided through proper exit values. If a user program ends in error for an incorrect flow control, without returning a value, the Agent Monitor does not set the Return Code as user return code (Urc), but follows the criteria described in "The agent return code retrieval" on page 416.

The following example shows an ILE C user program where two batch jobs are launched and a value of 10 is returned to the caller, regardless of the completion status of the batch jobs.

```
===========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main(int argc, char *argv[])
{
   int   EnvVarRC=0;
   printf("issuing SBMJOB CMD(CALL MYLIB/DIVBY0)...\n");
   system("SBMJOB CMD(CALL MYLIB/DIVBY0)");
   printf("issuing SBMJOB CMD(WRKACTJOB OUTPUT(*PRINT))...\n");
   system("SBMJOB CMD(WRKACTJOB OUTPUT(*PRINT)) LOG(4 0 *SECLVL)");
   exit(10);
   return;
}
===========================================================================
```

Alternative method to set the user return code

In some IBM i environments, the system API retrieving the user return code (Urc) from the Agent Monitor code does not retrieve the correct value for Urc. It is therefore not recommended that you use any IBM i system APIs to retrieve the user return code. To receive a value returned by a called program, it is better to provide, instead, a parameter to receive the value.

Even if the Agent Monitor can retrieve the user return code using system API, an alternative user return code retrieval method was implemented in the Agent Monitor code. The alternative retrieval method has the following logic. The `USERRC` job environment variable is created and set to the *INI* value before submitting the user command. When the command ends, the Agent Monitor retrieves its user return code using the system APIs, but it also verifies if the `USERRC` job environment variable was updated at user program level. If a value different from *INI* is found, this is considered as the user return code and the value retrieved using the system APIs is ignored because the user program modified the value of `USERRC` job environment variable.

The change of the `USERRC` variable at user program level requires the `USERRC` value change before exiting from the application user code. In the ILE C case, you can do this using the **putenv** statement, where the user return code is set to be returned.

The following example shows how the user code returns the user return code using the IBM i agent reserved job environment variable `USERRC`. This code was obtained from the code of the example in "Controlling the job environment with the user return code" on page 416 by replacing the **exit** with the **putenv** statement.

```
==========================================================================
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main(int argc, char *argv[])
{
   int   EnvVarRC=0;
   printf("issuing SBMJOB CMD(CALL MYLIB/DIVBY0)...\n");
   system("SBMJOB CMD(CALL MYLIB/DIVBY0)");
   printf("issuing SBMJOB CMD(WRKACTJOB OUTPUT(*PRINT))...\n");
   system("SBMJOB CMD(WRKACTJOB OUTPUT(*PRINT)) LOG(4 0 *SECLVL)");
   EnvVarRC = putenv("USERRC=10");
   return;
}
==========================================================================
```

Chapter 12. Adding dynamic scheduling capabilities to your environment

This section explains how you can add dynamic scheduling capabilities to your environment to schedule both existing Tivoli Workload Scheduler jobs and job types with advanced options, both those supplied with the product and the additional types implemented through the custom plug-ins.

Dynamic capabilities help you maintain business policies and ensure service level agreements by:
- Automatically discovering scheduling environment resources
- Matching job requirements to available resources
- Controlling and optimizing use of resources
- Automatically following resource changes
- Requesting additional resources when needed

You can add dynamic capabilities to your environment by defining a set of workstation types:

Dynamic agent
A workstation that manages a wide variety of job types, for example, specific database or FTP jobs, in addition to existing job types. This workstation is automatically defined and registered in the Tivoli Workload Scheduler database when you install the dynamic agent. You can group dynamic agents in pools and dynamic pools.

Pool
A workstation that groups a set of dynamic agents with similar hardware or software characteristics to submit jobs to. Tivoli Workload Scheduler balances the jobs among the dynamic agents within the pool and automatically reassigns jobs to available dynamic agents if a dynamic agent is no longer available. To create a pool of dynamic agents in your Tivoli Workload Scheduler environment, define a workstation of type **pool** hosted by the dynamic workload broker workstation, then select the dynamic agents you want to add to the pool. You can define the pool using the Dynamic Workload Console or the **composer** command.

Dynamic pool
A workstation that groups a set of dynamic agents, which is dynamically defined based on the resource requirements you specify and hosted by the dynamic workload broker workstation. For example, if you require a workstation with low CPU usage and Windows installed to run your job, you specify these requirements using the Dynamic Workload Console or the **composer** command. When you save the set of requirements, a new workstation is automatically created in the Tivoli Workload Scheduler database. This workstation is hosted by the dynamic workload broker workstation. This workstation maps all the dynamic agents in your environment that meet the requirements you specified. The resulting pool is dynamically updated whenever a new suitable dynamic agent becomes available. Jobs run on the first workstation in the dynamic pool which marches all the requirements. Jobs scheduled on this workstation automatically inherit the requirements defined for the workstation.

For information on how to create pools and dynamic pools using the Dynamic Workload Console, see the section on creating a pool of agents in the *Tivoli Dynamic Workload Console User's Guide*. For more information on how to create pools and dynamic pools using the **composer** command, see the *User's Guide and Reference, SC32-1274*.

The dynamic agents, pools, and dynamic pools leverage the dynamic functionality built into Tivoli Workload Scheduler and provide the possibility at run time to dynamically associate your submitted workload (or part of it) to the best available resources. You can add dynamic scheduling capabilities to workstations at installation time. For more information on installing the dynamic agents, see the section on installing a new agent in the *Planning and Installation Guide*, *SC32-1273*.

You can use dynamic agents, pools and dynamic pools to schedule job types with advanced options. The job types with advanced options include both those supplied with the product and the additional types implemented through the custom plug-ins. Both job types run only on dynamic agents, pools, and dynamic pools. For more information on how to schedule job types with advanced options, see "Creating job types with advanced options" on page 422. For more information on how to create custom plug-ins, see *Extending Tivoli Workload Automation*, *SC14-7623*.

You can also use dynamic agents, pools, and dynamic pools to run the jobs you created for the existing Tivoli Workload Scheduler workstation types. To run these jobs on the dynamic workstation types, you only have to change the specification of the workstation where you want the job to run. For more information on how to schedule existing Tivoli Workload Scheduler jobs, see "Adding dynamic capabilities to existing Tivoli Workload Scheduler jobs" on page 429.

If you want to leverage the dynamic capability when scheduling job types with advanced options, you schedule them on pools and dynamic pools, which assign dynamically the job to the best available resource. If you are interested only in defining job types with advanced options, without using the dynamic scheduling capability, you schedule these jobs on a specific dynamic agent, on which the job runs statically.

Advantages of job types with advanced options

This section describes the advantages you can obtain implementing job types with advanced options, both those supplied with the product and the additional types implemented through the custom plug-ins, and scheduling them on dynamic agents, pools and dynamic pools.

While the standard Tivoli Workload Scheduler job is a generic script or command, you can define jobs to perform specific tasks, such as database, file transfer, Java, and web service operations, using the job types with advanced options available from the Dynamic Workload Console or the composer command. You can easily define these job types without having specific skills on the applications where the job runs.

The job types with advanced options include both those supplied with the product and the additional types implemented through the custom plug-ins

The following job types with advanced options are available

File transfer jobs
: Transfer files to and from a server reachable using FTP, SSH, or other protocols.

JCL Run a JCL job, either by reference or by definition. If you define a job by reference, you provide a reference the job you want to submit, without having to write or import the whole job into the JCL. If you define a job by definition, you provide a JCL definition to be submitted. This job type runs only on Tivoli Workload Scheduler distributed - Agent for z/OS.

Web services jobs
Call a web service.

Database jobs
Perform queries, SQL statements, and jobs on a number of databases, including custom databases. You can also create and run stored procedures on DB2, Oracle, and MSSQL databases.

Executable jobs
Run a script or command with advanced options, such as redirecting standard input and standard output to a file.

Java jobs
Run a Java class

MSSQL jobs
Run a Microsoft SQL job.

| **Access method jobs**
Extend job scheduling functions of Tivoli Workload Scheduler to other systems and applications using access methods. The access methods communicate with the external system to launch the job and return the status of the job. The following access methods are available:

- Oracle E-Business Suite
- PeopleSoft
- SAP
- MVS
- Custom methods

J2EE jobs
Allow Java applications in the same network to send and receive messages from and to a JMS destination.

IBM i jobs
Run a command on IBM i systems.

In addition to configuring job types with advanced options using the Dynamic Workload Console or the **composer** command, you can use the related configuration files. For more information, see the section about configuring to schedule job types with advanced options in the *Administration Guide*, *SC23-9113*.

For more information about the procedure for defining job types with advanced options, see the section about creating job types with advanced options in *Tivoli Dynamic Workload Console User's Guide*. For more information about each job type, see the Dynamic Workload Console online help. For information on how to create jobs using the **composer** command, see the section about job definition in the *User's Guide and Reference*, *SC32-1274*.

In addition, you can create custom plug-ins to implement your own job types with advanced options for applications not supported by Tivoli Workload Scheduler. For more information on how to create custom plug-ins, see *Extending Tivoli Workload Automation*, *SC14-7623*.

You run the job types with advanced options, both those supplied with the product and the additional types implemented through the custom plug-ins only on dynamic agents, pools, and dynamic pools.

Creating job types with advanced options

This section explains how to create a specific job type using the job types with advanced options provided with the Dynamic Workload Console.

You can easily define job types with advanced options, without having specific skills on the applications where the job runs. You can then schedule these job types only on dynamic agents, pools, and dynamic pools. The following procedure describes how to create a file transfer job using the Dynamic Workload Console. The procedure for creating the other job types is similar, but each job type contains job-specific options. For more information about each job type, see the Dynamic Workload Console online help.

To create a file transfer job using the Dynamic Workload Console, do the following:

1. Log on to the Dynamic Workload Console.
2. Expand Tivoli Workload Scheduler.
3. In the console navigation tree, expand **Workload** > **Design** and click **Create Workload Definitions**.
4. Specify an engine name. The **Workload Designer** is displayed.
5. In the Working List pane, select **New** > **Job Definition** > **File Transfer**. The properties of the job are displayed in the right-hand panel for editing.
6. In the properties panel, specify the attributes for the job definition you are creating.
7. Click **Save** to save the job definition in the database.

For more information about all the available job types and categories, see the following table:

Table 61. Job Types

Category	Job Type	Description	job types with advanced options
Native	Windows	Jobs that run on Windows operating systems.	NO
	UNIX	Jobs that run on UNIX platforms.	NO
	Other	Jobs that run on extended agents. Refer to *Tivoli Workload Scheduler for Applications User's Guide* for information about customized task types for supported vendor acquired applications.	NO
	Executable	Jobs that run script or command with advanced options, such as redirecting standard input and standard output to a file.	YES
	z/OS	Jobs that run the command that you specify in the JCL tab on a JCL system.	
	IBM i	Jobs that run a command on IBM i systems.	YES

Table 61. Job Types (continued)

Category	Job Type	Description	job types with advanced options
ERP	SAP Job on XA Workstations	Jobs that run on an SAP extended agent. This includes the three types of SAP R/3 job definitions: • Standard R/3 job • BW Process Chain job • BW InfoPackage job	NO
	SAP Job on Dynamic Workstations	Jobs that run on dynamic agent workstations, pools, dynamic pools, and z-centric agents. The following types of SAP job definition are available: • Standard R/3 job • BW Process Chain job • BW InfoPackage job	NO
	Access Method	Jobs extend job scheduling functions of Tivoli Workload Scheduler to other systems and applications using access methods. The access methods communicate with the external system to launch the job and return the status of the job. The following access methods are available: • Oracle E-Business Suite • PeopleSoft • SAP • MVS • Custom methods	NO
	PI Channel	Jobs that run SAP Process Integration (PI) Channel jobs to control communication channels between the Process Integrator and a backend SAP R/3 system. **Note:** Provided with Tivoli Workload Scheduler for Applications 8.6 or higher and available only if specifically installed.	YES
Cloud	Workload Broker	Jobs that manage the lifecycle of a dynamic workload broker job. *Tivoli Workload Scheduler Scheduling Workload Dynamically* for information about how to use dynamic workload broker.	NO
File Transfer and Coordination	File Transfer	Jobs that run program to transfer files to and from a server reachable using FTP, SSH, or other protocols.	YES
	Shadow Distributed	Jobs that run locally and map other jobs running in remote Tivoli Workload Scheduler distributed environment.	NO
	Shadow z/OS	Jobs that run locally and map other jobs running in remote Tivoli Workload Scheduler for z/OS environment.	NO
Database and Integrations	Database	Jobs that perform queries, SQL statements, and jobs on a number of databases, including custom databases. You can also create and run stored procedures on DB2, Oracle, and Microsoft SQL Server databases.	YES
	Web Services	Jobs that run a web service.	YES
	MS SQL	Jobs that run a Microsoft SQL Server job.	YES
	J2EE	Jobs that allow Java applications in the same network to send and receive messages from and to a JMS destination.	YES
	Java	Jobs that run a Java class.	YES
Business Analytics	Cognos Reports	Jobs that run IBM Cognos reports, interactive reports, query, and report views. **Note:** Provided with Tivoli Workload Scheduler for Applications 8.6 or higher and available only if specifically installed.	YES
	InfoSphere DataStage	Jobs that run IBM InfoSphere DataStage jobs. **Note:** Provided with Tivoli Workload Scheduler for Applications 8.6 or higher and available only if specifically installed.	YES

Return codes

The following is a list of the return codes for job types with advanced options

```
Database jobs:
RC =  0 -> Job completed successfully
RC = -1 -> SQL statement was run with an exit code different from 1
RC = -2 -> MSSQL Job error
RC = -3 -> SQL statement did not run because of an error in the statement
```

```
File transfer jobs:
RC =  0 -> The file transfer completed successfully

RC = -1 -> The file transfer is not performed. The job fails with the following
error code: AWKFTE007E

Explanation: An error occured during the file transfer operation

Possible reasons: Remote file not found or permission denied

RC = -2 -> The file transfer is not performed. The job fails with the following
error code: AWKFTE020E

Explanation: Only for SSH or Windows protocols. An error was returned
while attempting to convert the code page
```

```
Possible reasons: For SSH or Windows protocols, the code page is
automatically detected and converted. In this case, there is an error in the
code page of the file to be transferred, which is not compliant with the
code page of the local system

RC = -3 -> The file transfer is not performed.The job fails with the following
error code: AWKFTE015E

Explanation: An error occurred during the file transfer operation

Possible reasons: Local file is not found

RC = -4 -> The file transfer is performed with the default code page. The job
fails with the following error code: AWKFTE023E

Explanation: The specified codepage conversion has not been performed.
File transfer has been performed with default code pages

Possible reasons: The specified code page is not available
IBM i jobs:
Return code = user return code when retrieved
Return code = 0 -> job completed successfully
Return code > -1 -> job completed unsuccessfully
Java jobs:
RC =  0 -> Job completed successfully
RC = -1 -> The Java application launched by the job failed due to an exception
Web services jobs:
RC =  0 -> Job completed successfully
RC = -1 -> The server hostname contained in the Web Service URL is unknown
RC = -2 -> Web Service invocation error
```

When the user return code is retrieved, the IBM i Agent Monitor assigns a priority to it.

Defining variables and passwords for local resolution on dynamic agents

For job types with advanced options you have the possibility to let variables and passwords be defined and resolved locally on the dynamic agents (including pools and dynamic pools).

This is particularly useful in the case of passwords because you are not required to specify them in the job definition. The advantage is that, if the password has to change, you do not modify the job definition, but you change it with the param command locally on the agents (or on the pool agents) that run or may run the job. If the job is to be submitted to a pool or dynamic pool, you can copy the file with the variable definitions to all the agents participating in that pool, so that the variables are resolved locally wherever the job will run.

This feature is not restricted to Windows workstations alone. You can use it also on UNIX, as long as you apply it on job types with advanced options.

To define a variable or a password locally on a dynamic agent, use the param utility command. This command has the power to create, delete, and list local variables in dynamic agents. See the details on this command to learn how to use it.

Specifying local variables and passwords in the job definitions

After defining a variable and its value with the `param` command, to add it within a job definition so that it is resolved locally on the agent at runtime, use the following syntax:

```
${agent:variable_name}
```

After defining a password with the `param` command, to add it within a job definition so that it is resolved locally on the agent at runtime, use the following syntax:

```
${agent:password.user_name}
```

You can nest variables within passwords. If you used a variable to define a user, enter the password as follows within the job definition:

```
${agent:password.${agent:variable_name}}
```

where *variable_name* was previously defined as having value *user_name* with the `param` command.

Example

A Tivoli Workload Scheduler administrator needs to define a file transfer job that downloads a file from a remote server to one of the dynamic agents making up a pool of agents. The administrator wants to parametrize in the job definition:

- The name with which the remote file will be saved locally
- The remote user name and its password

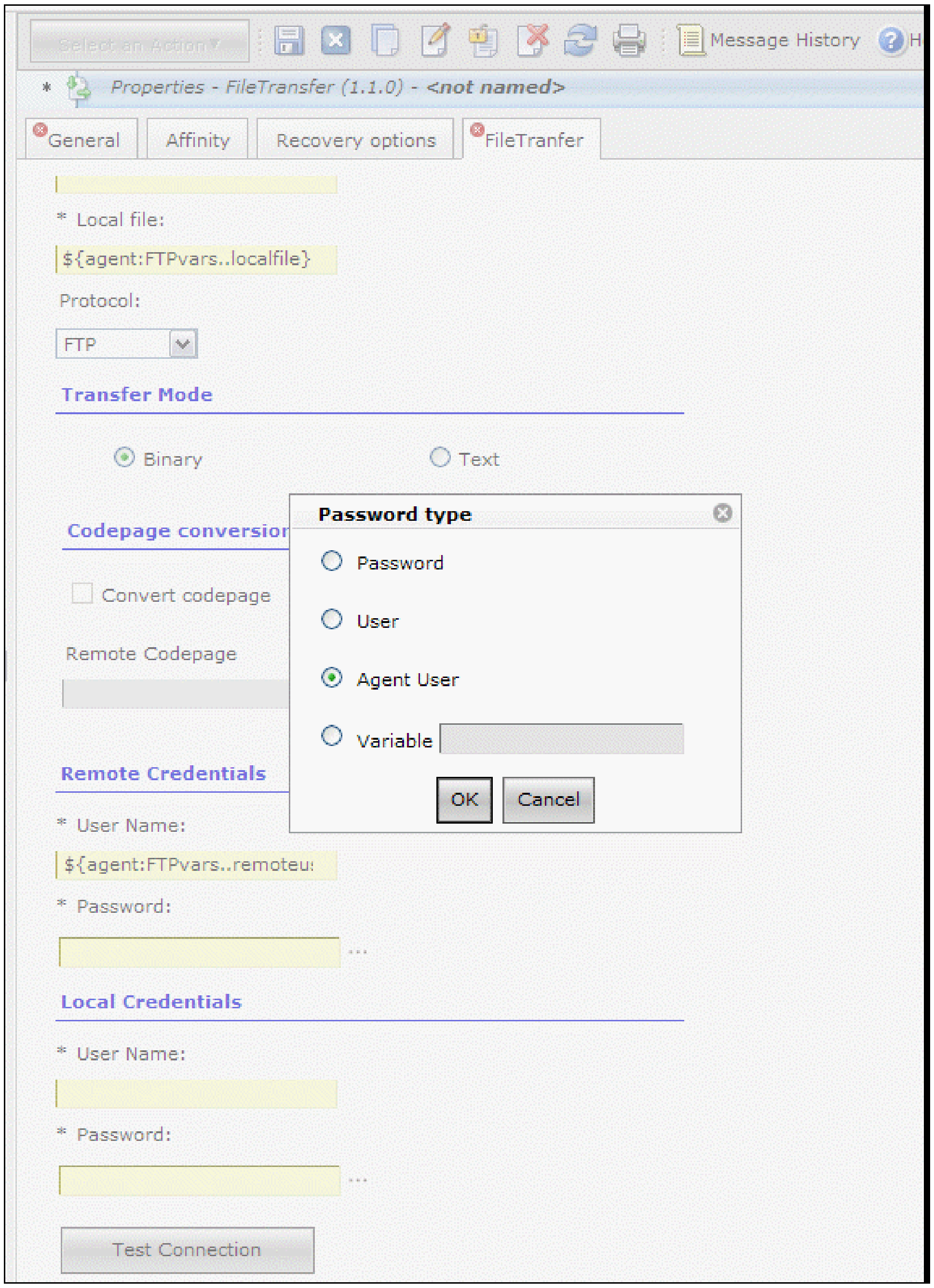
The administrator proceeds in the following way on one of the agents:

1. Defines a variable named `localfile`. The variable is given a value equal to `./npp.5.1.1.Installer.DW.exe` and is created in a new variables file named `FTPvars` (no section). The command to do this is:
   ```
   E:\IBM\TWA\TWS\CLI\bin>param -c FTPvars..localfile ./npp.5.1.1.Installer.DW.exe
   ```
2. Defines a variable named `remoteUser`. The variable is given a value equal to `FTPuser` and is created in the `FTPvars` file (no section). The command to do this is:
   ```
   E:\IBM\TWA\TWS\CLI\bin>param -c FTPvars..remoteUser FTPuser
   ```
3. Defines the password for `FTPuser`. The password value is `tdwb8nxt` and is created in the `password` section of the `FTPvars` file. The command to do this is:
   ```
   E:\IBM\TWA\TWS\CLI\bin>param -c FTPvars.password.FTPuser tdwb8nxt
   ```
4. With a text editor opens file `E:\IBM\TWA\TWS\ITA\cpa\config\jm_variables_files\FTPvars` and checks its contents:
   ```
   localfile = ./npp.5.1.1.Installer.DW.exe
   remoteuser = FTPuser

   [password]
   FTPuser = {aes}XMMYMY2zBHvDEDBo5DdZVmwOJao60pX1K6x2HhRcovA=
   ```
5. Copies file `FTPvars` in the *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files>` path of every other agent defined in the pool.
6. Starts defining the new file transfer job in the Workload - Design windows of Tivoli Dynamic Workload Console. In the `FileTransfer` window:
   a. Enters `${agent:FTPvars..localfile}` in the `Local file` field.

b. Enters `${agent:FTPvars..remoteuser}` in the `Remote Credentials →User Name` field.

c. Clicks the `...` button next to the `Remote Credentials →Password` field. The `Password type` window pops up and the administrator selects `Agent User`.



Properties - FileTransfer (1.1.0) - <not named>

Local file: ${agent:FTPvars..localfile}

Protocol: FTP

Transfer Mode: (●) Binary ( ) Text

Password type: ( ) Password ( ) User [✓ Agent User] ( ) Variable

Remote Credentials

User Name: ${agent:FTPvars..remoteu:

d. After the administrator clicks the `OK` button for confirmation in the popup window, the `Remote Credentials →Password` field is filled with the `${agent:password.${agent:FTPvars..remoteuser}}` value.

7. Fills in all the other fields to complete the job definition.

When the job is run, the entities and the password entered as variables are resolved with the values defined in the `FTPvars` file.

Using variables in dynamic workload broker jobs

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

You can include variables in your job definition. The variables are resolved at submission time.

The supported variables are as follows:

Table 62. 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.
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 "Promoting jobs scheduled on dynamic pools" on page 428.
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 "Promoting jobs scheduled on dynamic pools" on page 428.
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)

Table 62. Supported Tivoli Workload Scheduler variables in JSDL definitions. (continued)

Variables that can be inserted in the dynamic workload broker job definition	Description
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)

Defining affinity relationships

Affinity relationships cause jobs to run on the same workstation. The workstation on which the first job runs is chosen dynamically by dynamic workload broker, and the affine job or jobs run on the same workstation. This section applies to job types with advanced options and workload broker jobs.

In dynamic workload broker, you can define affinity relationships between two or more jobs when you want them to run on the same workstation. 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 using the Tivoli Workload Scheduler job name as follows:

```
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.

Note: The jobs must belong to the same job stream

Promoting jobs scheduled on dynamic pools

This section explains how to promote a critical job scheduled on a dynamic pool. A promoted job can run on a larger number of dynamic agents in the dynamic pool than a non-promoted job. This ensures that an important job runs before other jobs that are less important.

To ensure that a critical job obtains the necessary resources and is processed in a timely manner, you can promote it using promotion variables:

tws.job.promoted
: This environment variable indicates if the job is promoted. Supported values are **YES** and **NO**. The value of this variable applies to all jobs submitted in the specified environment.

tws.job.resourcesForPromoted
: This variable is defined in the dynamic pool definition and indicates the 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. The quantity is indicated with this notation: **${tws.job.resourcesForPromoted}**.

When a job is scheduled on the dynamic pool, the value of the **tws.job.promoted** variable in the job determines the behavior of the dynamic pool:

- If the value of the **tws.job.promoted** variable is **NO**, the value of the **tws.job.resourcesForPromoted** variable on the dynamic pool is 10, which means that few resources match the requirement.
- If the value of the **tws.job.promoted** variable is **YES**, the value of the **tws.job.resourcesForPromoted** variable on the dynamic pool is 1, which means that more resources match the requirement because the dynamic pool includes workstations with resource quantity equal to or greater than 1 and not only workstations with value equal or greater than 10.

For example, you can write a script that checks the value assigned to the **tws.job.promoted** variable in the job and performs different actions based on whether or not the job is promoted.

Adding dynamic capabilities to existing Tivoli Workload Scheduler jobs

This section explains how to modify an existing job to use the dynamic capabilities provided with dynamic agents, pools, and dynamic pools.

You can modify your existing Tivoli Workload Scheduler jobs to use the dynamic capabilities provided with dynamic agents, pools, and dynamic pools. To modify an existing job, do the following:

1. Install the required number of dynamic agents.
2. Optionally, assign the dynamic agents to pools or create dynamic pools based on your requirements.
3. Analyze your existing Tivoli Workload Scheduler jobs and decide which ones would obtain the best results when using the dynamic capability.
4. Log in to the Dynamic Workload Console.
5. Expand Tivoli Workload Scheduler.
6. In the console navigation tree, expand **Workload > Design** and click **Create Workload Definitions**.
7. Specify an engine name. The **Workload Designer** is displayed.
8. In the Working List panel, select **Search > Job Definition**. The search panel is displayed.
9. Enter your search criteria and click **Search**.
10. Select one or more jobs among the search results and click **Edit**. The selected jobs are displayed in the right-hand panel for editing.
11. In the **General** tab, click on the browse button of the **Workstation** field. The search panel is displayed.
12. Enter your search criteria and click **Search**.
13. Select the appropriate dynamic agent, pool, or dynamic pool and click **OK**. The job is now assigned to the specified workstation and will run on it when scheduled.

A business scenario on dynamic capability

This section demonstrates a sample business scenario which outlines the advantages of job types with advanced options and dynamic capability.

An insurance company runs a number of jobs at night to save the data processed during the day in the backup database. They also need to gather all data about the transactions completed during the day from all the workstations in the company branches. They use DB2 databases. Using the job types with advanced options provided in the Workload Designer, they create a job to perform a DB backup and another job to extract the data for the daily transactions. To perform these operations, they use the new database job type with advanced options.

After gathering data from all the company workstations, they copy the resulting data on a single workstation and process it to generate a report. They choose dynamically the best available workstation by defining the requirements necessary to run the job: a workstation with large disk space, powerful CPU and the program required to generate the report.

If the administrator does not want to modify the job stream he used before Tivoli Workload Scheduler. version 8.6 to run a Java job, for example, he can modify the name of the workstation where he wants the job to run, inserting the name of a pool or dynamic pool of dynamic agents where the Java executable is installed. Tivoli Workload Scheduler translates the syntax of the job so that it can be run by the Java program and assigns the job to the best available resource in the pool.

The report highlights how many new contracts were signed and how many customers are late with their payments. A mail is sent to the chief accountant, listing the number of new contracts and late customers.

The company can reach this objective by:
- Using the new workstations with dynamic capabilities to run the jobs the administrator created for the existing Tivoli Workload Scheduler workstations. To run these jobs on the new workstations, the administrator changes only the workstation where he wants the job to run. The major advantage is that he can use the workflows he previously created without additional effort.
- Defining several job types with advanced options without having specific skills on the applications where the job runs.

These job types with advanced options run on the following workstations:

dynamic agents
Workstations capable of running both existing jobs and job types with advanced options.

Pools Groups to which you can add dynamic agents depending on your needs. Jobs are assigned dynamically to the best available agent.

Dynamic pools
Groups of dynamic agents for which you specify your requirements and let Tivoli Workload Scheduler select the dynamic agents which meet your needs. Jobs are assigned dynamically to the best available dynamic agent.

Scenario: Creating a job definition and submitting to a dynamic pool

In this scenario, you define the requirements for running the job when creating the dynamic pool, for example you can include in the dynamic pool all workstations with Windows operating system and DB2 installed and maximum CPU utilization at 50%. The dynamic pool is then populated with the workstations which match your requirements and is ready for the job to be submitted.

To create a dynamic pool and submit a job to it, perform the following steps:

1. Log in to the Dynamic Workload Console and select **Tivoli Workload Scheduler > Scheduling Environment > Design > Create Workstations**.
2. Select an engine and click **OK**.
3. The **Workstation Properties** page is displayed.
4. Select **Dynamic Pool** in the **Workstation type** menu.
5. Fill in the required fields.
6. Click **Edit Requirements**. The **Requirements** page is displayed.
7. Specify the following requirements:
   - Select the operating system in the **Operating System** pane.
   - Select the CPU utilization in the **CPU utilization** pane.
   - Select the required logical resource in the **Logical Resources** pane. To include workstations with DB2 installed, click **Add** and specify your requirement in the **Requirements** pane.
   - Optionally, select the required optimization policy.
8. Click **OK** to save your requirements.
9. Click **Save** to save the dynamic pool.
10. From the portfolio, select **Tivoli Workload Scheduler > Workload > Design > Create Workload Definitions**.
11. Select an engine and click **OK**. The **Workload Designer** opens.
12. Select **New > Job Definition > Database and Integrations > Database**.
13. Fill in the required fields and specify the dynamic pool you previously created in the **Workstation** field.
14. Type your SQL instructions in the **SQL** tab.
15. Click **Save** to save the job.
16. From the portfolio, expand **Workload > Submit**, and click **Submit Predefined Jobs**.

Scenario: Creating a job definition and submitting to a pool

In this scenario, you need to run an inventory update script, therefore you create a pool, grouping workstations where the required program is installed, and a job to run the script. You select the target system for the job from the `invadmin` pool.

To create a pool and submit a job to it, perform the following steps:
1. Log in to the Dynamic Workload Console and select **Tivoli Workload Scheduler > Scheduling Environment > Design > Create Workstations**.
2. Select an engine and click **OK**.
3. The **Workstation Properties** page is displayed.
4. Select **Pool** in the **Workstation type** menu.
5. Name the pool `invadmin` and fill in the required fields.
6. Select the workstations you want to add to the pool. Select the workstations where the required program is installed.
7. Click **Save** to save the pool.
8. From the portfolio, select **Tivoli Workload Scheduler > Workload > Design > Create Workload Definitions**.
9. Select an engine and click **OK**. The **Workload Designer** is displayed.
10. Select **New > Job Definition > Native > Executable**.
11. Specify a name for the job.

12. In the **Workstation** field, specify the `invadmin` pool you previously defined.
13. Browse to the **Task** tab and select **Command**.
14. Type the name and path to the executable file you want to run, located on each workstation in the pool.
15. Click **Save** to save the job.
16. From the portfolio, expand **Workload > Submit**, and click **Submit Predefined Jobs**.

Chapter 13. Using utility commands

This chapter describes Tivoli Workload Scheduler utility commands. These commands, with the exceptions listed below, are installed in the `TWS_home/bin` directory. You run utility commands from the operating system command prompt.

The **StartUp** is installed in the *TWS_home* directory and **version** is installed in the `TWS_home/version` directory.

Command descriptions

Table 63 contains the list of the utility commands, and for each command, its description and the operating systems it supports.

Table 63. List of utility commands

Command	Description	Operating system
at	Submits a job to be run at a specific time.	UNIX
batch	Submits a job to be run as soon as possible.	UNIX
cpuinfo	Returns information from a workstation definition.	UNIX, Windows
datecalc	Converts date and time to a desired format	UNIX, Windows
delete	Removes script files and standard list files by name.	UNIX, Windows
evtdef	Imports/exports custom events definitions.	UNIX, Windows
evtsize	Defines the maximum size of event message files.	UNIX, Windows
exportservedata	Downloads the list of dynamic workload broker instances from the Tivoli Workload Scheduler database and changes a port number or a host name.	UNIX, Windows
importservedata	Uploads the list of dynamic workload broker instances to the Tivoli Workload Scheduler database after editing the temporary file to change a port number or a host name.	UNIX, Windows
jobinfo	Returns information about a job.	UNIX, Windows
jobstdl	Returns the pathnames of standard list files.	UNIX, Windows
listproc	Lists processes. This command is not supported.	Windows
killproc	Kills processes. This command is not supported.	Windows
maestro	Returns the Tivoli Workload Scheduler home directory.	UNIX, Windows
makecal	Creates custom calendars.	UNIX, Windows

Table 63. List of utility commands (continued)

Command	Description	Operating system
metronome.pl	Is replaced by **tws_inst_pull_info**.	UNIX, Windows
morestdl	Displays the contents of standard list files.	UNIX, Windows
movehistorydata	Moves the data present in the Tivoli Workload Scheduler database to the archive tables	UNIX, Windows
param	Creates, displays, and deletes variables and user passwords on dynamic agents.	UNIX, Windows
parms	Displays, changes, and adds parameters.	UNIX, Windows
release	Releases units of a resource.	UNIX, Windows
rmstdlist	Removes standard list files based on age.	UNIX, Windows
sendevent	Sends generic events to the currently active event processor server.	UNIX, Windows
showexec	Displays information about executing jobs.	UNIX
shutdown	Stops the **netman** process and, optionally, WebSphere Application Server.	UNIX, Windows
ShutDownLwa	Stops the dynamic agent locally	UNIX, Windows **Note:** On UNIX systems, it can be run by *TWS_user* or root user only.
StartUp	Starts the **netman** process and, optionally, WebSphere Application Server.	UNIX, Windows
StartUpLwa	Starts the dynamic agent locally	UNIX, Windows **Note:** On UNIX systems, it can be run by *TWS_user* or root user only.
tws_inst_pull_info	Collects data on the local Tivoli Workload Scheduler instance and workstation, WebSphere Application Server, and DB2 for diagnostic purposes. It is documented in *Tivoli Workload Scheduler: Troubleshooting Guide*.	UNIX, Windows
version	Displays version information.	UNIX

at and batch

Submit ad hoc commands and jobs to be launched by Tivoli Workload Scheduler.

These command runs on UNIX only.

See **at.allow** and **at.deny** below for information about the availability to users.

Syntax

at -V | -U

at {**-s** *jstream* | **-q** *queue*} *time-spec*

batch -V | -U

batch [**-s** *jstream*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-s *jstream*

Specifies the *jobstream_id* of the job stream instance into which the job is submitted. If a job stream instance with that *jobstream_id* does not exist, it is created a new job stream having *jstream* both as alias and as *jobstream_id*. The name must start with a letter, and can contain alphanumeric characters and dashes. It can contain up to 16 characters.

If the **-s** and **-q** arguments are omitted, a job stream name is selected based on the value of the environment variable ***ATSCRIPT***. If ***ATSCRIPT*** contains the word **maestro**, the job stream alias will be the first eight characters of the user's group name. If ***ATSCRIPT*** is not set, or is set to a value other than **maestro**, the job stream alias will be **at** (for jobs submitted with **at**), or **batch** (for jobs submitted with **batch**).

See "Other considerations" on page 437 for more information about job streams.

The following keywords apply only to **at** jobs:

-q*queue*

Specifies to submit the job into a job stream with the name *queue*, which can be a single letter (a through z). See "Other considerations" on page 437 for more information about job streams.

time-spec

Specifies the time at which the job will be launched. The syntax is the same as that used with the UNIX **at** command.

Comments

After entering **at** or **batch**, enter the commands that constitute the job. End each line of input by pressing the Return key. The entire sequence is ended with end-of-file (usually **Control+d**), or by entering a line with a period (.). Alternatively, use an angle bracket (<) to read commands from a file. See "Examples" on page 436.

Information about **at** and **batch** jobs is sent to the master domain manager, where the jobs are added to job streams in the production plan, `Symphony` file. The jobs are launched based on the dependencies included in the job streams.

The UNIX shell used for jobs submitted with the **at** and **batch** commands is determined by the ***SHELL_TYPE*** variable in the `jobmanrc` configuration script. Do not use the C shell. For more information, see "Customizing job processing on a UNIX workstation - jobmanrc" on page 39.

Once submitted, jobs are launched in the same way as other scheduled jobs. Each job runs in the submitting user's environment. To ensure that the environment is complete, **set** commands are inserted into the script to match the variable settings in the user's environment.

Examples

To submit a job into job stream with *jobstream_id* `sched8` to be launched as soon as possible, run the following command:

```
batch -s sched8
command <Return>
...
<Control d>
```

To submit a job to be launched two hours from the time when the command was entered, run the following command:

```
at now + 2 hours
command <Return>
...
<Control d>
```

If the variable ***ATSCRIPT*** is null, the job is submitted into a job stream having the same name as the user's group. Otherwise, it is submitted into a job stream named `at`.

To submit a job into a job stream instance with *jobstream_id* `sked-mis` to be launched at 5:30 p.m., run the following command:

```
at -s sked-mis 17h30
command <Return>
...
<Control d>
```

The following command is the same as the previous command, except that the job's commands are read from a file:

```
at -s sked-mis 17h30 < ./myjob
```

The fact that the commands are read from a file does not change the way they are processed. That is, the commands are copied from the `./myjob` file into a script file.

Replacing the UNIX commands

The standard UNIX **at** and **batch** commands can be replaced by Tivoli Workload Scheduler commands. The following commands show how to replace the UNIX **at** and **batch** commands:

```
$ mv /usr/bin/at /usr/bin/uat
$ mv /usr/bin/batch /usr/bin/ubatch
$ ln -s TWShome/bin/at /usr/bin/at
$ ln -s TWShome/bin/batch /usr/bin/batch
```

The at.allow and at.deny files

The **at** and **batch** commands use the files `/usr/lib/cron/`**`at.allow`** and `/usr/lib/cron/`**`at.deny`** to restrict usage. If the `at.allow` file exists, only users listed in the file are allowed to use **at** and **batch**. If the file does not exist, `at.deny` is checked to see if the user is explicitly denied permission. If neither of the files exists, only the **root** user is permitted to use the commands.

Script files

The commands entered with **at** or **batch** are stored in script files. The file are created by Tivoli Workload Scheduler using the following naming convention:

`TWS_home``/atjobs/`***`epoch.sss`***

where:

epoch The number of seconds since 00:00, 1/1/70.

sss The first three characters of the job stream name.

Note: Tivoli Workload Scheduler removes script files for jobs that are not carried forward. However, you should monitor the disk space in the `atjobs` directory and remove older files if necessary.

Job names

All **at** and **batch** jobs are given unique names by Tivoli Workload Scheduler when they are submitted. The names consist of the user's process ID (PID) preceded by the user's name truncated so as not to exceed eight characters. The resulting name is upshifted.

Other considerations

- The job streams into which **at** and **batch** jobs are submitted should be created beforehand with composer. The job streams can contain dependencies that determine when the jobs will be launched. At a minimum, the job streams should contain the **carryforward** keyword. This ensures that jobs that do not complete, or are not launched, while the current production plan is in process are carried forward to the next production plan.
- Include the expression **on everyday** to have the job streams selected every day.
- Use the **limit** keyword to limit the number of submitted jobs that can be run concurrently.
- Use the **priority** keyword to set the priority of submitted jobs relative to other jobs.

If the time value is less than the current time, the value is regarded as for the following day. If the time value is greater than the current time, the value is regarded as for the current day.

cpuinfo

Returns information from a workstation definition.

Syntax

cpuinfo -V | -U

cpuinfo *workstation* [*infotype*] [...]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

workstation

The name of the workstation.

infotype

The type of information to display. Specify one or more of the following:

os_type
Returns the value of the **os** field: **UNIX**, **WNT**, **ZOS**, **OTHER**, and **IBM i**. The value **ZOS** applies only to remote engine workstations used to communicate to a Tivoli Workload Scheduler for z/OS controller.

node Returns the value of the **node** field. For a workload broker server it is the host name or the TCP/IP address of the workstation where you installed the Tivoli Workload Scheduler Bridge. For a remote engine workstation it is the host name of workstation where the remote engine is installed. In any other case specify the host name or the TCP/IP address of the workstation.

port Returns the value of the **tcpaddr** field. If you are defining a workload broker workstation, specify the value of the **TWS.Agent.Port** property of the TWSAgentConfig.properties file. For remote engine workstations the value of this field is the HTTP port number used by the remote engine. If HTTPS protocol is used the value of this field is 31111.

sslport
Returns the value of the **secureaddr** field. It is the port used to listen for incoming SSL connections. For remote engine workstations the value of this field is the HTTPS port number used by the remote engine. If HTTP protocol is used the value of this field is 31113.

engineaddr
For any type of workstations the value of this field is 0.

protocol
Returns the value of the **protocol** field: HTTP or HTTPS. When the type of workstation is remote engine this value indicates the protocol used to communicate between the broker server and the remote engine.

sec_level
Returns the value of the **securitylevel** field: ENABLED, ON or FORCE.

autolink
Returns the value of the **autolink** field: ON or OFF.

fullstatus
Returns the value of the **fullstatus** field: ON or OFF.

resolvedep
Returns ON or OFF. No longer used in version 8.6.

behindfirewall
Returns the value of the **behindfirewall** field: ON or OFF.

host Returns the value of the **host** field. It is the name of the workstation hosting the agent.

domain
Returns the value of the **domain** field.

ID Returns the agent identifier used by the workstation when connecting to the broker server. For workstation with type: AGENT, REM-ENG, POOL, D-POOL.

method
For extended and network agents only. Returns the value of the **access** field.

server Returns the value of the **server** field.

type Returns the value of **type** field. It shows the type of workstation: MASTER, MANAGER, FTA, S-AGENT, REM-ENG, AGENT, POOL, D-POOL and X-AGENT.

time_zone
Returns the value of **timezone** field. It shows the time zone of the workstation. For an extended agent, the field is blank. For a remote engine workstation, this is the time zone of the remote engine.

version
Returns the Tivoli Workload Scheduler version that is running on the workstation. For an extended agent, the field is blank.

info Returns the operating system version and workstation model. For extended agents the field is blank. For remote engine workstations this field displays Remote Engine.

Comments

The values are returned, one on each line, in the same order that the arguments were entered on the command line. If no arguments are specified, all applicable information is returned with labels, one on each line.

Examples

The examples below are based on the following workstation definition:

```
Workstation Name  Type     Domain            Updated On  Locked By
----------------  -------  ----------------  ----------  ------------------
RE-ZOS            REM-ENG  -                 09/06/2010  -

CPUNAME RE-ZOS
  OS ZOS
  NODE 9.168.119.189 TCPADDR 635
  FOR MAESTRO HOST NC123162_DWB
    TYPE REM-ENG
    PROTOCOL HTTP
END
```

To print the **type** and **protocol** for workstation RE-ZOS, run the following command:

```
>cpuinfo RE-ZOS type protocol
REM-ENG
HTTP
```

To print all information for workstation RE-ZOS, run the following command:

```
>cpuinfo RE-ZOS
 OS_TYPE: ZOS
 NODE: 9.168.119.189
 PORT: 635
 SSLPORT: 31113
 ENGINEADDR: 0
 PROTOCOL: HTTP
 AUTOLINK: OFF
 FULLSTATUS: OFF
 RESOLVEDEP: OFF
 BEHINDFIREWALL: OFF
 HOST: NC123162_DWB
 DOMAIN: MASTERDM
 ID: D795263CBCD2365CA7B5C5BC0C3DD363
 SERVER:
 TYPE: REM-ENG
 TIME_ZONE: Europe/Rome
 VERSION: 8.6
 INFO: Remote Engine
```

datecalc

Resolves date expressions and returns dates in the format you choose.

Syntax

datecalc -V | **-U**

datecalc *base-date*
  [*offset*]
  [**pic** *format*]
  [**freedays** *Calendar_Name* [**-sa**] [**-su**]]

datecalc -t *time*
  [*base-date*]
  [*offset*]
  [**pic** *format*]

datecalc *yyyymmddhhtt*
  [*offset*]
  [**pic** *format*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

base-date

Specify one of the following:

day | *date* | **today** | **tomorrow** | **scheddate**

where:

day Specifies a day of the week. Valid values are: **su**, **mo**, **tu**, **we**, **th**, **fr**, or **sa**.

date Specifies a date, in the format *element*/*element*[/*element*], where *element* is: *d*[*d*], *m*[*m*], and *yy*[*yy*]. Any different format of date is not valid.

If two digits are used for the year (*yy*), a number greater than 70 is a 20th century date, and a number less than 70 is a 21st century date.

The parameter refers to the actual date, not to the UNIX *date* command. The following example shows an option to use the output of the UNIX *date* as input for Tivoli Workload Scheduler *date* parameter.

```
hdate=´date +"%m/%d/%y"´
echo $hdate
datecalc $hdate pic mm/dd/yyyy
```

Valid values for the month (*m*[*m*]) are **jan**, **feb**, **mar**, **apr**, **may**, **jun**, **jul**, **aug**, **sep**, **oct**, **nov**, or **dec**.

The slashes (/) can be replaced by dashes (-), periods (.), commas (,), or spaces. For example, any of the following can be entered for March 28, 2005:

03/28/05
3-28-2005
28.mar.05
05,28,3
mar 28 2005
28 3 05

If numbers are used, it is possible to enter an ambiguous date, for example, 2,7,04. In this case, **datecalc** uses the date format defined in the Tivoli Workload Scheduler message catalog to interpret the date. If the date does not match the format, **datecalc** generates an error message.

today Specifies the current system date.

tomorrow
Specifies the current system date plus one day, or, in the case of time calculations, plus 24 hours.

scheddate
Specifies the date of the production plan. This might not be the same as the system date. When used inside jobs within a job stream that is not a carried forward job stream, it returns the date when the job should run, which could be different from the production date of the job stream if the job has an `at` dependency specified.

When used inside jobs within a carried forward job stream, it returns the date when the job should have run, which could be different from the production date of the carried forward job stream if the job has an `at` dependency specified. If the `at` dependency is used with the following syntax: **at**=*hhmm* + *n* **days**, the *n* **days** are not added to the variable ***TIVOLI_JOB_DATE*** and therefore, the **datecalc** command does not report these days.

-t *time* [*base-date*]
Specify *time* in one of the following formats:

now | **noon** | **midnight** | [*h*[*h*][[:]*mm*] [**am** | **pm**] [**zulu**]

where:

now Specifies the current system date and time.

noon Specifies 12:00 p.m. (or 1200).

midnight
Specifies 12:00 a.m. (or 0000).

h[*h*][[**:**]*mm*]
Specifies the hour and minute in 12-hour time (if **am** or **pm** are used), or 24-hour time. The optional colon (:) delimiter can be replaced by a period (.), a comma (,), an apostrophe ('), the letter **h**, or a space. For example, any of the following can be entered for 8:00 p.m.:

```
8:00pm
20:00
0800pm
2000
8pm
20
8,00pm
20.00
8\'00pm
20 00
```

zulu Specifies that the time you entered is Greenwich Mean Time (Universal Coordinated Time). **datecalc** converts it to the local time.

yyyymmddhhtt
Specifies the year, month, day, hour, and minute expressed in exactly twelve digits. For example, for 2005, May 7, 9:15 a.m., enter the following: **200505070915**

offset Specifies an offset from *base-date* in the following format:

{[**+** | **>** | **-** | **<** *number* | **nearest**] | **next**} **day[s]** | **weekday[s]** | **workday[s]** | **week[s]** | **month[s]** | **year[s]** | **hour[s]** | **minute[s]** | *day* | *calendar*

where:

+ | > Specifies an offset to a later date or time. Use + (Plus) in Windows; use > (greater than) in UNIX. Be sure to add a backslash (\) before the angle bracket (>).

- | < Specifies an offset to an earlier date or time. Use - (Minus) in Windows; use < (less than) in UNIX. Be sure to add a backslash (\) before the angle bracket (>).

number
The number of units of the specified type.

nearest
Specifies an offset to the nearest occurrence of the unit type (earlier or later).

next Specifies the next occurrence of the unit type.

day[s] Specifies every day.

weekday[s]
Specifies every day except Saturday and Sunday.

workday[s]
Same as **weekday[s]**, but also excludes the dates on the **holidays** calendar.

week[s]
Specifies seven days.

month[s]
Specifies calendar months.

year[s]
Specifies calendar years.

hour[s]
Specifies clock hours.

minute[s]
Specifies clock minutes.

day Specifies a day of the week. Valid values are: **su**, **mo**, **tu**, **we**, **th**, **fr**, or **sa**.

calendar
Specifies the entries in a calendar with this name.

pic *format*
Specifies the format in which the date and time are returned. The *format* characters are as follows:

m Month number.

d Day number.

y Year number.

j Julian day number.

h Hour number.

t Minute number.

^|/ One space. Use / (slash) in Windows; use ^ (carat) in UNIX (add a backslash (\) before the carat (^) if you are in the Bourne shell).

You can also include punctuation characters. These are the same as the delimiters used in *date* and *time*.

If a format is not defined, **datecalc** returns the date and time in the format defined by the Native Language Support (NLS) environment variables. If the NLS variables are not defined, the native language defaults to C.

freedays
Specifies the name of a non-working days calendar *Calendar_Name* that is to replace **holidays** in the evaluation of *workdays*.

In this case, *workdays* is evaluated as *everyday* excluding *saturday*, *sunday*, and all the dates listed in *Calendar_Name*.

By default, *saturday* and *sunday* are not regarded as *workdays*, unless you explicitly state the opposite by adding **-sa** and **-su** after *Calendar_Name*.

You can also specify **holidays** as the name of the non-working days calendar.

Examples

To return the next date, from today, on the `monthend` calendar, run the following command:

```
>datecalc today next monthend
```

In the following examples, the current system date is Friday, April 16, 2006.

```
>datecalc today +2 days pic mm/dd/yyyy
04/16/2006
```

```
>datecalc today next tu pic yyyy\^mm\^dd
2006 04 16
```

```
>LANG=american;export LANG
>datecalc -t 14:30 tomorrow
Sat, Apr 17, 2006 02:30:00 PM
```

```
>LANG=french;datecalc -t 14:30 tomorrow
Samedi 17 avril 2006 14:30:00
```

In the following example, the current system time is 10:24.

```
>datecalc -t now \> 4 hours pic hh:tt
14:24
```

delete

Removes files. Even though this command is intended to remove standard list files you are suggested to use the `rmstdlist` command instead. The users **maestro** and **root** in UNIX, and **Administrator** in Windows can remove any file. Other users can remove only files associated with their own jobs.

Syntax

delete -V | -U

delete *filename*

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

filename

Specifies the name of the file or group of files to be removed. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumerics, dashes (-), slashes (/), backslashes (\), and underscores (_). Wildcard characters are permitted.

Note: Use this command carefully. Improper use of wildcard characters can result in removing files accidentally.

Examples

To remove all the standard list files for 4/11/04, run the following command:

```
delete d:\win32app\maestro\stdlist\2004.4.11\@
```

The following script, included in a scheduled job in UNIX, removes the job's standard list file if there are no errors:

```
...
#Remove the stdlist for this job:
if grep -i error $UNISON_STDLIST
then
exit 1
else
`maestro`/bin/delete $UNISON_STDLIST
fi
...
```

The standard configuration script, `jobmanrc`, sets the variable ***UNISON_STDLIST*** to the name of the job standard list file. For more information about `jobmanrc`, refer to "Customizing job processing on a UNIX workstation - jobmanrc" on page 39.

evtdef

Imports/exports a generic event provider XML definition file where you can add and modify custom event types. You can then use the **sendevent** command to send these events to the event processing server.

Syntax

evtdef -U | -V

evtdef [*connection parameters*] **dumpdef** *file-path*

evtdef [*connection parameters*] **loaddef** *file-path*

Arguments

-U Displays command usage information and exits.

-V Displays the command version and exits.

connection parameters
If you are using **evtdef** from the master domain manager, the connection parameters were configured at installation and do not need to be supplied, unless you do not want to use the default values.

If you are using **evtdef** from the command line client on another workstation, the connection parameters might be supplied by one or more of these methods:
- Stored in the `localopts` file
- Stored in the `useropts` file
- Supplied to the command in a parameter file
- Supplied to the command as part of the command string

For an overview of these options see "Setting up options for using the user interfaces" on page 45. For full details of the configuration parameters see the topic on configuring the command-line client access in the *Tivoli Workload Scheduler: Administration Guide*.

dumpdef *file-path*
Downloads the generic event provider XML file. The file is downloaded with the file name and path you provide in *file-path*. You can edit the file to add your own custom event types.

The name of the generic event provider supplied with the product is `GenericEventPlugIn`. You can change this name by acting on the `name` tag of the `eventPlugin` keyword.

Important: You must use this name as the value of:
- The `source` keyword of the “sendevent” on page 461 command
- The `eventProvider` keyword in the definition of the event rules triggered by these custom events.

loaddef *file-path*
Uploads the modified generic event provider XML file from the file and path you provide in *file-path*.

Comments

The following rule language schemas are used to validate your custom event definitions and, depending upon the XML editor you have, to provide syntactic help:
- eventDefinitions.xsd
- common.xsd

The files are located in the `schemas` subdirectory of the Tivoli Workload Scheduler installation directory.

Examples

In this example you:
1. Download the generic event provider XML file as file `c:\custom\myevents.xml`
```
evtdef dumpdef c:\custom\myevents.xml
```
2. Edit the file to add your own event type definitions. The first time you download the generic event provider file, it looks like this:
```
 <?xml version="1.0" encoding="UTF-8" ?>
<eventDefinitions
xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/plugins/events"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/
plugins/events/eventDefinitions.xsd" >
<eventPlugin>
<complexName displayName="Custom event" name="GenericEventPlugIn" />
<scopes>
<scope name="Generic">
<scopedef text="{Param1} on {Workstation}" />
</scope>
</scopes>
<!--  Generic Event -->
<event baseAliasName="genericEvt" scope="Generic">
<complexName displayName="Generic event" name="Event1" />
<displayDescription>The event is sent when the specified expression is
matched.</displayDescription>
<property type="string" required="true" wildcardAllowed="true"
multipleFilters="true" minlength="1">
<complexName displayName="Parameter 1" name="Param1" />
<displayDescription>The value of parameter 1</displayDescription>
</property>
<property type="string" required="true" wildcardAllowed="false"
multipleFilters="false" minlength="1>
<complexName displayName="Workstation" name="Workstation" />
<displayDescription>The workstation for which the event is
generated.</displayDescription>
</property>
</event>
</eventPlugin>
</eventDefinitions>
```

3. When finished, you upload the generic event provider XML file from file `c:\custom\myevents.xmll`

   ```
   evtdef loaddef c:\custom\myevents.xml
   ```

evtsize

Defines the size of the Tivoli Workload Scheduler message files. This command is used by the Tivoli Workload Scheduler administrator either to increase the size of a message file after receiving the message, “End of file on events file.”, or to monitor the size of the queue of messages contained in the message file. You must be **maestro** or **root** in UNIX, or **Administrator** in Windows to run **evtsize**. Stop the IBM Tivoli Workload Scheduler engine before running this command.

Syntax

evtsize -V | -U

evtsize *filename size*

evtsize -compact *filename* [*size*]

evtsize -info *filename*

evtsize -show *filename*

evtsize -info | -show pobox

Arguments

-V
Displays the command version and exits.

-U
Displays command usage information and exits.

-compact *filename* **[***size***]**
Reduces the size of the specified message file to the size occupied by the messages present at the time you run the command. You can optionally use this keyword to also specify a new file size.

-info *filename*
Displays the percentage use of the queue of messages contained in the message file.

-show *filename*
Displays the size of the queue of messages contained in the message file

filename
The name of the event file. Specify one of the following:

```
Courier.msg
Intercom.msg
Mailbox.msg
Planbox.msg
pobox/workstation.msg
```

size
The maximum size of the event file in bytes. It must be no less than 1048576 bytes (1 MB).

When first built by Tivoli Workload Scheduler, the maximum size is set to 10 MB.

Note: The size of the message file is equal to or bigger than the real size of the queue of messages it contains and it progressively increases until the queue of messages becomes empty; as this occurs the message file is emptied.

-info | -show pobox
Displays the name of the message file, within the pobox directory, with the largest queue size calculated as a percentage of the total file size. Both the name of the file and the percentage used are returned. Either **-info** and **-show** return the same results.

Examples

To set the maximum size of the Intercom.msg file to 20 MB, run the following command:

```
evtsize Intercom.msg 20000000
```

To set the maximum size of the pobox file for workstation chicago to 15 MB, run the following command:

```
evtsize pobox\chicago.msg 15000000
```

The following command:

```
evtsize -show Intercom.msg
```

returns the following output:

```
Tivoli Workload Scheduler (UNIX)/EVTSIZE 8.3 (1.2.2.4) Licensed Materials -
Property of IBM(R)
5698-WSH
(C) Copyright IBM Corp 1998, 2006 All rights reserved.
US Government User Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
IBM is a registered trademark of International Business Machines
Corporation in the United States, other countries, or both.
AWSDEK703I Queue size current 240, maximum 10000000 bytes (read 48, write 288)
```

where:

880 Is the size of the current queue of the Intercom.msg file

10000000
Is the maximum size of the Intercom.msg file

read 48
Is the pointer position to read records

write 928
Is the pointer position to write records

If the following command:

```
evtsize -info Mailbox.msg
```

returns:

```
25
```

it means that 25 percent of the file has been used.

jobinfo

Used in a job script to return information about the job. This command is not supported on dynamic agents, pools, dynamic pools, and job types with advanced options.

Syntax

jobinfo -V | -U

jobinfo *job-option* [...]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

job-option

The job option. Specify one or more of the following:

confirm_job
Returns **YES** if the job requires confirmation.

is_command
Returns **YES** if the job was scheduled or submitted using the **docommand** construct.

job_name
Returns the job's name without the workstation and job stream names.

job_pri
Returns the job's priority level.

programmatic_job
Returns **YES** if the job was submitted with using the **at** or **batch** command. UNIX only.

re_job Returns **YES** if the job is being rerun as the result of a **conman rerun** command, or the rerun recovery option.

re_type
Returns the job's recovery option (**stop**, **continue**, or **rerun**).

rstrt_flag
Returns **YES** if the job is being run as the recovery job.

rstrt_retcode
If the current job is a recovery job, returns the return code of the parent job.

schedule
Returns the name of the job stream where the job is submitted.

schedule_ia
Returns the time and date the job stream is scheduled to start.

schedule_id
Returns the *jobstream_ID* of the job stream where the job is submitted.

time_started
Returns the time the job started running.

Comments

Job option values are returned, one on each line, in the same order they were requested.

Examples

1. The script file `/jcl/backup` is referenced twice, giving it the job names **partback** and **fullback**. If the job runs as **partback**, it performs a partial backup. If it runs as **fullback**, it performs a full backup. Within the script, commands like the following are used to make the distinction:

   ```
   #Determine partial (1) or full (2):
   if [ "`\`maestro\`/bin/jobinfo job_name`" = "PARTBACK" ]
   then
   bkup=1
   else
   bkup=2
   fi
   ...
   ```

2. To display the return code of the parent job, if the current job is a recovery job, run the following command:

   ```
   $ jobinfo rstrt_retcode
   ```

   The first job (parent job) has been defined in the script `recovery.sh` while the second job (recovery job) gets enabled only if the first job abends.

   When combined with a return code condition, **jobinfo rstrt_retcode** can be used to direct the recovery job to take different actions depending on the parent job's return code. A recovery job is shown in the example below:

   ```
   $JOBS
   MASTER#DBSELOAD DOCOMMAND "/usr/local/tws/maestro/scripts/populate.sh"
   STREAMLOGON "^TWSUSER^"
   DESCRIPTION "populate database manual"
   RECOVERY RERUN AFTER MASTER#RECOVERY
   RCCONDSUCC "(RC = 0) OR ((RC > 4) AND (RC < 11))"
   ```

   Note: The job is defined with the recovery action RERUN. This enables the recovery job to take some corrective action, before the parent job attempts to run again.

   The recovery job itself is defined as shown in the example below:

   ```
   $ JOBS
   MASTER#RECOVERY DOCOMMAND "^TWSHOME^/scripts/recovery.sh"
   STREAMLOGON "^TWSUSER^"
   DESCRIPTION "populate database recovery manual"
   RECOVERY STOP
   ```

jobstdl

Returns the names of standard list files. This command must be run by the user for which Tivoli Workload Scheduler was installed. If you use this command without any parameters, ensure that you are logged on as a Tivoli Workload Scheduler user.

Syntax

jobstdl -V | -U

jobstdl
  [**-day** *num*]
  [{**-first** | **-last** | **-num** *n* | **-all**}]

[**-twslog**]
[{**-name** ["*jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"
| *jobnum* | **-schedid** *jobstream_id.jobname*}]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-day *num*
Returns the names of standard list files that are the specified number of days old (1 for yesterday, 2 for the day before yesterday, and so on). The default is zero (today).

-first Returns the name of the first qualifying standard list file.

-last Returns the name of the last qualifying standard list file.

-num *n*
Returns the name of the standard list file for the specified run of a job.

-all Returns the name of all qualifying standard list files.

-twslog
Returns the path of the current day *stdlist* file.

-name ["*jobstreamname***[(***hhmm date***), (***jobstream_id***)].]***jobname***" |** *jobnum*
Specifies the instance of the job stream and name of the job for which standard list file names are returned.

jobnum
Specifies the job number of the job for which standard list file names are returned.

-schedid *jobstream_id***.***jobname*
Specifies the job stream ID and name of the job for which standard list file names are returned.

Comments

File names are returned in a format suitable for input to other commands. Multiple names are returned separated by a space.

When you use the full syntax of the **-name** argument, the square brackets in the expression `[(`*`hhmm date`*`), (`*`jobstream_id`*`)]` are part of the command, not syntax indicators. Also, the whole job identification string must be enclosed in double quotes if the part identifying the job stream instance contains blanks. For example, because the *schedtime*, represented by *`hhmm date`*, has a space in it, you must enclose the whole job identification in double quotes.

You can also run abbreviated versions of the **-name** argument using a simpler syntax. If you want less specific outputs from the command, you can specify just the *schedtime* (the *date* is not required if it is for the same day) or the *jobstream_id* together with the *jobname*. As long as there are no blanks in the arguments, you can omit the double quotes. You can also omit the square brackets if you do not specify both the *schedtime* and the *jobstream_id*.

The following examples show the syntax you must use with the **-name** argument for the different types of information you expect in return, ranging from the more specific to the more general. In the example, `job_stream1` is the name of the job stream, `0600 04/05/06` is the scheduled time, `0AAAAAAAAAAAAAB5` is the job stream ID, and `job1` is the job name. The job number of `job1` is `310`. You can run **jobstdl** for `job1` as follows:

```
jobstdl -name "job_stream1[(0600 04/05/10),(0AAAAAAAAAAAAAB5)].job1"
```

Returns the standard list file name of `job1` for the specific instance of `job_stream1` with the specified *schedtime* and *jobstream_id*.

```
jobstdl -name job_stream1(0AAAAAAAAAAAAAB5).job1
```

Returns the standard list file name for `job1` for the instance of `job_stream1` with ID `0AAAAAAAAAAAAAB5`.

```
jobstdl -name "job_stream1(0600 04/05/10).job1"
```

Returns the standard list file names for `job1` for all possible instances of `job_stream1` scheduled to run at 0600 of 04/05/10.

```
jobstdl -name job_stream1(0600).job1
```

Returns the standard list file names for `job1` for all possible instances of `job_stream1` scheduled to run at 0600 of the current day.

```
jobstdl -name 310
```

Returns the standard list file names for `job1` for all the instances it had job number `310`.

Examples

To return the path names of all standard list files for the current day, run the following command:

```
jobstdl
```

To return the path name of the standard list for the first run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` on the current day, run the following command:

```
jobstdl -first -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To return the path name of the standard list for the first run of job `0AAAAAAAAAAAAAEE.DIR` on the current day, run the following command:

```
jobstdl -first -schedid 0AAAAAAAAAAAAAEE.DIR
```

To return the path name of the standard list for the second run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` on the current day, run the following command:

```
jobstdl -num 2 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To return the path names of the standard list files for all runs of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` from three days ago, run the following command:

```
jobstdl -day 3 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To return the path name of the standard list for the last run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` from four days ago, run the following command:

```
jobstdl -day 4 -last -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To return the path name of the standard list for job number **455**, run the following command:

```
jobstdl 455
```

To print the contents of the standard list file for job number **455**, run the following command:

```
cd `maestro`/bin
lp -p 6 `jobstdl 455`
```

maestro

Returns the path name of the Tivoli Workload Scheduler home directory, referred to as *TWS_home*.

Syntax

maestro [**-V** | **-U**]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

Examples

To display the Tivoli Workload Scheduler home directory, run the following command:

```
$ maestro
/usr/lib/maestro
```

To change the directory to the Tivoli Workload Scheduler home directory, run the following command:

```
$ cd `maestro`
```

makecal

Creates a custom calendar. In UNIX, the Korn shell is required to run this command.

Syntax

makecal [**-V** | **-U**]

makecal
  [**-c** *name*]
  -d *n*
    | **-e**
    | {**-f** **1** | **2** | **3** **-s** *date*}
    | **-l**
    | **-m**
    | **-p** *n*
    | {**-r** *n* **-s** *date*}
    | **-w** *n*
  [**-i** *n*]
  [**-x** | **-z**]
  [**-freedays** *Calendar_Name* [**-sa**] [**-su**]]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-c *name*
: Specifies a name for the calendar. Tivoli Workload Scheduler keywords (such as *Freedays* or *Schedul*e) cannot be used as calendar names. The name can contain up to eight alphanumeric characters and must start with a letter. Do not use the names of weekdays for the calendar names. The default name is: **C***hhmm*, where *hhmm* is the current hour and minute.

-d *n* Specifies the *n*th day of every month.

-e Specifies the last day of every month.

-f 1 | 2 | 3
: Creates a fiscal month-end calendar containing the last day of the fiscal month. Specify one of the following formats:

1	4-4-5 week format
2	4-5-4 week format
3	5-4-4 week format

This argument requires the **-s** argument.

-i *n* Specifies to insert *n* dates in the calendar.

-l Specifies the last workday of every month. For this argument to work properly, the production plan (`Symphony` file) and the **holidays** calendar must already exist.

Note: Using this argument results in the new calendar also including the last workday of the month that precedes the date of creation of the calendar.

-m Specifies the first and fifteenth days of every month.

-p *n* Specifies the workday before the *n*th day of every month. For this argument to work properly, the production plan (`Symphony` file) and the **holidays** calendar must already exist

-r *n* Specifies every *n*th day. This argument requires the **-s** argument.

-s *date* Specifies the starting date for the **-f** and **-r** arguments. The date must be enclosed in quotation marks, and must be valid and unambiguous, for example, use **JAN 10 2005**, not **1/10/05**. See *base-date* for **datecalc** on page 440 for more information about date formats.

-w *n* Specifies the workday after the *n*th of the month. For this argument to work properly, the production plan (Symphony file) and the **holidays** calendar must already exist.

-x Sends the calendar output to **stdout** instead of adding it to the database.

-z Adds the calendar to the database and compiles the production plan (`Symphony` file).

Note: This argument re-submits jobs and job streams from the current day's production plan. It might be necessary to cancel job streams and jobs.

-freedays

Specifies the name of a non-working days calendar *Calendar_Name* that is to replace **holidays** in the evaluation of *workdays*.

In this case, *workdays* is evaluated as *everyday* excluding *saturday*, *sunday* and all the dates listed in *Calendar_Name*.

By default, *saturday* and *sunday* are not regarded as *workdays*, unless you explicitly state the opposite by adding **-sa** and/or **-su** after *Calendar_Name*.

You can also specify **holidays** as the name of the non-working days calendar.

This keyword affects the processing of **makecal** with options **-l**, **-p**, and **-w**.

Examples

To make a two-year calendar with the last day of every month selected, run the following command:

```
makecal -e -i 24
```

To make a calendar with 30 days that starts on May 30, 2005, and has every third day selected, run the following command:

```
makecal -r 3 -s "30 MAY 2005" -i 30
```

metronome

Metronome is replaced by **tws_inst_pull_info**. See *IBM Tivoli Workload Scheduler Troubleshooting Guide* for information on this command.

morestdl

Displays the contents of standard list files. This command must be run by the user for which Tivoli Workload Scheduler was installed. If you use this command without any parameters, ensure that you are logged on as a Tivoli Workload Scheduler user. This command is supported for fault-tolerant agents and standard agents.

Syntax

morestdl -V | -U

morestdl
  [**-day** *num*]
  [**-first** | **-last** | **-num** *n* | **-all**]
  [**-twslog**]
  [{**-name** ["*jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"
    | *jobnum* | **-schedid** *jobstream_id.jobname*}]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-day *num*

Displays standard list files that are the specified number of days old (1 for yesterday, 2 for the day before yesterday, and so on). The default is zero (today).

-first Displays the first qualifying standard list file.

-last Displays the last qualifying standard list file.

-num *n*
Displays the standard list file for the specified run of a job.

-all Displays all qualifying standard list files.

-twslog
Displays the content of the current day *stdlist* file.

-name ["*jobstreamname* **[(***hhmm date***),(***jobstream_id***)].]***jobname***"|***jobnum*
Specifies the instance of the job stream and the name of the job for which the standard list file is displayed.

jobnum
Specifies the job number of the job for which the standard list file is displayed.

-schedid *jobstream_id***.***jobname*
Specifies the job stream ID and name of the job for which standard list file names are returned.

Comments

The square brackets in the expression `[(`*`hhmm date`*`), (`*`jobstream_id`*`)]` are part of the command, not syntax indicators. This means that you can supply either of the following for the **-name** argument:

```
morestdl -name ["jobstreamname[(hhmm date),(jobstream_id)].jobname"
morestdl -name jobnum
```

The whole job identification string must be enclosed in double quotes if the part identifying the job stream instance contains blanks. For example, because the *schedtime*, represented by *`hhmm date`*, has a space in it you must enclose the whole job identification in double quotes.

If you just want to identify a job name, you do not need the double quotes.

The following is an example of the syntax to use when identifying a job both with and without its job stream. In the example, `job_stream1` is the name of the job stream, `0600 04/05/06` is the scheduled time, `0AAAAAAAAAAAAAB5` is the job stream ID, and `job1` is the job name. You can run the **morestdl** command against `job1` using either of these two formats:

```
morestdl -name "job_stream1[(0600 04/05/06),(0AAAAAAAAAAAAAB5)].job1"
morestdl -name job1
```

Examples

To display the standard list file for the first run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` on the current day, run the following command:

```
morestdl -first -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To display the standard list file for the first run of job `0AAAAAAAAAAAAAEE.DIR` on the current day, run the following command:

```
morestdl -first -schedid 0AAAAAAAAAAAAAEE.DIR
```

To display the standard list file for the second run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR` on the current day, run the following command:

```
morestdl -num 2 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAAEE)].DIR"
```

To display the standard list files for all runs of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAAEE)].DIR` from three days ago, run the following command:

```
morestdl -day 3 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAAEE)].DIR"
```

To display the standard list file for the last run of job `MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAAEE)].DIR` from four days ago, run the following command:

```
morestdl -day 4 -last -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAAEE)].DIR"
```

To print the standard list file for job number 455, run the following command:

```
morestdl 455 | lp -p 6
```

parms

Manages parameters defined locally on workstations. Parameters managed by **parms** can only be used in job or job stream definitions with the **scriptname** or **opens** keywords or in a job script file.

These parameters are resolved at submission time on the workstation where the job or job stream is submitted. If there is no match between the specified *parametername* and the name of the parameters defined in the local database on the workstation, then a *null* value is returned.

Authorization

You must have ***display*** access to the locally defined parameters database. In addition you must be authorized with the following access:

build on object file
If you use the **-b** option to create or rebuild the local parameters database.

delete If you use the **-d** option to delete parameter definitions.

modify on object file
If you use the **-replace** option to add or modify parameter definitions.

Syntax

parms {[**-V** | **-u**] | **-build**}

parms {-replace | -extract} *filename*

parms [**-d**]*parametername*

parms -c *parametername value*

Arguments

-V Displays the command version and exits.

-u Displays command usage information and exits.

-build Creates the parameters database on the workstation if it does not exist. Rebuilds the parameters database, removing unused records and avoiding fragmentation from numerous additions and deletions, if it already exists.

-extract
Extracts all parameter definitions from the local database and stores them in the file with name *filename*. Use this option if you want to export local parameter definitions to import them as global parameter definitions into the scheduling objects database using the "add" on page 248 or the "replace" on page 283 commands.

-replace
Add in the local database new parameter definitions stored in a file named *filename* or substitute the already existing ones. Use this option if you want to import, as local parameter definitions, the global parameter definitions contained in the file named *filename* and extracted from the scheduling objects database using the "extract" on page 260 command.

-d Deletes the parameters with name *parametername* from the local database on the workstation.

parametername
Specifies the name of the parameter whose value is displayed. When used with the argument **-d** it represents the name of the parameter to be deleted.

-c *name value*
Specifies the name and the value of a parameter. The name can contain up to 16 alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter. The value can contain up to 72 characters. Enclose the value in double quotes if it contains special characters. If the parameter does not exist, it is added to the database. If the parameter already exists, its value is changed.

Comments

When **parms** is run on the command line without arguments, it prompts for parameter names and values.

The use of **parms** in either job definitions and job script files requires that the parameter already exists locally in the parameters database on the workstation.

This is a sample usage of a local parameter, `MYFILE`, in a file dependency clause:

```
schedule test_js
on everyday
opens "/usr/home/tws_99/'/usr/home/tws_99/bin/parms MYFILE'"
:
test_job
end
```

The following example explains how the variable ***var*** enclosed by carets (^) is replaced while the job is in process. If the job is submitted as an ad hoc job, the parameter ***var*** is expanded, that means replaced by the value assigned to ***var*** in the local database, at submission time and not when the job launches.

UNIX job definition example:

```
DATA#UX_P_TEST DOCOMMAND "ls ^var^"
STREAMLOGON "mae82"
DESCRIPTION "Test parms in job definition on UNIX."
RECOVERY STOP
```

Windows job definition example:

```
BORG#WIN_P_TEST DOCOMMAND "dir ^var^"
STREAMLOGON "mae82"
DESCRIPTION "Test parms in job definition on Windows."
RECOVERY STOP
```

When used in a job script file, the parameter is not expanded until the script launches. It is not expanded when the job stream containing the job is processed by **JnextPlan**. These are examples on how to use the ***var*** parameter in job script files.

UNIX script example:

```
#!/bin/sh
TWS_HOME="/opt./tws/mae82/maestro"
export TWS_HOME
MDIR='$TWS_HOME/bin/parms var'
export MDIR
ls -l $MDIR
```

Windows script example:

```
set TWS_HOME=d:\win32app\TWS\mae82\maestro
echo %TWS_HOME%
FOR /F "Tokens=*" %%a in (%TWS_HOME%\bin\parms var) do set MDIR=%%a
echo %MDIR%
dir %MDIR%
```

Examples

To return the value of `myparm`, run the following command:

```
parms myparm
```

To change the value of `myparm`, run the following command:

```
parms -c myparm "item 123"
```

To create a new parameter named `hisparm`, run the following command:

```
parms -c hisparm "item 789"
```

To change the value of `myparm` and add `herparm`, run the following command:

```
parms
Name of parameter ? myparm < Return>
Value of parameter? "item 456" < Return>
Name of parameter ? herparm < Return>
Value of parameter? "item 123" < Return>
Name of parameter ? < Return>
```

release

Releases jobs and job streams from **needs** dependencies on a resource. This command must be issued only from within the job script file.

Syntax

release -V | -U

release
[**-s**]
[*workstation*#]
resourcename
[*count*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-s Releases the **needs** dependency from the specified resource only at the job stream level.

If **-s** is not used, the **needs** dependency from the specified resource is released at the job level, or at the job stream level if the **needs** dependency from that resource is not found at the job level.

workstation#
Specifies the name of the workstation or workstation class on which the resource is defined. The default is the local workstation.

resourcename
Specifies the name of the resource involved in the **needs** dependency.

count Specifies the number of units of the resource to be released.

Comments

Units of a resource are acquired by a job or job stream at the time it is launched and are released automatically when the job or job stream completes. The **release** command can be used in a job script to release resources before job or job stream completion or to release manually jobs and job streams from needs dependencies in emergency situations.

Examples

In the following job stream, two units of the `dbase` resource are required by job stream `sked5`:

```
schedule ux1#sked5 on tu
needs 2 dbase :
job1
jobrel follows job1
job2 follows jobrel
end
```

To release the `dbase` resource before `job2` begins, the script file for `jobrel` contains the following command:

```
`maestro`/bin/release -s dbase
```

Note: The **-s** argument can be omitted, because no resources were reserved at the job level.

rmstdlist

Removes or displays standard list files based on the age of the file. This utility should be used by the Tivoli Workload Scheduler administrator to maintain the scheduling environment.

Syntax

rmstdlist -V | -U

rmstdlist [**-p**] [*age*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-p Displays the names of qualifying standard list file directories. No directories or files are removed. If you do not specify **-p**, the qualifying standard list files are removed.

age The minimum age, in days, of standard list file directories to be displayed or removed. The default is 10 days.

Note: Because the list of directories and files shown or deleted using **rmstdlist** is produced based on the last time they were accessed, the dates shown in the list of directories could differ from the dates displayed in the list of files.

Syntax

As a rule, you should regularly remove standard list files somewhere between every 10-20 days. Larger backlogs may be harder to manage and, if the number of files becomes exceedingly large, you might be required to erase some of them manually before you can use **rmstdlist** again.

This problem might occur on AIX systems, particularly, because of a currently unresolved limitation with the `rm -rf` command. When **rmstdlist** fails because of this limitation, it does not display any errors other than exit code 126. If you would rather have the `rm -rf` error displayed, you can edit the `rmstdlist` script in the following way:

1. Locate the script in the *`TWS_home`*`/bin` directory
2. Find the line:
   ```
   rm -rf `cat /tmp/rm$$` 2> /dev/null
   ```
3. Remove the redirection to `/dev/null` so that the line becomes:
   ```
   rm -rf `cat /tmp/rm$$`
   ```

Examples

To display the names of standard list file directories that are more than 14 days old, run the following command:

```
rmstdlist -p 14
```

To remove all standard list files (and their directories) that are more than seven days old, run the following command:

```
rmstdlist 7
```

sendevent

The command sends the custom events defined with the evtdef command to the event processor server currently active in the production plan. As the events are received by the event processor, they trigger the event rules in which they were specified.

Users can override the default destination server (defined by global options) by specifying the host and the port of a new server.

Syntax

sendevent -V | ? | -help | -u | -usage

sendevent [**-hostname** *hostname*][**-port** *port*] *eventType source* [[*attribute*=*value*]...]

Arguments

-V Displays the command version and exits.

? | -help | -u | -usage
: Displays command usage information and exits.

-hostname *hostname*
: Specifies the host name of an alternate event processor server other than the currently active one. This parameter is required if the command is launched from a command-line client.

-port *port*
: Specifies the port number of an alternate event processor server other than the currently active one. This parameter is required if the command is launched from a command-line client.

eventType
: One of the custom event types defined with the evtdef command in the generic event provider and specified as the triggering event in an event rule definition.

source The name of the event provider that you customized with evtdef. This is also the name you must specify as the argument for the `eventProvider` keyword in the definition of the event rules triggered by these custom events.

The default name is `GenericEventPlugIn`.

attribute=*value*
: One or more of the attributes qualifying the custom event type that are specified as the triggering event attributes for the event rule.

Comments

This command can be run also on systems where only the Tivoli Workload Scheduler remote command line client is installed.

Examples

In this example an application sends the `BusProcCompleted` custom event type to an alternate event processor server named `master3`. The event is that file `calcweek` finished processing.

```
sendevent -hostname master3 -port 4294 BusProcCompleted GenericEventPlugIn
TransacName=calcweek Workstation=ab5supp
```

The file name and the associated workstation are the two `BusProcCompleted` event attributes that were specified as triggering event attributes in an associated event rule.

showexec

Displays the status of running jobs. This command applies to UNIX only. This command is for standard agents. On domain managers and fault-tolerant agents, use the **conman showjobs** command instead.

Syntax

showexec [**-V** | **-U** | **INFO**]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

INFO Displays the name of the job file instead of the user, date, and time.

Results

The output of the command is available in two formats: **standard** and **INFO**.

Examples

To display running jobs in the **standard** format, run the following command:

```
showexec
```

To display running jobs in the **INFO** format, run the following command:

```
showexec INFO
```

Standard format

CPU The workstation on which the job runs.

Schedule
The name of the job stream in which the job runs.

Job The job name.

Job# The job number.

User The user name of the job.

Start Date
The date the job started running.

Start Time
The time the job started running.

(Est) Elapse
The estimated time, in minutes, that the job will run.

Info format

CPU The workstation on which the job runs.

Schedule
The name of the job stream in which the job runs.

Job The job name.

Job# The job number.

JCL The file name of the job.

shutdown

Stops the Tivoli Workload Scheduler processes, and optionally also stops the embedded application server. Applies to Windows workstations only. You must have ***shutdown*** access to the workstation.

Syntax

shutdown [-V | -U] [-appsrv]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-appsrv
Stops also WebSphere Application Server.

Comments

Make sure the *TWS_user* you are using belongs to the Admnistrators group defined on the Windows workstation.

Examples

To display the command name and version, run the following command:

```
shutdown -V
```

To stop both the Tivoli Workload Scheduler processes and WebSphere Application Server, run the following command:

```
shutdown -appsrv
```

ShutDownLwa

Stops the dynamic agent. No specific access to the workstation is required. Run this command locally on the dynamic agent you want to stop.

Syntax

ShutDownLwa

Arguments

No arguments are necessary.

Examples

To stop the dynamic agent , run the following command:

```
ShutDownLwa
```

StartUp

Starts **netman**, the Tivoli Workload Scheduler network management process.

You must have ***start*** access to the workstation.

Syntax

StartUp [**-V** | **-U**]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

Comments

In Windows, the **netman** service is started automatically when a computer is restarted. **StartUp** can be used to restart the service if it is stopped for any reason.

In UNIX, the **StartUp** command can be run automatically by invoking it from the `/etc/inittab` file, so that WebSphere Application Server infrastructure and **netman** is started each time a computer is rebooted. **StartUp** can be used to restart **netman** if it is stopped for any reason.

The remainder of the process tree can be restarted with the

```
conman start
conman startmon
```

commands. See conman “start” on page 385 for more information.

Examples

To display the command name and version, run the following command:

```
StartUp -V
```

To start the **netman** process, run the following command:

```
StartUp
```

StartUpLwa

Starts the dynamic agent .

No specific access to the workstation is required. Run this command locally on the dynamic agent you want to start.

Syntax

StartUpLwa

Arguments

No arguments are necessary.

Examples

To start the dynamic agent, run the following command:

```
StartUpLwa
```

tws_inst_pull_info

This is is a script that produces information about your Tivoli Workload Scheduler environment and your local workstation, and can take a snapshot of DB2 and WebSphere Application Server data on the master domain manager, saving them as a dated package.

It can also generate a report containing not only the results of the snapshot, but also many configuration and environment parameters. The tool is useful when describing a problem to IBM Software Support. For best results, it must be run as soon as the problem is discovered.

Comments

Refer to *IBM Tivoli Workload Scheduler Troubleshooting Guide* for more information about this command.

version

Displays information about the current release of Tivoli Workload Scheduler installed on the system. This command applies to UNIX only. The information is extracted from a version file.

Syntax

version -V | **-u** | **-h**

version [**-a**] [**-f** *vfile*] [*file* [...]]

Arguments

-V Displays the version of the command and exits.

-u Displays command usage information and exits.

-h Displays command help information and exits.

-a Displays information about all product files. The default is to display information only about the specified files.

-f *vfile* Specifies the path and name of the version file if different from the default setting. The default is a file named `version.info` in the current working directory.

file Specifies the names of product files, separated by spaces, for which version information is displayed. The default is to display no file information, or, if **-a** is used, all file information.

Results

The output header contains the product name, version, operating system, patch level, and installation date. The remaining display lists information about the file or files specified. The files are listed in the following format:

File The name of the file.

Revision
The revision number of the file.

Patch The patch level of the file, if any.

Size (bytes)
The size of the file in bytes.

Checksum
The checksum for the file. Checksum is calculated using the UNIX **sum** command. On AIX®, **sum** is used with the **-o** argument.

Comments

Tivoli Workload Scheduler file information is contained in the `version.info` file. This file is placed in the *TWS_home*/`version` directory during installation. The `version.info` file is in a specific format and is not altered.

You can move the `version.info` file to another directory. However, you must then include the **-f** argument to locate the file.

Examples

To display information about the release of Tivoli Workload Scheduler installed, run the following command:

```
./version
```

A sample output of this command is:

```
IBM Tivoli Workload Scheduler/VERSION 8.3 (9.9) (C) Copyright IBM Corp 1998, 2006

IBM Tivoli Workload Scheduler 8.3 UNIX linux-ix86
PATCH February 2006
```

To display information about all files, run the following command:

```
version/version -a -f version/version.info
```

To display information about the file `customize`, run the following command:

```
cd version
./version customize
```

To display information about the file `customize`, when `version.info` is in `/apps/maestro`, run the following command:

```
cd version
./version -f /apps/maestro/version.info customize
```

Unsupported commands

The following unsupported utility commands provide functions in Windows that are similar to UNIX **ps** and **kill** commands. They can be used if similar Windows utilities are not available.

Syntax

listproc

killproc *pid*

Comments

listproc
Displays a tabular listing of processes on the system.

killproc

Kills the process with the process ID *pid*.

Note: When run by the Administrator, **killproc** is capable of killing system processes.

Chapter 14. Using utility commands in the dynamic environment

This chapter describes Tivoli Workload Scheduler utility commands for the dynamic environment. While some commands run on the master domain manager or on a dynamic domain manager, others run on the dynamic agents. They are installed in path `TWA_home/TDWB/bin` and run with the UNIX and Windows operating systems. You run utility commands from the operating system command prompt.

Table 64 contains the list of the utility commands, and for each command, its description and the type of workstation where you can run it.

Table 64. List of utility commands for dynamic workstations

Command	Description	Workstation type
exportserverdata	Downloads the list of dynamic workload broker instances from the Tivoli Workload Scheduler database and changes a port number or a host name.	master domain manager or dynamic domain manager
importserverdata	Uploads the list of dynamic workload broker instances to the Tivoli Workload Scheduler database after editing the temporary file to change a port number or a host name.	master domain manager or dynamic domain manager
movehistorydata	Moves the data present in the Tivoli Workload Scheduler database to the archive tables	master domain manager or dynamic domain manager
param	Creates, displays, and deletes variables and user passwords on dynamic agents.	dynamic agents
resource	Creates, modifies, associates, queries, or sets resources online or offline.	master domain manager, dynamic domain manager, or dynamic agents
twstrace	Modifies at runtime the settings for tracing on dynamic agents	dynamic agents

Note: To remove the job logs files for dynamic agents workstations, set the value of the `MaxAge` property in the `JobManager.ini`. For more details, see *IBM Tivoli Workload Scheduler manuals: Administration guide - Configuring the agent - Configuring common launchers properties [Launchers]*.

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`**

exportserverdata

Use the **exportserverdata** command to download the list of dynamic workload broker instances from the Tivoli Workload Scheduler database and change a port number or a host name.

Syntax

exportserverdata ?

exportserverdata -dbUsr *db_user_name* **-dbPwd** *db_user_password* **-exportFile** *filename*

Description

This command extracts a list of URIs (Uniform Resource Identifier) of all the dynamic workload broker instances from the Tivoli Workload Scheduler database and copies them to a temporary file so that, if either the hostname or the port number of any of the instances listed are changed, the administrator can record this information in the file and place it back in the database with the `importserverdata` command. By default, the list of URIs is saved to the server.properties file, located in the current directory.

This action is necessary because the list of dynamic workload broker instances must be kept up-to-date at all times, since the Resource Advisor agents periodically connect to the active instance to send their data about the resources discovered in each computer. They are able to automatically switch between the instances of this list and find the active one to copy these data in its Resource Repository. Since the master domain manager and every backup master are installed with a dynamic workload broker instance, the active dynamic workload broker instance runs in the master domain manager, while an idle instance resides in each backup master.

The URI pointing to each dynamic workload broker instance is the following:

```
https://hostname:port_number/JobManagerRESTWeb/JobScheduler
```

You can change only the hostname and the port number.

Important: The list is ordered. You can change the order of the instances as they appear in this list, and the agents will follow this order. If you have several backup masters and decide to follow a specific switching order when a master fails, you can instruct the agents to switch to the right instance using this ordered list, speeding up the transition time.

If your Tivoli Workload Scheduler database is DB2 and you use FIPS security, to run this command successfully you need to have the **com.ibm.tdwb.dao.rdbms.useSSLConnections** option set to `TRUE` in the `CLIConfig.properties` file.

Options

? Displays help information.

-dbUsr *db_user_name*
The user name required to access the Tivoli Workload Scheduler database server.

-dbPwd *db_user_password*
: The user password required to access the Tivoli Workload Scheduler database server.

-exportFile *filename*
: The name of the temporary file where the URIs extracted from the database are copied for editing. This text file is created when you run the command and you can open it with any editor to change the hostname or the port number. If you do not specify a path, the file is created in the same directory where the command is located, that is:

```
<TWA_home>/TDWB/bin
```

If you do specify a different path, make sure the path exists before you run this command.

Example

To download the current list of all (active and backup) dynamic workload broker instances and copy them in a file named `c:\myservers\uris160709`, run:

```
exportserverdata -dbUsr twsadm -dbPwd fprefect -exportFile c:\myservers\uris160709
```

The command returns file `uris160709`, that looks like this:

```
https://accrec015:42127/JobManagerRESTWeb/JobScheduler
https://prodop099:52529/JobManagerRESTWeb/JobScheduler
https://prodop111:31116/JobManagerRESTWeb/JobScheduler
```

`prodop099` is the active dynamic workload broker instance because is hosted by the currently active master domain manager, whereas `accrec015` and `prodop111` are idle because they are hosted by backup masters.

You can edit this file to apply your changes before using the `importserverdata` command to upload the URIs back to the database.

See Also

"importserverdata"

importserverdata

Use the **importserverdata** command to upload the list of dynamic workload broker instances to the Tivoli Workload Scheduler database after editing the temporary file to change a port number or a host name.

Syntax

importserverdata ?

importserverdata -dbUsr *db_user_name* **-dbPwd** *db_user_password* **-importFile** *filename*

Description

This command puts back the list of dynamic workload broker instances in the Tivoli Workload Scheduler database from the temporary file where they were previously downloaded with the `exportserverdata` command.

Use the `exportserverdata` and `importserverdata` commands if you have to record any hostname or port number changes in the URIs of the instances. This is necessary to keep the list of dynamic workload broker instances up-to-date at all times, since the Resource Advisor agents periodically connect to the active instance to send their data about the resources discovered in each computer. They are able to automatically switch between the instances of this list and find the active one to copy these data in its Resource Repository. Since the master domain manager and every backup master are installed with a dynamic workload broker instance, the active dynamic workload broker instance runs in the master domain manager, while an idle instance resides in each backup master.

Important: The list is ordered. You can change the order of the instances as they appear in this list, and the agents will follow this order. If you have several backup masters and decide to follow a specific switching order when a master fails, you can instruct the agents to switch to the right instance using this ordered list, speeding up the transition time.

If your Tivoli Workload Scheduler database is DB2 and you use FIPS security, to run this command successfully you need to have the **com.ibm.tdwb.dao.rdbms.useSSLConnections** option set to `TRUE` in the `CLIConfig.properties` file.

Options

? Displays help information.

-dbUsr *db_user_name*
The user name required to access the Tivoli Workload Scheduler database server.

-dbPwd *db_user_password*
The user password required to access the Tivoli Workload Scheduler database server.

-importFile f*filename*
The name of the temporary file you specified with the `-exportFile` keyword in the `exportserverdata` command.

Example

To upload the edited list of dynamic workload broker instance URIs from file `c:\myservers\uris160709` to the Tivoli Workload Scheduler database, run:

```
importserverdata -dbUsr twsadm -dbPwd fprefect -importFile c:\myservers\uris160709
```

See Also

"exportserverdata" on page 473

movehistorydata

Use the **movehistorydata** when access to the database becomes too slow. This command moves the data present in the Job Repository to the archive tables

Slow database access might be due to a huge number of records being present in the database, for example when bulk job submissions are performed.

When you run this command, the jobs are moved to the following tables in the database:

JOA_JOB_ARCHIVES
Contains archived job instances

JRA_JOB_RESOURCE_ARCHIVES
Contains resource information related to the jobs

MEA_METRIC_ARCHIVES
Contains metrics collected for the jobs

For more information on historical tables, refer to the *Tivoli Workload Scheduler: Administration Guide*, *SC23-9113*.

Note: Depending on the number of jobs and accesses to the database, a cleanup operation might cause some peaks in memory or CPU usage.

If your Tivoli Workload Scheduler database is DB2 and you use FIPS security, to run this command successfully you need to have the **com.ibm.tdwb.dao.rdbms.useSSLConnections** option set to `TRUE` in the `CLIConfig.properties` file.

Syntax

movehistorydata ?

movehistorydata -dbUsr *db_user_name***-dbPwd** *db_user_password* [**-successfulJobsMaxAge** *successfulJobsMaxAge* [**-notSuccessfulJobsMaxAge** *notSuccessfulJobsMaxAge*][**-archivedJobsMaxAge** *archivedJobsMaxAge*]]

Description

This command performs a cleanup operation on the Job Repository database. Based on the values you specify, information on submitted jobs is moved to the archive database and the information in the archive database is deleted.

Use this command to temporarily override the settings defined in the **JobDispatcherConfig.properties** file, when unexpected events require an immediate database cleanup. The settings in the **JobDispatcherConfig.properties** file remain unchanged. For more information on the **JobDispatcherConfig.properties** file, refer to the *Tivoli Workload Scheduler: Administration Guide*.

Options

? Displays help information.

`-dbUsr` *`db_user_name`*
Specifies the username for a user authorized to perform operations on the database server.

`-dbPwd` *`db_user_password`*
Specifies the password for a user authorized to perform operations on the database server.

`-successfulJobsMaxAge` *`successfulJobsMaxAge`*
Specifies how many hours jobs completed successfully or canceled must be kept in the Job Repository database before being archived. The default value is 240 hours, that is ten days.

-notSuccessfulJobsMaxAge *notSuccessfulJobsMaxAge*
Specifies how many hours jobs completed unsuccessfully or in unknown status must be kept in the Job Repository database before being archived. The default value is 720 hours, that is 30 days.

-archivedJobsMaxAge *archivedJobsMaxAge*
Specifies how many hours jobs must be kept in the archive database before being archived. The default value is 720 hours, that is 30 days.

Return Values

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

0	Indicates that **movehistorydata** completed successfully.
< > 0	Indicates that **movehistorydata** failed.

Examples

1. To move to the archive database all successful jobs completed in the last 40 hours, type the following command:
```
movehistorydata -dbUsr halmst -dbPwd dgordon -successfulJobsMaxAge 40
```
2. To move to the archive database all jobs in all supported statuses and remove from the archive database all jobs older than 700 hours, type the following command:
```
movehistorydata -dbUsr halmst -dbPwd dgordon -successfulJobsMaxAge 0
                -notSuccessfulJobsMaxAge 0 -archivedJobsMaxAge 700
```

param

Use the **param** command to define and manage user passwords and variables locally on dynamic agents and Tivoli Workload Scheduler for z/OS agents.

You can use this command on job types with advanced options. The values are resolved at submission time on the agent where the job is submitted.

Authorization

To create, delete, or display variables or passwords, you must have Administrator or root user rights on the workstation that runs the agent or *TWS_user* rights on the agent.

Syntax

param -u | -V |
  {-c | -ec} [*file.section.*|*file..*|*section.*] *variable* [*value*] |
  [*file.section.*|*file..*|*section.*] *variable* |
  {-d | -fd} [*file.section.*|*file..*|*section.*] *variable*

Arguments

-u Displays command usage information and exits.

-V Displays the command version and exits.

-c | -ec
Creates variable or password *variable* and defines its value *value*. The variable or password is placed in a namespace *file* that you can organize in one or more sections named *section*.

If you do not provide a file name *file*, the variable or password is placed in default file `jm_variables` in path *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files` (`/TWA/TWS/ITA/cpa/config/jm_variables_files`) on the dynamic agent.

If you do not provide a section name *section*, the variable or password is placed in the main body of the file.

Important: If you are defining a password, you must specify a section named `password` for *variable*. This specifies that *variable* is a password.

If you are creating a variable, *variable* is the variable name and *value* is its value. If you are creating a password, *variable* is the user name and *value* is its password. If you do not enter *value* within the arguments, the command requests interactively to enter a value.

Argument **-c** creates the variable in clear form. Argument **-ec** creates the variable in encrypted form. Passwords are encrypted by default also if you use **-c**.

-d | -fd
Deletes (**-d**) or forces deletion (**-fd**) of a file, section, or variable (password). You can use the following wildcards:

***** Replaces one or more alphanumeric characters.

? Replaces one alphanumeric character.

With **-d** the command asks for confirmation before deleting. With **-fd** it deletes without asking confirmation.

When you delete all the variables in a section, the section is removed from the file. When you delete all the sections and all the variables from a file, the file is removed.

file The name of the file used as a namespace for *variable*. If you do not specify *file*, the command uses the default file `jm_variables` in path *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files` (`/TWA/TWS/ITA/cpa/config/jm_variables_files`).

All the variable namespaces go in path *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files` (`/TWA/TWS/ITA/cpa/config/jm_variables_files`).

section The name of the section within *file* where *variable* is defined. When *variable* is used for a password, it must be placed in a section named `password`. No section name is required to store variables.

value The value for *variable*.

variable
Can be a variable name or a user identification. If it is used for identification, it must be placed in a section named `password` within the namespace file.

Comments

To display a variable or password, a namespace file, or a section, use the command as follows:

```
param [file.section.|file..|section.] variable
```

where you can use the * and ? wildcards described for the deletion command.

The namespace files, including default `jm_variables`, have no extension.

Variable names are case sensitive.

On IBM i systems, if you use the QP2TERM and the QSH shells, passwords are made visible during the creation process with `param` and are displayed clearly in the shell logs. To guarantee the obfuscation of a password, you need to use the AIXTERM or XTERM shells.

Examples

The command:

```
param -c compassets.hardware.platform1 unix
```

defines variable `platform1` with value `unix` in section `hardware` of the new or existing file named `compassets`. The value is not encrypted.

The command:

```
param -c compassets..platform1 unix
```

defines variable `platform1` with value `unix` in the new or existing file named `compassets`. The value is not encrypted.

The command:

```
param -ec hardware.platform1 unix
```

defines variable `platform1` with value `unix` in section `hardware` in the default file *agent_installation_path*`\TWA\TWS\ITA\cpa\config\jm_variables_files\jm_variables`. The value is encrypted.

The command:

```
param -c compassets.password.jladams san07rew
```

defines variable `jladams` with value `san07rew` in section `password` of the new or existing file named `compassets`. Since `jladams` is defined in section `password`, it is interpreted as a username. The value `san07rew` is encrypted by default since it is interpreted as a password.

The command:

```
param *.*.platform1
```

lists variable `platform1` in all its defined locations. That is:

```
...\TWA\TWS\ITA\cpa\config\jm_variables_files\compassets.hardware.platform1=unix
...\TWA\TWS\ITA\cpa\config\jm_variables_files\compassets..platform1=unix
...\TWA\TWS\ITA\cpa\config\jm_variables_files\jm_variables.hardware.platform1=***
```

The command:

```
param password.*adam*
```

lists all variables including the string `adam` contained in the `password` section of all files. In this case:

```
...\TWA\TWS\ITA\cpa\config\jm_variables_files\compassets.password.jladams=********
```

The command:

```
param -d compassets.password.jladams
```

deletes variable `jladams`.

The command:

```
param -d compassets.password.*
```

deletes all the variables found in section `password` and therefore removes this section from file `compassets`.

The command:

```
param -d compassets.*.*
```

deletes all the contents (variables and sections containing variables) found in file `compassets` and therefore removes the file.

resource

Use the **resource** command to create, modify, associate, query, or set resources online or offline.

By correctly configuring the **`CLIConfig.properties`** file on the agent, you can run this command also from any connected Tivoli Workload Scheduler agent. See "Using the resource command from an agent" on page 488 for details.

Syntax

resource ?

resource [**-usr** *user_name* **-pwd** *password*]
{
[**-create**{ **-logical** *name* **-type** *type*[**-quantity** *quantity*][**-offline**] |
-group *name*[**-offline**]}]
|
[**-delete**{**-logical** *name* |
-group *name* }]
|
[**-update**{**-computer** *name*{[**-setOnline** | **-setOffline**]} |
-logical *name*
[**-setName** *name*]
[**-setType** *type*]
[**-setQuantity** *quantity*]
[**-setOnline** | **-setOffline**]
[**-addComputer** *name* |
-addComputerByID *ID* |
-removeComputer *name* |
-removeComputerByID *ID*]
|
-group *name*
[**-setName** *name*]
[**-setOnline** | **-setOffline**]
[**-addComputer** *name* |
-addComputerByID *ID* |
-removeComputer *name* |

-removeComputerByID *ID* |
-addLogical *name* |
-removeLogical *name*]}]
|
[**-query**{**-computer** *name* [*-v*] |
-logical *name* [*-v*] |
-group *name* [*-v*]}
[**-configFile** *configuration_file*]
}

Description

Use this command to work with computers, logical resources, and resource groups. In particular it is possible to:

- Create, update, list, and delete logical resources or groups
- Create logical resources, associate them to computers, define groups of logical resources or computers, and set them online or offline
- Retrieve and update resource properties using the query and the update options
- Discover the list of computers associated to a logical resource performing a detailed query on the logical resource
- Change the association between computers and logical resources
- Set resources online or offline and query computer properties

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).

`-create -logical` *`name`* **`-type`** *`type`*
  Creates the logical resource with the specified name and type. It is also possible to set a specific quantity or set the resource offline by using optional parameters in the following way:

  -create -logical *name* **-type** *type***-quantity** *quantity* **-offline**

`-create -group` *`name`*
  Creates the resource group with the specified name. It is also possible to set it offline by using the `-offline` optional parameter in the following way:

  -create -group *name* **-offline**

`-delete -logical` *`name`*
  Deletes the logical resource with the specified name.

`-delete -group` *`name`*
  Deletes the resource group with the specified name.

-update -computer *name*
Updates the computer system with the specified name. You can set the computer online or offline as follows:

-update -computer *name* **-setOnline**
Sets the specified computer online.

-update -computer *name* **-setOffline**
Sets the specified computer offline.

-update -logical *name*
Updates the specified logical resource. You can update the properties and status of a resource in the following ways:

-update -logical *name* **-setName** *name*
Updates the name of the specified logical resource.

-update -logical *name* **-setType** *type*
Updates the type of the specified logical resource.

-update -logical *name* **-setQuantity** *quantity*
Updates the quantity of the specified logical resource.

-update -logical *name* **-setOnline**
Sets online the specified logical resource.

-update -logical *name* **-setOffline**
Sets offline the specified logical resource.

You can change the association between a logical resource and a computer in the following ways:

-update -logical *name* **-addComputer** *name*
Associates the specified logical resource to the computer with the specified name.

-update -logical *name* **-addComputerByID** *ID*
Associates the specified logical resource to the computer with the specified ID.

-update -logical *name* **-removeComputer** *name*
Removes the association between the specified logical resource and the computer with the specified name.

-update -logical *name* **-removeComputerByID** *ID*
Removes the association between the specified logical resource and the computer with the specified ID.

-update -group *name*
Updates the specified resource group. You can update the properties and status of a resource group in the following ways:

-update -group *name* **-setName** *name*
Updates the name of the specified resource group.

-update -group *name* **-setOnline**
Sets online the specified resource group.

-update -group *name* **-setOffline**
Sets offline the specified resource group.

You can add and remove logical resources or computers to and from a resource group in the following ways:

-update -group *name* **-addLogical** *name*
: Adds the logical resource with the specified name to the resource group.

-update -group *name* **-removeLogical** *name*
: Removes the logical resource with the specified name from the resource group.

-update -group *name* **-addComputer** *name*
: Adds the computer with the specified name to the resource group.

-update -group *name* **-addComputerByID** *ID*
: Adds the computer with the specified ID to the resource group.

-update -group *name* **-removeComputer** *name*
: Removes the computer with the specified name from the resource group.

-update -group *name* **-removeComputerByID** *ID*
: Removes the computer with the specified ID from the resource group.

-query -computer *name*

Retrieves the following properties of the specified computer:

- Name
- Computer ID
- Operating system name
- Operating system type
- Operating system version
- Status
- Availability status

Retrieves the following additional properties if you add the `-v` option:

- Physical memory
- Virtual memory
- CPU utilization
- Free physical memory
- Free virtual memory
- Free swap space
- Allocated physical memory
- Allocated virtual memory
- Allocated swap space
- Processors number
- Allocated processors number
- Processor type
- Processor speed
- Manufacturer
- Model
- Serial number

You can use the asterisk (*) as a wildcard character in the following ways:

As a single parameter

You must enclose it between double quotes, for example:

```
C:\IBM\TWA\TDWB\bin>resource –query –computer "*"
```

This command returns a list of all existing computers.

To complete a computer name
You must enclose the entire name between double quotes, for example:

```
C:\IBM\TWA\TDWB\bin> resource –query –computer "lab123*"
```

This command returns a list of all existing computers with a name starting with `lab123`.

-query -logical *name*
Retrieves the name and the type of the specified logical resource. Retrieves the following additional properties if you add the `-v` option:

- Status
- Quantity
- Current allocation
- Computers list

You can use the asterisk (*) as a wildcard character in the following ways:

As a single parameter
You must enclose it between double quotes, for example:

```
C:\IBM\TWA\TDWB\bin>resource –query –logical "*"
```

This command returns a list of all existing logical resources.

To complete a resource name
You must enclose the entire name between double quotes, for example:

```
C:\IBM\TWA\TDWB\bin> resource –query –logical "myRes*"
```

This command returns a list of all existing logical resources with a name starting with `myRes`.

-query -group *name*
Retrieves the name and the status of the specified resource group. Retrieves the list of computers and of logical resources contained in the resource group if you use the `–v` option.

You can use the asterisk (*) as a wildcard character in the following ways:

As a single parameter
You must enclose it between double quotes, for example:

```
C:\IBM\TWA\TDWB\bin>resource –query –group "*"
```

This command returns a list of all existing resource groups.

To complete a resource group name
You must enclose the entire name between double quotes, for example:

```
C:\IBM\TWA\TDWB\bin> resource –query –group "myResGrou*"
```

This command returns a list of all existing resource groups with a name starting with `myResGrou`.

-configFile *configuration_file*
Specifies the name and the path of a custom configuration file. This keyword is optional. If you do not specify it, the default configuration file is assumed. For more information on the configuration file, see the section about the **CLIConfig.properties**.

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 the password when you type the command. For more information on the `CLIConfig.properties` file, see the section about the **`CLIConfig.properties`**.

Return Values

The **`resource`** command returns one of the following values:

0	Indicates that the command completed successfully.
< > 0	Indicates that the command failed.

Examples

- To create a logical resource named `myApplication`, of type Applications, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -create -logical myApplication
  -type Applications
  ```

  The following output is displayed:

  ```
  AWKCLI153I Logical resource "myApplication" created.
  ```
- To update the quantity of the logical resource named `myApplication`, type the following command:

  ```
  resource.bat -update -logical myApplication -setQuantity 5
  -usr john -pwd BXVFDCGS
  ```

  The following output is displayed:

  ```
  AWKCLI165I Logical resource "myApplication" updated.
  ```
- To add the relationship between a logical resource and a computer, type the following command:

  ```
  resource.bat -update -logical myApplication -addComputer myComputer
  -usr john -pwd BXVFDCGS
  ```

  The following output is displayed:

  ```
  AWKCLI165I Logical resource "myApplication" updated.
  ```
- To retrieve details of a logical resource named `myApplication`, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -query -logical myApplication –v
  ```

  The following output is displayed:

  ```
  AWKCLI171I Calling the resource repository to perform a query on resources.

  AWKCLI172I "1" logical resources were found for your query.
  Details are as follows:

  Resource Name:myApplication
  Resource Type:Applications
  Resource Status:Online
  Resource Quantity:5
  Resource Current Allocation:0
  Computers List:
          Computer Name:myComputer
          Computer ID:D656470E8D76409F9F4FDEB9D764FF59
          Computer Status:Online
          Computer Availability Status:Unavailable
  ```

- To set the logical resource named `myApplication` offline, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -update -logical myApplication
  -setOffline
  ```

  The following output is displayed:

  ```
  AWKCLI165I Logical resource "myApplication" updated.
  ```
- To set the computer named `myComputer` offline, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -update -computer myComputer
  -setOffline
  ```

  The following output is displayed:

  ```
  AWKCLI165I Computer "myComputer" updated.
  ```
- To retrieve basic properties of the computer named `myComputer`, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -query -computer myComputer
  ```

  The following output is displayed:

```
AWKCLI171I Calling the resource repository to perform a query on resources.
AWKCLI174I "1" computers were found for your query.
Details are as follows:


Computer Name: myComputer
Computer ID:D656470E8D76409F9F4FDEB9D764FF59
Computer OS Name: Microsoft Windows XP Professional English (United States) version
Computer OS Type:Windows XP
Computer OS Version:5
Computer Status:Offline
Computer Availability Status:Unavailable
```

- To retrieve detailed properties of the computer named `myComputer`, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -query -computer myComputer -v
  ```

  The following output is displayed:

```
AWKCLI171I Calling the resource repository to perform a query on resources.
AWKCLI174I "1" computers were found for your query.
Details are as follows:


Computer Name: myComputer
Computer ID:D656470E8D76409F9F4FDEB9D764FF59
Computer OS Name:Microsoft Windows XP Professional English (United States) version
Computer OS Type:Windows XP
Computer OS Version:5
Computer Status:Offline
Computer Availability Status:Unavailable
Computer details:
        Physic memory = 2095536.0
        Virtual memory = 3513788.0
        Cpu utilization = 16.0
        Free physic memory = 947972.0
        Free virtual memory = 2333484.0
        Free swap space = 52.0
        Allocated physic memory = 0.0
        Allocated virtual memory = 0.0
        Allocated swap space = 0.0
        Processors number = 1.0
        Allocated processors number = 0.0
        Processor type = x86
```

```
        Processor speed = 1995.00
        Manufacturer = IBM
        Model = 2668F8G
        Serial number = L3WZYNC
```

- To retrieve detailed properties of the logical resource named `geneva`, including the list of associated computers, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -query -logical geneva -v
  ```

  The following output is displayed:

```
Setting CLI environment variables....
AWKCLI171I Calling the resource repository to perform a query on resources.
AWKCLI172I "1" logical resources were found for your query.
Details are as follows:


Resource Name:geneva
Resource Type:prod_wks
Resource Status:Online
Resource Quantity:1
Resource Current Allocation:0
Computers List:
    Computer Name:bd_ff139_1
    Computer ID:666AADE61CBA11E0ACBECD0E6F3527DE
    Computer Status:Online
    Computer Availability Status:Available
    AWKCLI171I Calling the resource repository to perform a query on resources.
```

- To create a resource group named `myGroup`, type the following command:

  ```
  resource.bat -usr john -pwd BXVFDCGS -create -group myGroup
  ```

  The following output is displayed:

  ```
  AWKCLI153I Resource group  "myGroup" created.
  ```

- To retrieve basic properties of a resource group named `myGroup`, type the following command:

  ```
  resource.bat -query -group myGroup
  ```

  The following output is displayed:

  ```
  Setting CLI environment variables....
    AWKCLI171I Calling the resource repository to perform a query on resources.
    AWKCLI173I "1" groups were found for your query.
    Details are as follows:


    Group Name:myGroup
    Group Status:Online
  ```

- To add the computer named `myComputer` to a resource group named `myGroup`, type the following command:

  ```
  resource.bat -update -group myGroup  -addComputer myComputer
  ```

  The following output is displayed:

  ```
  Setting CLI environment variables....
    AWKCLI165I Resource Group  "myGroup" updated.
  ```

- To retrieve details of a resource group named `myGroup`, type the following command:

  ```
  resource.bat -query -group myGroup -v
  ```

  The following output is displayed:

```
Setting CLI environment variables....
  AWKCLI171I Calling the resource repository to perform a query on resources.
  AWKCLI173I "1" groups were found for your query.
  Details are as follows:


  Group Name:myGroup
  Group Status:Online
  Computers List:
   Computer Name:myComputer
           Computer ID:D656470E8D76409F9F4FDEB9D764FF59
           Computer Status:Online
   Computer Availability Status:Unavailable

  Resources List:
```

Using the resource command from an agent

You can create and manage resources and groups of resources and computers from Tivoli Workload Scheduler agents other then on the master domain manager.

Enabling the resource command

To enable this feature you must:

1. Add the runtime for Java jobs when installing the agent. See information on how to install the agent in the *Planning and Installation* manual.
2. Configure the **`CLIConfig.properties`** file. See the section about the **`CLIConfig.properties`**.
3. Run the **resource** command. See "Running the resource command" on page 489.

For this purpose an additional instance of the **`CLIConfig.properties`** file is installed on every agent. If you intend to run the **`resource`** command from an agent, you must configure the **`CLIConfig.properties`** locally.

Configuring the local CLIConfig.properties file

When you install the agent, a local copy of **`CLIConfig.properties`** is automatically installed and partially configured on your agent in the following path:

```
TWA_home/TWS/TDWB_CLI/config
```

To run the **`resource.bat`** or **`resource.sh`** command from the agent, customize the following keywords of the local **`CLIConfig.properties`** file:

ITDWBServerHost
: Specify the IP address or the hostname of the master domain manager.

ITDWBServerPort
: Specify the number of the WebSphere Application Server HTTP port.

ITDWBServerSecurePort
: Specify the number of the WebSphere Application Server HTTPS port.

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

tdwb_pwd
: Specify 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*.

Running the resource command

To run the command, enter:

On Windows:
```
resource.bat
```
On UNIX:
```
resource.sh
```

twstrace

Changes the trace level on the dynamic agent without having to stop and restart the agent.

Authorization

You must login with the credentials of the user which installed the dynamic agent. You can also use any authorization higher than the user which installed the dynamic agent.

Syntax

twstrace -u | -V -enable | -disable [**-level** *value*] [**-maxFiles***files_number*] [**-maxFileBytes** *bytes_number*] [**-getLogs** [**-zipFile** *zip_file_name*] [**-host***hostname*] [**-protocol** *{http|https}*] [**-port** *port number*] [**-inifile** *ini_filename*]]

Arguments

enable
: Enables tracing to the maximum level. The maximum level is **3000**. By default, traces are disabled.

disable
: Disables tracing.

level *value*
: The level of detail for the traces:

1000	Error, warning, and informational messages are traced.
2000	Error and warning messages are traced.
3000	Error messages are traced.

maxFiles *files_number*
: The maximum number of the trace files you want to create.

maxFileBytes *bytes_number*
: The maximum size in bytes that the trace file can reach. The default is **1024000** bytes.

getLogs
: To collect the trace files, the message files and the configuration files in a zip file.

zipfile *zip_file_name*
: The name of the zip file that contains all the information, that is logs, traces, and configuration files (`ita.ini` and `jobManager.ini`) for the agent. The default is `logs.zip`.

host *hostname*
: The host name or the IP address of the agent for which you want to collect the traces. The default is **localhost**.

protocol *http|https*
: The protocol of the agent for which you are collecting the traces. The default is the protocol specified in the **.ini** file of the agent.

port *port number*
: The port of the agent. The default is the port number of the agent where you are running the command line.

inifile *ini_filename*
: The name of the **.ini** file that contains the SSL configuration of the agent for which you want to collect the traces. The default is the **.ini** file of the local agent. If you are collecting the tracing for a remote agent for which you customized the security certificates, you must import the certificate on the local agent and specify the name of the **.ini** file that contains the configuration. To do this, perform the following actions:

1. Extract the certificate from the keystore of the remote agent.
2. Import the certificate in a local agent keystore. You can create an ad hoc keystore whose name must be **TWSClientKeyStore.kdb**.
3. Create a **.ini** file in which you specify:
   - **0** in the **tcp_port** property as follows:
     ```
     tcp_port=0
     ```
   - The port of the remote agent in the **ssl_port** property as follows:
     ```
     ssl_port=<ssl_port>
     ```
   - The path to the keystore you created in step 2. in the **key_repository_path** property as follows:
     ```
     key_repository_path=<local_agent_keystore_path>
     ```

u Displays the command usage.

V Displays the version of the product.

Examples

To set the trace level to record error and warning messages, run the following command:

```
 twstrace -enable -level 2000
```

To retrieve the information about the trace level, run the following command:

```
 twstrace -level -maxFiles -maxFileBytes

 AWSITA1761 The trace properties are: level="1000",
maxFiles="3", file size="1024000"
```

Chapter 15. Getting reports and statistics

This chapter describes the report commands that you use to get summary or detailed information about the previous or next production plan. These commands are run from the operating system command prompt on the master domain manager. The chapter is divided into the following sections:

- "Setup for using report commands"
- "Command descriptions" on page 492
- "Sample report outputs" on page 499
- "Report extract programs" on page 509
- "Running Dynamic Workload Console reports and batch reports" on page 520
- "Running batch reports from the command line interface" on page 525

Setup for using report commands

To configure the environment for using report commands set the *PATH* and *TWS_TISDIR* variables by running one of the following scripts:

- `. ./`*`TWS_home`*`/`**`tws_env.sh`** for Bourne and Korn shells in UNIX
- `. ./`*`TWS_home`*`/`**`tws_env.csh`** for C shells in UNIX
- *`TWS_home`*`\`**`tws_env.cmd`** in Windows

The report commands must be run from the *TWS_home* directory.

The output of the report commands is controlled by the following environment variables:

MAESTROLP
: Specifies the destination of the output of a command. The default is **stdout**. You can set it to any of the following:

  filename
  : Writes the output to a file.

  > *filename*
  : UNIX only. Redirects output to a file, overwriting the contents of the file. If the file does not exist it is created.

  >> *filename*
  : UNIX only. Redirects output to a file, appending to the end of the file. If the file does not exist it is created.

  | *command*
  : UNIX only. Pipes output to a system command or process. The system command is always run.

  || *command*
  : UNIX only. Pipes output to a system command or process. The system command is not run if there is no output.

MAESTRO_OUTPUT_STYLE
: | Specifies the output style for long object names. With value **LONG**, full
  | length (long) fields are used for object names. This is the default.

| If the variable is set to anything other than **LONG**, long names are
| truncated to eight characters and a plus sign. For example: **A1234567+**.

You should use a fixed font size to obtain the correct format of the reports outputs.

Changing the date format

In Tivoli Workload Scheduler, the date format affects all commands that accept a date as an input option (except the **datecalc** command), and the headers in all reports. The default date format is *mm/dd/yy*. To select a different format, edit the ***date format*** local option store in the `localopts` file. The values are:

Table 65. Date formats

date format **value**	**Corresponding date format output**
0	yy/mm/dd
1	mm/dd/yy
2	dd/mm/yy
3	Native language support variables.

See the *IBM Tivoli Workload Scheduler Administration Guide* for details on modifying local variables in the `localopts` file.

Command descriptions

Tivoli Workload Scheduler report commands are listed in Table 66:

Table 66. List of report commands

Command	Description
rep1	Report 01 - Job Details Listing
rep2	Report 02 - Prompt Listing
rep3	Report 03 - Calendar Listing
rep4a	Report 04A - Parameter Listing
rep4b	Report 04B - Resource Listing
rep7	Report 07 - Job History Listing
rep8	Report 08 - Job Histogram
rep11	Report 11 - Planned Production Schedule
reptr	Report 09A - Planned Production Summary Report 09B - Planned Production Detail Report 09D - Planned Production Detail (Long Names) Report 10A - Actual Production Summary Report 10B - Actual Production Detail
xref	Report 12 - Cross Reference Report

rep1 - rep4b

These commands print the following reports:

Report 01
Job Details Listing

Report 02
: Prompt Listing

Report 03
: Calendar Listing

Report 04A
: Parameters Listing

Report 04B
: Resource Listing

Syntax

rep[*x*] [**-V**|**-U**]

Run the command from the *TWS_home* directory.

For `rep3`, run the command from a directory to which you have `write` access.

When printing reports for job types with advanced options, the JCL file field returns the application name.

Arguments

x	A number corresponding to the report. The numbers are: **1**, **2**, **3**, **4a**, or **4b**.
-U	Displays the command usage information and exits.
-V	Displays the command version and exits.

Comments

The Job Details Listing (report 01) cannot include jobs that were submitted using an alias name.

The elapsed time displayed for a shadow job is the elapsed time of the remote job to which it is bound.

Examples

Print Report 03, User Calendar Listing:

```
rep3
```

Display usage information for the **rep2** command:

```
rep2 -U
```

In UNIX, print two copies of report 04A, User Parameters Listing, on printer `lp2`:

```
MAESTROLP="| lp -dlp2 -n2"
export MAESTROLP
rep4a
```

This is a sample report for job WAGES2_1:

```
Job: WAGES2_1      #FTP                                    Description:
JCL File         : filetransfer
Logon            :                                         Creator: tws86
Recovery Job     :
Recovery Type    : STOP
Recovery Prompt  :
Composer Autodoc : Yes
```

```
Total Runs       :     0 -     0     Successful,       0 Aborted

                  Elapsed(secs)        CPU(secs)
Total             0                    0
Normal            0
Last Run          0                    0 (On       0 at     0)
Maximum           0                    0 (On       0)
Minimum           0                    0 (On       0)>
```

rep7

This command prints Report 07-Job History Listing.

Syntax

rep7 -V|-U

rep7
  [**-c** *wkstat*]
  [**-s** *jstream_name*]
  [**-j** *job*]
  [**-f** *date*]
  [**-t** *date*]
  [**-l**]

Run the command from the *TWS_home* directory.

Arguments

-U Displays the command usage information and exits.

-V Displays the command version and exits.

-c *wkstat*
Specifies the name of the workstation on which the jobs run. The default is all workstations.

-s *jstream_name*
Specifies the name of the job stream in which the jobs run. The default is all job streams.

-j *job* Specifies the name of the job. The default is all jobs.

-f *date* Specifies to print job history from this date forward. Enter the date as *yyyymmdd*. The default is the earliest available date.

-t *date* Specifies to print job history up to this date. Enter the date as *yyyymmdd*. The default is the most recent date.

-l Limits the summary line information to the jobs which fall in the date range specified by the -f or -t options. Using this option causes the order of output to be reversed: the job summary line will be printed after the individual job run lines. This option is valid only if you also specify at least one of the -f or -t options.

Comments

The elapsed time displayed for a shadow job is the elapsed time of the remote job to which it is bound.

Any time you run **rep7** the output of the command contains the information stored until the last time you run **JnextPlan**, the information related to the run of the current production plan will be contained in the **rep7** output the next time you run **JnextPlan**. For this reason if you run **rep7** after having generated the production plan for the first time or after a **ResetPlan** command, the output of the command contains no statistic information.

Examples

Print all job history for workstation `ux3`:

```
rep7 -c ux3
```

Print all job history for all jobs in job stream `sked25`:

```
rep7 -s sked25
```

Print job history for all jobs in job stream `mysked` on workstation `x15` between 1/21/2005 and 1/25/2005:

```
rep7 -c x15 -s mysked -f 20050121 -t 20050125
```

rep8

This command prints Report 08-Job Histogram.

Syntax

rep8 -V|-U

rep8
  [**-f** *date* **-b** *time* **-t** *date* **-e** *time*]
  [**-i** *file*]
  [**-p**]

rep8
  [**-b** *time* **-e** *time*]
  [**-i** *file*]
  [**-p**]

Run the command from the *TWS_home* directory.

Arguments

-U Displays the command usage information and exits.

-V Displays the command version and exits.

-f *date* Specifies to print job history from this date forward. Enter the date as *yyyymmdd*. The default is today's date.

-b *time*
Specifies to print job history from this time forward. Enter the time as *hhmm*. The default is the Tivoli Workload Scheduler *startOfDay*.

-t *date* Specifies to print job history up to this date. Enter the date as *yyyymmdd*. The default is the most recent date.

-e *time* Specifies to print job history up to this time. Enter the time as *hhmm*. The default is the Tivoli Workload Scheduler start of day time.

-i *file* Specifies the name of the log file from which job history is extracted. Note that log files are stored in the `schedlog` directory. The default is the current `Symphony` file.

Note: Ensure that the time range specified by the [**-f** *date* **-b** *time* **-t** *date* **-e** *time*] arguments is within the date and time range defined in the **-i** *file* log file name.

-p Specifies to insert a page break after each run date.

Comments

Any time you run **rep8** the output of the command contains the information stored until the last time you run **JnextPlan**, the information related to the run of the current production plan will be contained in the **rep8** output the next time you run **JnextPlan**. For this reason if you run **rep8** after having generated the production plan for the first time or after a **ResetPlan** command, the output of the command contains no statistic information.

Examples

Print a job histogram which includes all information in the current plan (`Symphony` file):

```
rep8
```

Print a job histogram beginning at 6:00 a.m. on 1/25/2005, and ending at 5:59 a.m. on 1/26/2005:

```
rep8 -f 20050125 -b 0600 -t 20050126 -e 0559 -i schedlog/M199801260601
```

Print a job histogram, from the current plan (`Symphony` file), beginning at 6:00 am, and ending at 10:00 pm:

```
rep8 -b 0600 -e 2200
```

rep11

This command prints Report 11-Planned Production Schedule.

Syntax

rep11 -V|-U

rep11
[**-m** *mm*[*yy*] [...]]
[**-c** *wkstat* [...]]
[**-s** *jstream_name*]
[**-o** *output*]

Run the command from the *TWS_home* directory.

Arguments

-U Displays the command usage information and exits.

-V Displays the command version and exits.

-m *mm*[*yy*]
Specifies the months to be reported. Enter the month number as *mm*. The default is the current month.

You can also enter a year as *yy*. The default is the current year or next year if you specify a month earlier than the current month.

-c *wkstat*
Specifies the workstations to be reported. The default is all workstations.

-s *jstream_name*
Specifies the name of the job stream in which the jobs run. The default is all job streams.

-o *output*
Specifies the output file. The default is the file defined by the ***MAESTROLP*** variable. If ***MAESTROLP*** is not set, the default is **stdout**.

Examples

Report on June, July, and August of 2004 for workstations `main`, `site1` and `sagent1`:

```
rep11 -m 062004 072004 082004 -c main site1 sagent1
```

Report on June, July, and August of this year for all workstations, and direct output to the file `r11out`:

```
rep11 -m 06 07 08 -o r11out
```

Report on this month and year for workstation `site2`:

```
rep11 -c site2
```

reptr

This command prints the following reports:

Report 09A
Planned Production Summary

Report 09B
Planned Production Detail

Report 10A
Actual Production Summary

Report 10B
Actual Production Detail

Report 09A and Report 09B refer to future production processing while Report 10A and Report 10B show processing results and status of each single job of already processed production.

Syntax

reptr [**-V**|**-U**]

reptr -pre
[-{**summary** | **detail**}]
[*symfile*]

reptr -post
[-{**summary** | **detail**}]
[*logfile*]

Run the command from a directory to which you have `write` access.

Arguments

-U Displays the command usage information and exits.

-V Displays the command version and exits.

-pre Specifies to print the pre-production reports (09A and 09B).

-post Specifies to print the post-production reports (10A and 10B).

-summary
Specifies to print the summary reports (09A and 10A). If **-summary** and **-detail** are omitted, both sets of reports are printed.

-detail Specifies to print the detail reports (09B and 10B). If **-summary** and **-detail** are omitted, both sets of reports are printed.

symfile Specifies the name of the plan file from which reports will be printed. The default is `Symnew` in the current directory. If the file is not in the current working directory, you must add the absolute path to the file name.

logfile Specifies the full name of the log file from which the reports will be printed. Note that plan log files are stored in the `schedlog` directory. The default is the current plan (`Symphony` file).

If the command is run with no options, the two **pre** reports (09A and 09B) are printed and the information is extracted from the `Symphony` file.

Examples

Print the pre-production detail report from the `Symnew` file:

```
reptr -pre -detail
```

Print the pre-production summary report from the file `mysym`:

```
reptr -pre -summary mysym
```

Print the post-production summary report from the log file `M199903170935`:

```
reptr -post -summary schedlog/M199903170935
```

Print the pre-production reports reading from the Symphony file.

```
reptr
```

When the arguments are specified, the pre-production reports are based on information read from the `Symnew` file while the post-production reports are based on information read from the `Symphony` file.

xref

This command prints Report 12-Cross Reference Report.

Syntax

xref [**-V**|**-U**]

xref
[**-cpu** *wkstat*]
[**-depends**|**-files**|**-jobs**|**-prompts**|**-resource**|**-schedules**|**-when**[...]]

Run the command from the *TWS_home* directory.

Arguments

-U Displays the command usage information and exits.

-V Displays the command version and exits.

-cpu *wkstat*
: Specifies to print the report for the named workstation. The @ wildcard is permitted, in which case, information from all qualified workstations is included. The default is all workstations.

-depends
: Specifies to print a report showing the job streams and jobs that are successors of each job.

-files Specifies to print a report showing the job streams and jobs that are dependent on each file.

-jobs Specifies to print a report showing the job streams in which each job is run.

-prompts
: Specifies to print a report showing the job streams and jobs that are dependent on each prompt.

-resource
: Specifies to print a report showing the job streams and jobs that are dependent on each resource.

-schedules
: Specifies to print a report showing the job streams and jobs that are successors of each job stream.

-when Specifies to print a report showing job stream `Include` and `Exclude` dates.

If the command is run with no options, all workstations and all options are selected.

Examples

Print a report for all workstations, showing all cross-reference information:

```
xref
```

Print a report for all workstations. Include cross-reference information about all successor dependencies:

```
xref -cpu @ -depends -schedules
```

Sample report outputs

Report 01 - Job Details Listing:

```
TWS for UNIX (AIX)/REPORT1 8.3 (1.7)        ibm                       Page    1
Report 01                             Job Details Listing              03/06/06

Job             : FTAWIN8+        #SCHEDDDD
Description     :
JCL File        : dir
Logon           : maestro_adm
Creator         : root
Recovery Job    :
Recovery Type   : STOP
Recovery Prompt :
```

```
composer Autodoc : Yes
Total Runs       :     0 -     0     Successful,       0 Aborted

                  Elapsed(mins)      CPU(secs)
Total             00:00:00              0
Normal            00:00:00
Last Run          00:00:00              0 (On          at 00:00)
Maximum           00:00:00              0 (On         )
Minimum           00:00:00              0 (On         )


Job              : MASTER8+        #JnextPlan
Description      : ADDED BY composer FOR SCHEDULE MASTER821#FINAL.
JCL File         : /test/maestro_adm/tws/JnextPlan
Logon            : maestro_adm
Creator          : maestro_adm
Recovery Job     :
Recovery Type    : STOP
Recovery Prompt  :
composer Autodoc : Yes
Total Runs       :    11 -    11     Successful,       0 Aborted

                  Elapsed(mins)      CPU(secs)
Total             00:00:14             44
Normal            00:00:01
Last Run          00:00:01              4 (On 03/05/06 at 23:16)
Maximum           00:00:02              4 (On 03/04/06)
Minimum           00:00:01              4 (On 03/04/06)


Job              : MASTER8+        #JOB1
Description      : ADDED BY composer.
JCL File         : pwd
Logon            : ^ACCLOGIN^
Creator          : root
Recovery Job     :
Recovery Type    : STOP
Recovery Prompt  :
composer Autodoc : Yes
Total Runs       :     1 -     1     Successful,       0 Aborted

                  Elapsed(mins)      CPU(secs)
Total             00:00:01              0
Normal            00:00:01
Last Run          00:00:01              0 (On 03/05/06 at 22:22)
Maximum           00:00:01              0 (On 03/05/06)
Minimum           00:00:01              0 (On 03/05/06)

                         * * * *  E n d  o f  R e p o r t  * * * *
```

In the output you see the values set in the “Job” on page 636 as follows::

composer Autodoc
Says if the job statement was described in the job stream definition using the command line interface.

CPU (secs)
Is the actual time, expressed in seconds, the job made use of the CPU to run.

Total Is the sum of CPU time recorded for the 'Total Runs'.

Normal
Is the average value of CPU time recorded during the 'Total Runs'.

Last Run
Is the CPU time recorded during the last run of the job.

Maximum
Is the maximum among the values collected for CPU time during the 'Total Runs' (calculated only for jobs ended successfully).

Minimum
Is the minimum among the values collected for CPU time during the 'Total Runs' (calculated only for jobs ended successfully).

Creator
Is the name of the user who created the job definition.

Description
Is the textual description of the job set in the **description** field of the job definition statement.

Elapsed
Is the amount of time, expressed in minutes, that includes both the time during which the job made use of the CPU and the time the job had to wait for other processes to release the CPU.

Total Is the sum of Elapsed time recorded for the 'Total Runs'.

Normal
Is the average value of Elapsed time recorded during the 'Total Runs'.

Last Run
Is the Elapsed time recorded during the last run of the job.

Maximum
Is the maximum among the values collected for Elapsed time during the 'Total Runs' (calculated only for jobs ended successfully).

Minimum
Is the minimum among the values collected for Elapsed time during the 'Total Runs' (calculated only for jobs ended successfully).

Note: The elapsed time displayed for a shadow job is the elapsed time of the remote job to which it is bound.

JCL File
Is the name of the file set in the **scriptname** field that contains the script to run, or the command specified in the **docommand** field to invoke when running the job.

Job Is the identifier of the job, [*workstation*#]*jobname*.

Logon Is the user name, specified in the **streamlogon** field, under which the job runs.

Recovery Job
Is the job, specified as *after [workstation#]jobname*, that is run if the parent job abends.

Recovery Prompt
Is the text of the prompt, specified in the **abendprompt** field, that is displayed if this job abends.

Recovery Type
Is the recovery option set in the job definition. It can be set to **stop**, **continue**, or **rerun**.

Report 02 - Prompt Listing:

```
TWS for UNIX (AIX)/REPORT2 8.3 (1.7)         ibm                        Page    1
Report 02                              Prompt Message Listing            03/06/06


Prompt      Message

PROMPT1     Reply YES when ready to run acc103 and acc104.
PROMPT2     Have all users logged out?
CALLNO      555-0911
CALLOPER    Call ^PERSON2CALL^ at ^CALLNO^ to ensure all time cards have been processed.
PERSON2CALL Lou Armstrong

Total number of prompts on file:      5

                          * * * *  E n d  o f  R e p o r t  * * * *
```

The Report 02 output lists the name and the text of the prompts defined in the environment.

Report 03 - Calendar Listing:

```
TWS for UNIX (AIX)/REPORT3 8.3 (1.7)        ibm                         Page    1
Report 03                             User Calendar List                 03/06/06


Calendar Type: MONTHEND
Description: End of month until end of 2006.


          Jan 2006                      Feb 2006                      Mar 2006
Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat
                         .             .   .   .   .   .             .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .  28                         .   .   .   .  31
 .  31

          Apr 2006                      May 2006                      Jun 2006
Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat
                     .   .     .   .   .   .   .   .   .                 .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .  30     .   .  31                     .   .   .   .  30


          Jul 2006                      Aug 2006                      Sep 2006
Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat
                     .   .         .   .   .   .   .   .                     .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .  31                 .   .   .   .   .  30
31

          Oct 2006                      Nov 2006                      Dec 2006
Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat   Sun Mon Tue Wed Thu Fri Sat
                         .             .   .   .   .   .                     .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .   .   .   .   .     .   .   .   .   .   .   .
 .   .   .   .   .   .   .     .   .   .  30                 .   .   .   .   .   .  31
 .  31
```

```
                          * * * *  E n d  o f  R e p o r t  * * * *
```

In the output you see highlighted the end of month days selected in calendar **MONTHEND**.

Report 04A - Parameter Listing:

```
TWS for UNIX (AIX)/REPORT4A 8.3 (1.7)         ibm                          Page    1
Report 4A                              User Parameter Listing               03/06/06


Parameter Name            Contents

ACCHOME                   /usr/local/Tivoli/maestro_adm
ACCLOGIN                  maestro_adm
BADEXIT                   99
GOODEXIT                  0
SCRPATH                   /usr/local/Tivoli/maestro_adm/scripts

Number of Parameters on file:      5

                          * * * *  E n d  o f  R e p o r t  * * * *
```

The Report 04A output lists the name and the content of the parameters defined in the environment.

Report 04B - Resource Listing:

```
TWS for UNIX (AIX)/REPORT4B 8.3 (1.7)         ibm                          Page    1
Report 4B                              TWS Resources Listing                03/06/06


               Resource             Number
CPU              Name                Avail          Description

FTAHP            #DATTAPES             1            DAT tape units

FTAWIN8+         #QUKTAPES             2            Quick tape units

MASTER8+         #TAPES                2            Tape units

MASTER8+         #JOBSLOTS           1024           Job slots


Number of Resources on file:      4

                           * * * *  E n d  o f  R e p o r t  * * * *
```

The Report 04B output lists the name, the number of available resources defined in the environment and their description.

Report 07 - Job History Listing:

```
TWS for UNIX (AIX)/REPORT7 8.3   (1.13)         ibm                        Page    1
Report 07                              Job History Listing                  03/08/06

Date        Time      Job Stream Name     Elapsed       CPU     Status

Job:MASTER8+#MyJS Runs: Aborted 0  Successful 11  Elapsed Time: Normal 1 Min 1 Max 2

03/03/06    01:46     MASTER8+#JS1            1          4        SU
03/03/06    19:08     MASTER8+#JS2            1          4        SU
03/03/06    19:33     MASTER8+#JS3            1          4        SU
```

```
03/03/06    19:37     MASTER8+#JS4           1         4        SU
03/03/06    23:08     MASTER8+#JS5           2         4        SU
03/03/06    05:59     MASTER8+#JS_A          1         4        SU
03/05/06    05:59     MASTER8+#JS_G          1         4        SU
03/06/06    05:59     MASTER8+#JS_H          1         4        SU
03/06/06    21:57     MASTER8+#TIMEJ         2         4        SU
03/06/06    23:16     MASTER8+#SLEEPJ        1         4        SU

Job:MASTER8+#JOB1     Runs: Aborted 0   Successful 1  Elapsed Time: Normal 1 Min 1 Max 1

03/06/06    22:22     MASTER8+#JOBS          1         0        SU


                           * * * *  E n d  o f  R e p o r t  * * * *
```

The Report 7 reads the information about job run stored in the database and displays them. The possible states for a job are:

AB for failed jobs

SU for successfully completed jobs

DN for submitted jobs whose state is unknown because neither a successful or a failure message has been received yet.

Report 08 - Job Histogram:

```
TWS for UNIX (AIX)/REPORT8 8.3 (1.7)        ibm                        Page    1
Report 08                Job Histogram 03/05/06 14:05 - 03/06/06 14:04     03/06/05

Interval Per Column: 15 minutes

                         1   1   1   1   2   2   2   0   0   0   0   0   0   0   1   1   1
                         4   5   7   8   0   1   3   0   2   3   5   6   8   9   1   2   4
                         0   3   0   3   0   3   0   3   0   3   0   3   0   3   0   3   0
                         5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   5   4

Job Name          Stat
03/06/06

CF05066+.JnextPlan   SU                                             .*.


                          * * * *  E n d  o f  R e p o r t  * * * *
```

The output of Report 8 shows the time slots during which the jobs run. The numbers at the top of the job histogram are times, written top-down, for example the first column 1405 means 2:05PM. The time slots when the job run are marked by asterisks when the position of the marker is aligned with a time written top-down, and dots.

Report 9B - Planned Production Detail:

```
TWS for UNIX (AIX) REPORTER 8.3 (1.7)       ibm                             Page    1
Report 09B   Symnew       Planned Production Detail For 03/06/06              03/06/06

                        Estimated
                Job Name Run Time Pri Start Time     Until           Every Limit Dependencies

Schedule NETAG #EXTERNAL           0
                E0000000           0
              Total        00:00
              Total        00:00

Schedule MYFTA #IWDSKE           10                                           NETAG#EXTERNAL.E0000000
```

```
                JOBIWD            10                23:00(03/06/06) 01:00
              Total        00:00

Schedule MYMST #TESTSKE    00:29  10
                TESTCRO+   00:01  10
                NEWTEST    00:29  10 08:30(03/06/06)                             TESTCROME
              Total        00:29


Schedule MYMST #FINAL      00:00  10 05:59(03/07/06)
                JnextPlan   00:01  10
              Total        00:01
              Total        00:34

                               * * * *   E n d  o f  R e p o r t  * * * *
```

The output of Report 9B shows what is in plan to run on the selected date in the Tivoli Workload Scheduler environment. The information displayed is taken from the definitions stored in the Tivoli Workload Scheduler database. The output shows the job streams that are planned to run on the 6th of March 2006 with their description, the list of jobs they contain, the time dependencies, repetition rate, and job limit, if set, and the dependency on other jobs or job streams. For example, job stream named `iwdske` that is planned to run on `MYFTA` has a `follows` dependency on job `NETAG#EXTERNAL.E0000000` that is planned to run on the network agent named `NETAG`.

The **Start Time** field in the output of the reports generated by the **reptr** command shows:

A time restriction set in the job stream definition using the at keyword.
If the date is enclosed in parenthesis (), for example:

```
Start Time
06:00(03/20/06)
```

The time the job stream is planned to run set in the job stream definition using the schedtime keyword.
If the date is enclosed in braces {}, for example:

```
Start Time
06:00{03/20/06}
```

The time the job stream actually started to run.
If the date is not enclosed either in braces or in parenthesis, for example:

```
Start Time
06:00 03/20/06
```

Report 10B - Actual Production Detail:

```
TWS for UNIX (AIX) REPORTER 8.3 (1.7)          ibm                          Page    1
Report 10B   Symphony           Actual Production Detail For 03/06/06          03/07/06

                         Estimated                         Actual    CPU    Job
                Job Name  Run Time Priority Start Time    Run Time Seconds Number  Status

Schedule NETAG #EXTERNAL             0                                            EXTRN
               E0000000              0                                            ERROR
                Total      00:00                            00:00    0

Schedule MYMST #MONTHSKE    00:02    10     06:01(03/06/06)  00:03                 SUCC
                GETLOGS     00:02    10     06:01(03/06/06)  00:03         #J11612   SUCC
                Total      00:02                            00:03    0

Schedule MYFTA #IWSKE              10                                             HOLD
                JOBIWD             10                                             HOLD
```

```
                 Total      00:00                            00:00    0

Schedule MYMST #TESTSKE     00:29    10     06:01(03/06/06)  00:02                STUCK
                TESTCRO+    00:01    10     06:01(03/06/06)  00:02        #J11613  ABEND
                NEWTEST     00:29    10                                            HOLD
                 Total      00:30                            00:02    0

Schedule MYMST #FINAL       00:01    10     05:59(03/07/06)                        HOLD
                JnextPlan    00:01    10                                            HOLD
                 Total      00:01                            00:00    0


                 Total      01:38                            00:09    0
                                 * * * *  E n d  o f  R e p o r t  * * * *
```

The output of Report 10B shows states of the scheduling activities currently running across the Tivoli Workload Scheduler network. The information displayed is taken from copy of the Symphony file that is currently used and updated across the scheduling environment. This means that anytime this report command is run during the processing the information displayed reflects the actual status of the planned activity.

If you compare this output with the output of Report 9B you see that job stream MONTHSKE has run during the current production day, the 6th of March, but is not in plan to run the next day, the 7th of March. The job stream EXTERNAL instead failed on the network agent NETAG and so the IWSKE job stream that has a follows dependency from EXTERNAL job stream remains in the HOLD state.

The job stream TESTSKE, instead, is in state STUCK, that means that operator intervention is needed, because within the job stream run time, job TESTCROME, after having started with job ID J11613, failed in ABEND state causing the depending job NEWTEST to turn into HOLD state.

Report 11 - Planned Production Schedule:

```
TWS for UNIX (AIX)/REPORT11 8.3 (1.7)                                      Page    1
Report 11                Planned Production Schedule for FEB 2006            03/08/065

CPU: FTAWIN8+

         Num  Est Cpu  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
Schedule Jobs  Time   Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo
SCHED1    1       1                              *

An * between Schedule name and Num Jobs indicates that the schedule has jobs running on other cpus.
----------------------------------------------------------------------------------------------------
Estimated Cpu Time Per Day in Seconds
     Mon     Tue     Wed     Thu     Fri     Sat     Sun

               1       2       3       4       5       6
               0       0       0       0       0       0

       7       8       9      10      11      12      13
       0       0       0       0       1       0       0

      14      15      16      17      18      19      20
       0       0       0       0       0       0       0

      21      22      23      24      25      26      27
       0       0       0       0       0       0       0

      28
      0


TWS for UNIX (AIX)/REPORT11 8.3 (1.7)                               Page    2
Report 11                        Planned Production Schedule for FEB 2006      03/08/06

CPU: MASTER8+

         Num  Est Cpu  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
Schedule Jobs  Time   Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo
FINAL     1       4    *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *

An * between Schedule name and Num Jobs indicates that the schedule has jobs running on other cpus.
----------------------------------------------------------------------------------------------------
Estimated Cpu Time Per Day in Seconds
     Mon     Tue     Wed     Thu     Fri     Sat     Sun

               1       2       3       4       5       6
               4       4       4       4       4       4

       7       8       9      10      11      12      13
       4       4       4       4       4       4       4

      14      15      16      17      18      19      20
       4       4       4       4       4       4       4

      21      22      23      24      25      26      27
       4       4       4       4       4       4       4

      28
       4


                          * * * *  E n d  o f  R e p o r t  * * * *
```

The output of Report 11 shows when the job streams are planned to run during the selected month. In the first line it is displayed the number of jobs the job stream contains, the estimated CPU time used by the job stream to run, and when the job stream is planned to run. In the matrix it is displayed for each day of the selected month the estimated CPU time used by that job stream to run.

Report 12 - Cross Reference Report:

The output of Report 12 shows different information according to the flag used when issuing the `xref` command. In this section you find some samples of output. For each of these sample the corresponding flag used with the xref command is highlighted.

xref -when

```
TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)         ibm                            Page    1
Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/06

CPU: FTAHP

WHEN                  Used by the following schedules:
REQUEST               TRFINAL



TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)         ibm                            Page    2
Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/06

CPU: FTAWIN8+
WHEN                  Used by the following schedules:
MONTHEND              SCHED1
REQUEST               SCHED1  , SCHEDDAA



TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)         ibm                           Page    3
Report 12 Cross Reference Report for the ON, EXCEPT(*) and FREEDAYS(f) options. 03/08/06

CPU: MASTER8+

WHEN                  Used by the following schedules:
EVERYDAY              FINAL
REQUEST               TMP


                         * * * *  E n d  o f  R e p o r t * * * *
```

xref -jobs

```
TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)         ibm                          Page    4
Report 12                    Cross Reference Report for Job Names.          03/08/06

CPU: FTAWIN8+

Job Name              Exists in Schedules
SCHEDDDD              SCHED1



TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)         ibm                          Page    5
Report 12                    Cross Reference Report for Job Names.          03/08/06

CPU: MASTER8+

Job Name              Exists in Schedules
JnextPlan              FINAL
JOB1                  TMP


                         * * * *  E n d  o f  R e p o r t * * * *
```

xref -resource

```
TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)         ibm                          Page    8
Report 12                   Cross Reference Report for Resource Users.      03/08/06

CPU: FTAWIN8+

Resource              Used by the following:
QUKTAPES(N/F)         SCHED1



TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)         ibm                          Page    9
Report 12                   Cross Reference Report for Resource Users.      03/08/06

CPU: MASTER8+

Resource              Used by the following:
TAPES(N/F)            TMP


                         * * * *  E n d  o f  R e p o r t * * * *
```

xref -prompts

```
TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)        ibm                             Page    6
Report 12                Cross Reference Report for Prompt Dependencies.     03/08/06
CPU: FTAWIN8+

Prompt               Used by the following:

User defined text     SCHED1



TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)        ibm                             Page    7
Report 12                Cross Reference Report for Prompt Dependencies.     03/08/06

CPU: MASTER8+

Prompt               Used by the following:
BADEXIT              FTAWIN8+#SCHED1
GOODEXIT             FTAWIN8+#SCHED1  , TMP

User defined text     TMP


                         * * * *  E n d  o f  R e p o r t  * * * *
```

xref -files

```
TWS for UNIX (AIX)/CROSSREF 8.3 (1.7)        ibm                             Page   10
Report 12                 Cross Reference Report for File Dependencies.      03/08/06

CPU: MASTER8+

File Name            Used by the following:
/root/MY_FILE.sh     FTAWIN8+#SCHED1  , TMP


                         * * * *  E n d  o f  R e p o r t  * * * *
```

Report extract programs

Data extraction programs are used to generate several of the Tivoli Workload Scheduler reports. The programs are listed in Table 67:

Table 67. Report extract programs.

Report extract program	Description
jbxtract	Used to generate Report 01 - Job Details Listing and for Report 07 - Job History Listing
prxtract	Used to generate Report 02 - Prompt Listing
caxtract	Used to generate Report 03 - Calendar Listing
paxtract	Used to generate Report 04A - Parameters Listing
rextract	Used to generate Report 04B - Resource Listing
r11xtr	Used to generate Report 11 - Planned Production Schedule
xrxtrct	Used to generate Report 12 - Cross Reference Report

The output of the extract programs is controlled by the ***MAESTRO_OUTPUT_STYLE*** variable, which defines how long object names are handled. For more information on the ***MAESTRO_OUTPUT_STYLE*** variable, refer to "Command descriptions" on page 492.

jbxtract

Extracts information about jobs from the database.

Syntax

jbxtract [**-V** | **-U**]
[**-j** *job*]
[**-c** *wkstat*]
[**-o** *output*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-j *job* Specifies the job for which extraction is performed. The default is all jobs.

-c *wkstat*
Specifies the workstation of jobs for which extraction is performed. The default is all workstations.

-o *output*
Specifies the output file. The default is **stdout**.

Results

| The ***MAESTRO_OUTPUT_STYLE*** variable specifies the output style for long object
| names. With value **LONG**, full length (long) fields are used for object names. This
| is the default. If the variable is set to anything other than **LONG**, long names are
| truncated to eight characters and a plus sign. For example: `A1234567+`.

Each job record contains tab-delimited, variable length fields. The fields are described Table 68.

Table 68. Jbxtract output fields

Field	**Description**	**Max Length (bytes)**
1	workstation name	16
2	job name	16
3	job script file name	4096
4	job description	65
5	recovery job name	16
6	recovery option (0=stop, 1=rerun, 2=continue)	5
7	recovery prompt text	64
8	auto-documentation flag (0=disabled, 1=enabled)	5
9	job login user name	36
10	job creator user name	36
11	number of successful runs	5
12	number of abended runs	5
13	total elapsed time of all job runs	8
14	total cpu time of all job runs	8
15	average elapsed time	8

Table 68. Jbxtract output fields (continued)

Field	Description	Max Length (bytes)
16	last run date (yymmdd)	8
17	last run time (hhmm)	8
18	last cpu seconds	8
19	last elapsed time	8
20	maximum cpu seconds	8
21	maximum elapsed time	8
22	maximum run date (yymmdd)	8
23	minimum cpu seconds	8
24	minimum elapsed time	8
25	minimum run date (yymmdd)	8

Note: The elapsed time displayed for a shadow job is the elapsed time of the remote job to which it is bound.

Examples

To extract information about job `myjob` on workstation `main` and direct the output to the file `jinfo`, run the following command:

```
jbxtract -j myjob -c main -o jinfo
```

prxtract

Extracts information about prompts from the database.

Syntax

prxtract [**-V** | **-U**] [**-o** *output*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-o *output*
Specifies the output file. The default is **stdout**.

Results

Each prompt record contains tab-delimited, variable length fields. The fields are described in Table 69.

Table 69. Prxtract output fields

Field	Description	Max Length (bytes)
1	prompt name	8
2	prompt value	200

Examples

To extract information about all prompt definitions and direct the output to the file `prinfo`, run the following command:

```
prxtract -o prinfo
```

caxtract

Extracts information about calendars from the database.

Syntax

caxtract [**-V** | **-U**] [**-o** *output*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-o *output*
Specifies the output file. The default is **stdout**.

Results

Each calendar record contains tab-delimited, variable length fields. The fields are described in Table 70.

Table 70. Caxtract output fields

Field	Description	Max Length (bytes)
1	calendar name	8
2	calendar description	64

Examples

To extract information about all calendar definitions and direct the output to the file `cainfo`, run the following command:

```
caxtract -o cainfo
```

paxtract

Extracts information about global parameters (variables) from the database.

Syntax

paxtract [**-V** | **-U**] [**-o** *output*] [**-a**]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-o *output*
Specifies the output file. The default is **stdout**.

-a Displays all the variables defined in all the variable tables. If not specified, only the variables defined in the default variable table are displayed.

Results

Each variable record contains tab-delimited, variable length fields. The fields are described in Table 71.

Table 71. Paxtract output fields

Field	Description	Max Length (bytes)
1	table name	80
2	variable name	16
3	variable value	72

Remember: If you do not specify the **-a** (all) option in the command, only fields 2 and 3 are displayed and the variables listed are the ones contained in the default variable table only.

Examples

To extract information about all variable definitions and direct the output to the file `allvarinfo`, run the following command:

```
paxtract -a -o allvarinfo
```

rextract

Extracts information about resources from the database.

Syntax

rextract [**-V** | **-U**] [**-o** *output*]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

-o *output*
Specifies the output file. The default is **stdout**.

Results

Each resource record contains tab-delimited, variable length fields. The fields are described in Table 72.

Table 72. Rextract output fields

Field	Description	Max Length (bytes)
1	workstation name	8/16
2	resource name	8
3	total resource units	4
4	resource description	72

Examples

To extract information about all resource definitions and direct the output to the file `reinfo`, run the following command:

```
rextract -o reinfo
```

r11xtr

Extracts information about job streams from the database.

Syntax

r11xtr [**-V** | **-U**]
  [**-m** *mm*[*yyyy*]]
  [**-c** *wkstat*]
  [**-o** *output*]
  [**-s** *jstream_name*]

Arguments

-V Displays the program version and exits.

-U Displays program usage information and exits.

-m *mm*[*yy*]
  Specifies the month (*mm*) and, optionally, the year (*yy*) of the job streams. The default is the current month and year.

-c *wkstat*
  Specifies the workstation to be reported. The default is all workstations.

-s *jstream_name*
  Specifies the name of the job stream in which the jobs run. The default is all job streams.

-o *output*
  Specifies the output file. The default is **stdout**.

Results

| The ***MAESTRO_OUTPUT_STYLE*** variable specifies the output style for long object
| names. With value **LONG**, full length (long) fields are used for object names. This
| is the default. If the variable is set to anything other than **LONG**, long names are
| truncated to eight characters and a plus sign. For example: `A1234567+`.

Each job stream record contains tab-delimited, variable length fields. The fields are described in Table 73.

Table 73. R11xtr output fields

Field	Description	Max Length (bytes)
1	workstation name	16
2	job stream name	16
3	job stream date (yymmdd)	6
4	estimated cpu seconds	6
5	multiple workstation flag (* means some jobs run on other workstations)	1

Table 73. R11xtr output fields (continued)

Field	Description	Max Length (bytes)
6	number of jobs	4
7	day of week (Su, Mo, Tu, We, Th, Fr, Sa)	2

Examples

To extract information about job streams on June 2004 for workstation `main`, run the following command:

```
r11xtr -m 0604 -c main
```

To extract information about job streams on June of this year for all workstations, and direct the output to file `r11out`, run the following command:

```
r11xtr -m 06 -o r11out
```

xrxtrct

Extracts information about cross-references from the database.

Syntax

xrxtrct [**-V** | **-U**]

Arguments

-V Displays the command version and exits.

-U Displays command usage information and exits.

Results

| The ***MAESTRO_OUTPUT_STYLE*** variable specifies the output style for long object names. With value **LONG**, full length (long) fields are used for object names. This is the default. If the variable is set to anything other than **LONG**, long names are truncated to eight characters and a plus sign. For example: `A1234567+`.

The command output is written to eight files, **xdep_job**, **xdep_sched**, **xfile**, **xjob**, **xprompt**, **xresources**, **xsched**, and **xwhen**. These files are written in the current working directory. You must have write and execution rights on this directory to run the command.

Examples

To extract information about all cross-references, run the following command:

```
xrxtrct
```

xdep_job file

The **xdep_job** file contains two record types. The first contains information about jobs and job streams that are dependent on a job. Each dependent job and job stream record contains the fixed length fields, with no delimiters. The fields are described in Table 74 on page 516.

Table 74. Xdep_job output fields

Field	Description	Length (bytes)
1	**03**	2
2	workstation name	16
3	job name	40
4	job stream name	16
5	not used	240
6	dependent job stream workstation name	16
7	dependent job stream name	16
8	dependent job workstation name	16
9	dependent job name	40
10	not used	6
11	not used	6
12	not used	8
13	end-of-record (null)	1

The second record type contains information about jobs and job streams that are dependent on an internetwork dependency. Each dependent job and job stream record contains fixed length fields, with no delimiters. The fields are described in Table 75.

Table 75. Xdep_job output fields (continued)

Field	Description	Length (bytes)
1	**08**	2
2	workstation name	16
3	job name	120
4	not used	128
5	dependent job stream workstation name	16
6	dependent job stream name	16
7	dependent job workstation name	16
8	dependent job name	40
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xdep_sched file

The **xdep_sched** file contains information about jobs and job streams that are dependent on a job stream. Each dependent job or job stream record contains fixed length fields, with no delimiters. The fields are described in Table 76.

Table 76. Xdep_sched output fields

Field	Description	Length (bytes)
1	**02**	2
2	workstation name	16

Table 76. Xdep_sched output fields (continued)

Field	Description	Length (bytes)
3	job stream name	16
4	not used	248
5	dependent job stream workstation name	16
6	dependent job stream name	16
7	dependent job workstation name	16
8	dependent job name	40
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xfile file

The **xfile** file contains information about jobs and job streams that are dependent on a file. Each record contains fixed length fields, with no delimiters. The fields are described in Table 77.

Table 77. Xfile output fields

Field	Description	Length (bytes)
1	**07**	2
2	workstation name	16
3	file name	256
4	dependent job stream workstation name	16
5	dependent job stream name	16
6	dependent job workstation name	16
7	dependent job name	40
8	not used	6
9	not used	6
10	not used	8
11	end-of-record (null)	1

xjob file

The **xjob** file contains information about the job streams in which each job appears. Each job record contains fixed length fields, with no delimiters. The fields are described in Table 78.

Table 78. Xjob output fields

Field	Description	Length (bytes)
1	**04**	2
2	workstation name	16
3	job name	40
4	not used	248
5	job stream workstation name	16
6	job stream name	16

Table 78. Xjob output fields (continued)

Field	Description	Length (bytes)
7	not used	8
8	not used	8
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xprompt file

The **xprompt** file contains information about jobs and job streams that are dependent on a prompt. Each prompt record contains fixed length fields, with no delimiters. The fields are described in Table 79.

Table 79. Xprompts output fields

Field	Description	Length (bytes)
1	**05**	2
2	workstation name	16
3	prompt name or text	20
4	not used	236
5	dependent job stream workstation name	16
6	dependent job stream name	16
7	dependent job workstation name	16
8	dependent job name	40
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xresource file

The **xresource** file contains information about jobs and job streams that are dependent on a resource. Each resource record contains fixed length fields, with no delimiters. The fields are described in Table 80.

Table 80. Xresource output fields

Field	Description	Length (bytes)
1	**06**	2
2	workstation name	16
3	resource name	8
4	not used	248
5	dependent job stream workstation name	16
6	dependent job stream name	16
7	dependent job workstation name	16
8	dependent job name	40
9	units allocated	6

Table 80. Xresource output fields (continued)

Field	Description	Length (bytes)
10	not used	6
11	not used	8
12	end-of-record (null)	1

xsched file

The **xsched** file contains information about job streams. Each job stream record contains fixed length fields, with no delimiters. The fields are described in Table 81.

Table 81. Xsched output fields

Field	Description	Length (bytes)
1	**00**	2
2	workstation name	16
3	job stream name	16
4	not used	248
5	workstation name (same as 2 above)	16
6	job stream name (same as 3 above)	16
7	not used	8
8	not used	8
9	not used	6
10	not used	6
11	not used	8
12	end-of-record (null)	1

xwhen file

The **xwhen** file contains information about when job streams will run. Each job stream record contains the following fixed length fields, with no delimiters. The fields are described in Table 82.

Table 82. Xwhen output fields

Field	Description	Length (bytes)
1	**01**	2
2	workstation name	16
3	ON/EXCEPT name or date	8
4	except flag (*=EXCEPT)	1
5	not used	128
6	workstation name	16
7	job stream name	16
8	not used	8
9	not used	8
10	not used	6
11	offset num	6
12	offset unit	8

Table 82. Xwhen output fields (continued)

Field	Description	Length (bytes)
13	end-of-record (null)	1

Running Dynamic Workload Console reports and batch reports

You can run the following reports from the Dynamic Workload Console:

Job Run History Report
: A report collecting the historical job run data during a specified time interval. It is useful to detect which jobs ended in error or were late. It also shows which jobs missed their deadline, long duration jobs, and rerun indicators for reruns.

Job Run Statistics Report
: A report collecting the job run statistics. It is useful to detect success, error rates; minimum, maximum, and average duration; late and long duration statistics.

Workstation Workload Summary Report
: A report showing the workload on the specified workstations. The workload is expressed in terms of number of jobs that ran on them. It is useful for capacity planning adjustments (workload modelling and workstation tuning).

Workstation Workload Runtimes Report
: A report showing job run times and duration on the specified workstations. It is useful for capacity planning adjustments (workload modelling and workstation tuning).

Planned Production Details Report
: A report based on the information stored either in a trial or in a forecast plan. The information contained in these plans is retrieved from the Tivoli Workload Scheduler database. A Planned Production Details Report can be run on distributed engines (master domain manager and backup domain manager). A real production report extracted from a fault-tolerant agent might contain different information with respect to a plan extracted from a master domain manager. For example, the number of jobs and job streams is the same, but their status can change, because a job successful on the master can be in hold or ready on the agent. The update status rate is the same only on the full status agent that runs on the domain master.

Actual Production Details Report
: A report based on the information stored either in the current or in an archived plan. The information contained in these plans is retrieved from the Symphony files. Actual Production Details Report can be run on distributed engines (master domain manager, backup domain manager, domain manager with connector, and fault-tolerant agent with connector).

Custom SQL Report
: It enables you to create reports by running your own SQL queries. The reports will display a table with the column name as specified in the SELECT part of the SQL statement. The data for reporting is stored in a DB2 relational database and resides on the distributed side. Tivoli Workload Scheduler for z/OS connects to the database through the Java Database Connectivity (JDBC) interface. A JDBC driver type 4 is used to connect to the remote DB2 for LUW version 8.2, or later.

Some of these reports are also available as *batch reports* and can be run from a command line. For more information on how to run batch reports, see "Running batch reports from the command line interface" on page 525.

Depending on the interface from where you run the report or the operating system of the engine the following output formats are available:

Table 83. Supported report output formats

Name of the report	Output formats supported by the Dynamic Workload Console	Output formats supported by batch reports
Job Run History Report	HTML, CSV Only table format	HTML, CSV, PDF Only table format
Job Run Statistics Report	HTML, CSV Table and chart formats	HTML, CSV, PDF Table and chart formats
Workstation Workload Summary Report	HTML, CSV Table and chart formats	HTML, CSV, PDF Table and chart formats
Workstation Workload Runtimes Report	HTML, CSV Table and chart formats	HTML, CSV, PDF Table and chart formats
Planned Production Details Report	XML, CSV Only table format	N/A
Actual Production Details Report	XML, CSV Only table format	N/A
Custom SQL Report	HTML, CSV Only table format	HTML, CSV, PDF Only table format
Auditing general report (For more information, see the *Administration Guide* Keeping track of database changes using audit reports)	N/A	HTML, CSV, PDF Only table format
Auditing details report (For more information, see the *Administration Guide* Keeping track of database changes using audit reports)	N/A	HTML, CSV, PDF Only table format

You must have the appropriate security file authorizations for report objects to run these reports (granted by default to the *tws_user* on fresh installations). See *Administration Guide* for security file information.

See also the *Administration Guide* to learn how to configure the Dynamic Workload Console to view reports.

Historical reports

The following table summarizes the historical reports in terms of their:

- Functionality
- Selection criteria
- Output content options

Table 84. Summary of historical reports

Report name	Description	Selection criteria	Output content options
Job run history	Corresponds to **Report 07**. Collects historical job execution data during a time interval. Helps you find: • Jobs ended in error • Late jobs • Missed deadlines • Long duration • Rerun indicators for reruns • Other historical information.	• Job name, job stream name, workstation name, and workstation name (job stream). Each field can be specified using a wildcard. • Status (Success, Error, Unknown) • Delay indicators • Job execution interval • Include/Exclude rerun iterations	You can select from the following: • Actual start time • Estimated duration • Actual duration • Job number • Started late (delay) • Ended late (delay) • Status • Long duration • Job definition name • CPU consumption (not available on Windows workstations) • Logon user • Rerun type • Iteration number • Return code The output is in table view.

Table 84. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
Job run statistics	Corresponds to **Report 01**. Collects job execution statistics. Helps you find: • Success/error rates • Minimum and maximum elapsed and CPU times • Average duration • Late and long duration statistics **Note:** The report does not include jobs that were submitted using an alias name.	• Job name, workstation name, and user login. Each field can be specified using a wildcard. • Percentage of jobs in Success, Error, Started late, Ended late, and Long duration • Total runs and total reruns	You can select from the following: • Job details: – Logon user – Job creator – Description – Script – Recovery information • Job statistics: – Total runs (divided in Successful and Error) – Total of runtime exceptions (Started late, Ended late, Long duration) – Minimum, maximum and average duration and CPU times (for successful runs only) – CPU consumption (not available on Windows workstations) • Report format: – Charts view – Table view – Include table of contents by job or by workstation

Table 84. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
Workstation workload summary	Provides data on the workload in terms of the number of jobs that have run on each workstation. Helps making the necessary capacity planning adjustments (workload modeling, and workstation tuning).	• Workstation names. Each field can be specified using a wildcard. • Date ranges or specific days for workload filtering. • Relative time intervals (allows to reuse the same report task for running each day and getting the report of the production of the day before)	You can select from the following: • Workstation information granularity arranged by: – Hour – Day – Production day • Information aggregation options: – Provide cumulative and aggregated workstation summary information for all or a desired subset of workstations • Report format: – Charts view – Table view – Include table of contents by date or by workstation
Workstation workload runtimes	Corresponds to **Report 08**. Provides data on the job runs (time and duration) on the workstations. Helps making the necessary capacity planning adjustments (workload modeling, and workstation tuning).	• Job and workstation names. Each field can be specified using a wildcard. • Workload execution period • Daily time intervals	You can select from the following: • Information grouped by: – Workstation – Run date Can be ordered by rerun iteration • Production day • Job information: – Actual duration – Status – Rerun iteration – Job definition name • Report format: – Charts view – Table view
Custom SQL	A wizard helps you define your custom SQL query (only on the database views which you are authorized to access).	The criteria specified in the custom SQL query.	The resulting report has a table with the column name as specified in the SELECT part of the SQL statement.

Production reports

The following table summarizes the production reports in terms of their:

- Functionality
- Selection criteria
- Output content options

Table 85. Summary of production reports

Report name	Description	Selection criteria	Output content options
Actual production details	Corresponds to **Report 10B**. Provides data on current and archived plans.	• Job name • Workstation name (job) • Job stream name • Workstation name (job stream)	You can select from the following: • Report format: – Flat – CSV – Microsoft Project • Include: – First level predecessor – Job log
Planned production details	Corresponds to **Report 9B**. Provides data on trial and forecast plans.	• Job name • Workstation name (job) • Job stream name • Workstation name (job stream)	You can select from the following: • Report format: – Flat – CSV – Microsoft Project • Include: – First level predecessor – Job log

Running batch reports from the command line interface

This section describes how you can run from the command line the reports listed in "Historical reports" on page 521.

Using a command-line interface, you can schedule these reports to run on a timely basis.

A sample business scenario

To avoid unexpected slowing down in the workload processing, the analyst of a big company needs weekly reports collecting historical information about the processed workload to determine and analyze any workload peaks that might occur.

To satisfy this request, the TWSWEBUIDeveloper creates *Workload Workstation Summary Reports* (WWS) and *Workload Workstation Runtimes Reports* (WWR).

To accomplish his task, he runs the following steps:

1. He customizes the property files related to the Workload Workstation Summary and Workload Workstation Runtimes reports, specifying the format and content of the report output.
2. He schedules jobs to obtain WWS and WWR reports:
   - The first job generates a WWS report to be saved locally.

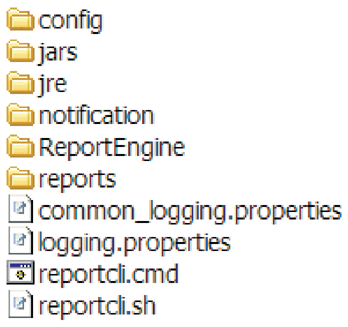
   - The second job runs a WWR report overnight on expected workload peaks time frames. The report output is sent using an mail to the analyst. The information collected are used to optimize the workload balance on the systems.
3. He adds the two jobs to a job stream scheduled to run weekly and generates the plan.

Setting up for command line reporting

Before running batch reports you must run a few setup steps:

1. The software needed to run batch reports is contained in a package named `TWSBatchReportCli`, included in the Tivoli Workload Scheduler installation image, in the `TWSBatchReportCli` directory. If you plan to run batch reports from within a scheduled job, extract the package file on one of the platforms listed in http://www.ibm.com/support/docview.wss?rs=672&uid=swg27020800

   After you extract the package, you will have the following file structure:

   ```
   config
   jars
   jre
   notification
   ReportEngine
   reports
   common_logging.properties
   logging.properties
   reportcli.cmd
   reportcli.sh
   ```

   Because the native UNIX tar utility does not support long file names, if you are extracting the files on AIX, Solaris, or HP-UX systems, ensure that the latest GNU version of tar (gtar) is installed to extract the files successfully.

   Note:

   a. Make sure you run the following commands in the directory where you extracted the files:

      On UNIX

      ```
      chmod -R +x *
      chown -R username *
      ```

      On Windows

      Ensure Tivoli Workload Scheduler is installed.

      ```
      setown -u username *
      ```

      where *username* is the Tivoli Workload Scheduler user that will run the reports.

   b. If you plan to schedule jobs that run batch reports, the system where you extract the package must be accessible as network file system from a fault-tolerant agent defined in the local scheduling environment.

2. Configure the template file `.\config\common.properties` specifying the information to:

   a. Connect to the database where the historical data are stored.

b. Set the date and time format, including the time zone. The file `.\config\timezone.txt` contains a list of time zones supported by Tivoli Workload Scheduler and the information on how to set them. The time zone names are case sensitive.

c. Make available the report output on the URL specified in **ContextRootUrl** field. This is an example of configuration settings:

```
######################################################################
 # HTTP Server information
 ######################################################################

 #Specify the context root where the report will be available
 #To leverage this possibility it needs to specify in the report output dir
 #the directory that is referred by your HTTP Server with this contect root

 ContextRootUrl=http://myserver/reportoutput
```

In this case make sure that the *output_report_dir* specified when running the batch reports command points to the same directory specified in the **ContextRootUrl** field.

d. Send the report output using a mail. This is an example of configuration settings:

```
######################################################################
 # Email Server configuration
 ######################################################################
 PARAM_SendReportByEmail=true

 #SMTP server
 mail.smtp.host=myhost.mydomain.com
 #IMAP provider
 mail.imap.socketFactory.fallback=false
 mail.imap.port=993
 mail.imap.socketFactory.port=993
 #POP3 provider
 mail.pop3.socketFactory.fallback=false
 mail.pop3.port=995
 mail.pop3.socketFactory.port=995

 ######################################################################
 # Email properties
 ######################################################################
 PARAM_EmailFrom=user1@your_company.com
 PARAM_EmailTo=user2@your_company.com,user3@your_company.com
 PARAM_EmailCC=user4@your_company.com
 PARAM_EmailBCC=user5@your_company.com
 PARAM_EmailSubject=Test send report by email
 PARAM_EmailBody=This is the report attached
```

An explanation of all the customizable fields is contained in the template file.

Note: If you plan to run Workstation Workload Runtime reports ensure that the file system where database is installed has enough free space. if a shortage of disk space occurs an SQL exception like the following is triggered:

```
DB2 SQL error: SQLCODE: -968, SQLSTATE: 57011
```

Running batch reports

The `\reports\templates` directory contains a sample template file for each type of report.

Before running any of these reports make sure you customize the corresponding template file.

In that file, named *report_name*.properties, you can specify:

- The information to display in the report header.
- How to filter the information to display the expected result.
- The format and content of the report output.

For more information about the specific settings see the explanation provided in the template file beside each field.

If you are using DBCS characters to specify the parameters in the template .properties files, ensure you save the file in UTF-8 encoding.

After you set up the environment as it is described in "Setting up for command line reporting" on page 526, and you configured the report template file, use the following syntax to run the report:

reportcli **-p** *report_name.property*
    [**-o** *output_report_dir*]
    [**-r** *report_output_name*]
    [**-k** key=*value*]
    [**-k** key=*value*]
    ....

where:

-p *report_name.property*
: Specifies the path name to the report template file.

-o *output_report_dir*
: Specifies the output directory for the report output.

-r *report_output_name*
: Specifies the name of the report output.

-k key=*value*
: Specifies the value of a settings. This value override the corresponding value, if defined, in the common.properties file or in the *report_name*.properties file.

Examples

1. In this example the reportcli.cmd is run with the default parameter and produces jrh1 report:
   ```
   reportcli.cmd -p D:\ReportCLI\TWSReportCli\reports\templates\jrh.properties
   -r jrh1
   ```
2. In this example the reportcli.cmd is run using the -k parameter to override the values set for **PARAM_DateFormat** in the .\config\common.properties file produces jrh1 report:
   ```
   reportcli.cmd -p D:\ReportCLI\TWSReportCli\reports\templates\jrh.properties
   -r jrh2 -k PARAM_DateFormat=short
   ```
3. In this example the reportcli.cmd is run using the -k parameter to override the format specified for the report output in the PROPERTIES file produces jrh1 report:
   ```
   ./reportcli.sh -p /TWSReportCli/REPCLI/reports/templates/wwr.properties
   -r wwr3 -k REPORT_OUTPUT_FORMAT=html -k OutputView=charts
   ```
4. Do the following if you want to run a Custom SQL report and make available the output of the report at the following URL as http://myserver/reportoutput/report1.html:

a. Configure the `ContextRootUrl` parameter in the `common.properties` files as follows:

   ```
   ######################################################################
    # HTTP Server information
    ######################################################################

    #Specify the context root where the report will be available
    #To leverage this possibility it needs to specify in the report output dir
    #the directory that is referred by your HTTP Server with this contect root

    ContextRootUrl=http://myserver/reportoutput
   ```

b. When you run a batch reports command specify as *output_report_dir* a directory that points to the same HTTP directory specified in the `ContextRootUrl`. For example, if you mapped locally the `http://myserver/` as `R:` driver, you can run the following command:

   ```
   reportclibat
      -p REPORT_CLI_DIR\reports\TWS\historical\templates\sql.properties
      -r report1
      -o R:\reportoutput
   ```

c. As a confirmation for the successful run of the report, the following message is displayed:

   ```
   AWSBRC0106I Report available on: http://myserver/reportoutput/report1.html
   ```

   This URL shows where the report output is available.

Note: If the report is run through a Tivoli Workload Scheduler job, the output of the command is displayed in the job output.

Logs and traces for batch reports

The file `./common_logging.properties` contains the parameters you can use to configure tracing and logging.

The file contains the following settings:

```
logFileName=reportcli.log
traceFileName=trace.log
trace=off
birt_trace=off
```

where:

logFileName
: Specifies the name of the file containing generic information, warning about potential problems, and information about errors. This file is store under `./log`.

traceFileName
: Specifies the name of the file containing traces. If you set `trace=on` the trace file is store under `./log`.

trace
: Specifies whether to enable or not traces. Enable the traces by setting `trace=on` if you want to investigate further about an error,

birt_trace
: Specifies whether to enable or not traces to diagnose errors in BIRT engine. If you set `birt_trace=on` a file containing the trace and named `ReportEngine_aaaa_mm_dd_hh_mm_ss.log` is stored in the `/ReportEngine/logs` folder

Chapter 16. Managing time zones

Tivoli Workload Scheduler supports different time zones. If you enable time zones you can manage your workload across a multiple time zone environment.

Both the 3-character and the variable length notations are supported.

The 3-character notation is supported for backward compatibility with previous versions of Tivoli Workload Scheduler.

The variable length notation format is area/city, for example Europe/Paris as equivalent to ECT (European Central Time).

The chapter is made up by the following sections:
- "Enabling time zone management"
- "How Tivoli Workload Scheduler manages time zones" on page 532
- "Moving to daylight saving time on" on page 533
- "Moving to daylight saving time off" on page 534
- "General rules" on page 534

Enabling time zone management

You can enable or disable the management of time zones by modifying the setting assigned to the global option ***enTimeZone*** on the master domain manager using the **optman** command line. The setting takes effect after the next **JnextPlan** is run. These are the available settings:

no Disable time zone management. This means that the values assigned to all **timezone** keywords in the definitions are ignored. All the `at`, `until`, and `deadline` time restrictions are managed individually by each fault-tolerant agent, including the master and the domain managers, thus ignoring the time zone of the agent scheduling the job or job stream. As a consequence, when different time zones are involved:
- For jobs, incorrect information is displayed about these time dependencies when looked at from an agent other than the job owner. This has no impact however on the scheduling process of the job.
- For job streams, the impact is that each agent processes the time dependencies by its own time zone, and therefore at different times, causing jobs of the same job stream, but defined on a different agent, to run at a different time.

yes Enable time zone management. This means that the values assigned to the **timezone** settings are used to calculate the time when the jobs and job streams run on the target workstations.

By default the *enTimeZone* option is set to **yes**.

For more details on how to use the **optman** command line to manage global options on the master domain manager, refer to the *IBM Tivoli Workload Scheduler Administration Guide*.

How Tivoli Workload Scheduler manages time zones

When the time zone is enabled, you can use time zone settings in workstation, job, and job stream definitions.

While performing plan management activities, Tivoli Workload Scheduler converts the value set for the time zones into object definitions. The conversions are applied in this order:

1. When the job stream instances are added to the preproduction plan, the time zone set in the job stream definitions is converted into the GMT time zone and then the external follows dependencies are resolved.
2. When the production plan is created or extended, the job stream instances are assigned to workstations where the instance is scheduled to run and the time zone is converted from GMT into the time zone set in the target workstation definition.

This is why if you use the **conman showsched** or **conman showjobs** commands to see the information about scheduled jobs and job streams you see the time zone values expressed using the time zone set on the workstation where the job or job stream is planned to run. Based on the setup of the *enLegacyStartOfDayEvaluation* global option, you can decide how the product manages time zones while processing, and precisely:

If you set the value of *enLegacyStartOfDayEvaluation* **to** *no*

The value assigned to the *startOfDay* option on the master domain manager is not converted into the local time zone set on each workstation across the network. This means that if the *startOfDay* option is set to 0600 on the master domain manager, it is 0600 in the local time zone set on each workstation in the network. This also means that the processing day begins at the same hour, but not at the same moment, on all workstations.

Figure 27 shows you how the start of day, set to 0600 on the master domain manager, is applied to the different time zones on the two fault-tolerant agents. The same time conversion is applied to the three instances of job stream **JS1** scheduled to run on the three machines and containing an **at** time dependency at `0745 US/Central` time zone. The time frame that identifies the new processing day is greyed out in Figure 27.

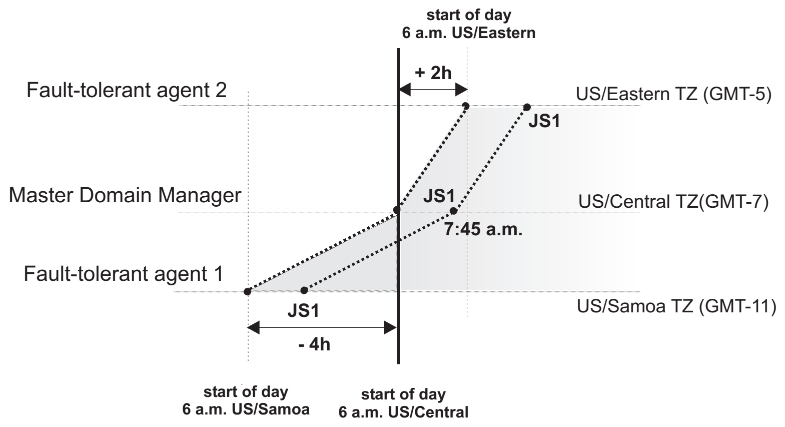


Figure 27. Example when start of day conversion is not applied

If you set the value of *enLegacyStartOfDayEvaluation* **to** *yes*

The value assigned to the *startOfDay* option on the master domain manager is converted into the local time zone set on each workstation across the network. This means that if the *startOfDay* option is set to 0600 on the master domain manager, it is converted on each workstation into the corresponding time according to the local time zone set on that workstation. This also means that the scheduling day begins at the same moment, but not necessarily at the same hour, on all workstations in the network.

Figure 28 shows you how the start of day, set to 0600 on the master domain manager, is applied to the different time zones on the two fault-tolerant agents. It also shows how the timing of the three instances of job stream **JS1** scheduled to run on the three machines and containing an **at** time dependency at `0745 US/Central` time zone is not modified because of the *startOfDay* conversion. The time frame that identifies the new processing day is greyed out in Figure 28.

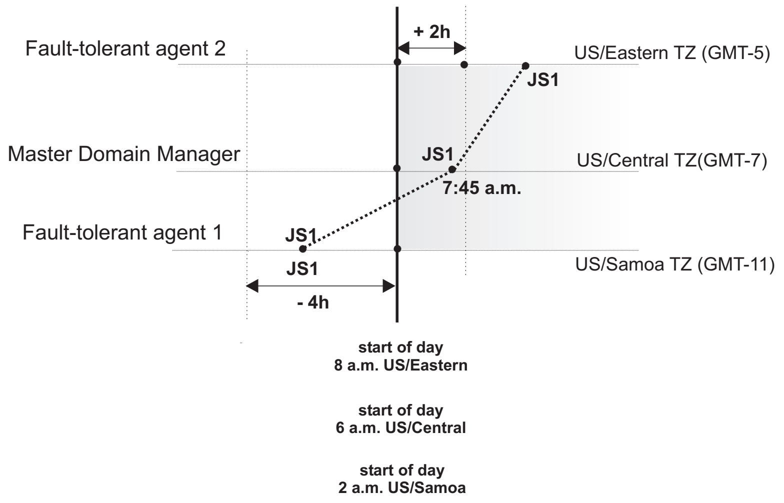


Figure 28. Example when start of day conversion is applied

Note: Starting from version 8.3 there is no linking between the time set for the *startOfDay* and the moment when **JnextPlan** is run. **JnextPlan** can be run at any time and the *startOfDay* indicates only the moment when the new processing day starts.

By default the *enLegacyStartOfDayEvaluation* global option is set to **no**.

For more details on how to use the **optman** command line to manage global options on the master domain manager, refer to the *IBM Tivoli Workload Scheduler Administration Guide*.

Moving to daylight saving time on

Tivoli Workload Scheduler manages the moving to daylight saving time (DST) when generating the production plan. This means that the date and time to run assigned to jobs and job streams in the plan is already converted into the corresponding date and time with DST on.

The following example explains how Tivoli Workload Scheduler applies the time conversion when **JnextPlan** is run to generate or extend the production plan while the time moves to DST.

If DST is switched on at 3:00 p.m. then all job streams scheduled to start between 2:00 and 2:59 are set to start one hour later. This happens because the planned phase of Tivoli Workload Scheduler. For example a job defined to run AT 0230 in the morning, will be scheduled to run at 03:30 a.m.

Moving to daylight saving time off

Moving to daylight saving time (DST) off, the clock time is set to one hour earlier with respect to the DST time. To maintain consistency with production planning criteria, Tivoli Workload Scheduler ensures that the job stream instances planned to run during the hour before the time shift backward are run only one time. Because the time conversion is applied when generating or extending the production plan, the date and time to run assigned to jobs and job streams in the plan is already converted into the corresponding date and time with DST off.

If a job stream or a job run on a timezone where the DST time switches off, that is the clock is put one hour back, and if you define a time dependency for such job streams or jobs in relation to another timezone, it might happen that this time dependency occurs during the second, repeated time interval. In this case the time dependency would be resolved during the first time interval.

Tivoli Workload Scheduler recognizes that the time dependency occurs on the second, repeated time interval and resolves it accordingly.

General rules

When the time zone is enabled in the Tivoli Workload Scheduler environment, regardless of which value is set for the *enLegacyStartOfDayEvaluation* option, some general rules are applied. These rules are now described divided by topic:

Identifying default time zone settings for jobs and job streams:
Within a job stream definition you can set a time zone for the entire job stream and for the jobs contained in the job stream. These time zones can differ from each other. To manage all possible time zone settings the time zone conversion is made respecting the following criteria:

- If a time zone is not set for a job within a job stream, then that job inherits the time zone set on the workstation where the job is planned to run.
- If a time zone is not set for a job stream, then the time zone set is the one set on the workstation where the job stream is planned to run.
- If none of the mentioned time zones is set, then the time zone used is the one set on the master domain manager.

Choosing the correct time zone for the workstations:
To avoid inconsistencies, before enabling the time zone management feature across the Tivoli Workload Scheduler network, make sure that, if a time zone is set in the workstation definition, it is the same as the time zone set on the system where the workstation is installed.

Default time zone setting for the master domain manager:
If a time zone is not set in the master domain manager definition, it inherits the time zone set on the system where the master domain manager is installed. To see which time zone is set on the master domain manager you can run the following command:

```
conman showcpu;info
```

Using the time zone on extended agents:

Extended agents inherit the time zone of the master domain manager.

Displaying time zone setting in production for an AT time dependency:
If you use **conman** commands **sj** or **ss** to display a job or a job stream having an **at** time dependency with a time zone set, the time specified for the **at** dependency is displayed applying the time zone defined on the workstation where the job or job stream is defined to run.

Applying an offset to a time zone when scheduling a job stream:
If you submit in production a job stream specifying an **at** dependency with an offset of +n days, then Tivoli Workload Scheduler first adds the offset to the date and then converts the time zone set in the **at** dependency. This is important especially when referring to the time when daylight saving time moving occurs.

As a best practice, if you enable time zone management, set a time zone on each workstation of your Tivoli Workload Scheduler network.

Chapter 17. Defining access methods for agents

Access methods are used to extend the job scheduling functions of Tivoli Workload Scheduler to other systems and applications. They run on:

Extended agents

They are logical workstation related to an access method hosted by a physical Tivoli Workload Scheduler workstation (not another extended agent). More than one extended agent workstation can be hosted by the same Tivoli Workload Scheduler workstation and use the same access method. The extended agent runs on fault-tolerant agents defined using a standard Tivoli Workload Scheduler workstation definition, which gives the extended agent a name and identifies the access method. The access method is a program that is run by the hosting workstation whenever Tivoli Workload Scheduler submits a job to an external system.

Jobs are defined for an extended agent in the same manner as for other Tivoli Workload Scheduler workstations, except that job attributes are dictated by the external system or application.

Information about job running execution is sent to Tivoli Workload Scheduler from an extended agent using the job `stdlist` file. A method options file can specify alternate logins to launch jobs and check *opens* file dependencies. For more information, see the *Tivoli Workload Scheduler: User's Guide and Reference*.

A physical workstation can host a maximum of 255 extended agents.

dynamic agents and Tivoli Workload Scheduler for z/OS agents

They communicate with external systems to start the job and return the status of the job. To run access methods on external applications using dynamic agents, you define a job of type **access method**.

Access methods are available on the following systems and applications.
- Oracle E-Business Suite
- SAP R/3
- z/OS
- Custom methods
- unixssh
- unixrsh
- Local UNIX (fault-tolerant agents only)

The UNIX access methods included with Tivoli Workload Scheduler, are described in UNIX access methods. If you are working with dynamic agents, for information about defining Tivoli Workload Scheduler workstations, see the section that explains how to define workstations in the database, see "Workstation definition" on page 127 in *Tivoli Workload Scheduler: User's Guide and Reference*. For information about writing access methods, also see the *Tivoli Workload Scheduler: User's Guide and Reference*, section "Access method interface."

Access method interface

The interface between Tivoli Workload Scheduler and an access method consists of information passed to the method on the command line, and of messages returned to Tivoli Workload Scheduler in **stdout**.

Method command line syntax

The Tivoli Workload Scheduler host runs an access method using the following command line syntax:

methodname **-t** *task options* **--** *taskstring*

where:

methodname
: Specifies the file name of the access method. All access methods must be stored in the directory: `TWS_home/methods`

-t *task*
: Specifies the task to be performed, where *task* is one of the following:

  LJ Launches a job.

  MJ Manages a previously launched job. Use this option to resynchronize if a prior **LJ** task ended unexpectedly.

  CF Extended agents only. Checks the availability of a file. Use this option to check file `opens` dependencies.

  GS Extended agents only. Gets the status of a job. Use this option to check job `follows` dependencies.

options
: Specifies the options associated with the task. See "Task options" for more information.

taskstring
: A string of up to 255 characters associated with the task. See "Task options."

Task options

The task options are listed in Table 86. An X means that the option is valid for the task.

Table 86. Method command task options

Task	-c	-n	-p	-r	-s	-d	-l	-o	-j	-q	-w	-S	Task String
LJ	X	X	X	X	X	X	X	X	X			X	*ljstring*
MJ	X	X	X	X	X	X	X	X	X				*mjstring*
CF	X	X	X							X			*cfstring*
GS	X	X	X	X		X					X		*gsstring*

-c *xagent***,***host***,***master*
: Specifies the names of the agent, the host, and the master domain manager separated by commas.

-n *nodename*
: Specifies the node name of the computer associated with the agent, if any. This is defined in the extended agent's workstation definition **Node** field.

-p *portnumber*
: Specifies the TCP/IP port number associated with the agent, if any. This is defined in the agent workstation definition **TCP Address** field.

-r *currentrun***,***specificrun*
: Specifies the current run number of Tivoli Workload Scheduler and the

specific run number associated with the job separated by a comma. The current and specific run numbers might be different if the job was carried forward from an earlier run.

-s *jstream*
: Specifies the name of the job's job stream.

-d *scheddate*,*epoch*
: Specifies the job stream date (*yymmdd*) and the epoch equivalent, separated by a comma.

-l *user*
: Specifies the job's user name. This is defined in the job definition **Logon** field.

-o *stdlist*
: Specifies the full path name of the job's standard list file. Any output from the job must be written to this file.

-j *jobname*,*id*
: Specifies the job's name and the unique identifier assigned by Tivoli Workload Scheduler, separated by a comma. The name is defined in the job definition **Job Name** field.

-q *qualifier*
: Specifies the qualifier to be used in a test command issued by the method against the file.

-w *timeout*
: Specifies the amount of time, in seconds, that Tivoli Workload Scheduler waits to get a reply on an external job before sending a SIGTERM signal to the access method. The default is 300.

-S *new name*
: Specifies that the job is rerun using this name in place of the original job name. Within a job script, you can use the `jobinfo` command to return the job name and run the script differently for each iteration.

-- *ljstring*
: Used with the **LJ** task. The string from the **Script File** or **Command** field of the job definition.

-- *mjstring*
: Used with the **MJ** task. The information provided to the Tivoli Workload Scheduler by the method in a message indicating a job state change **%CJ** (see "Method response messages" on page 540 for additional details on messages indicating job state change) following to a **LJ** task. Usually, this identifies the job that was launched. For example, a UNIX method can provide the process identification (PID) of the job it launched, which is then sent by the Tivoli Workload Scheduler as part of an **MJ** task.

-- *cfstring*
: Used with the **CF** task. For a file `opens` dependency, the string from the **Opens Files** field of the job stream definition.

-- *gsstring*
: Used with the **GS** task. Specifies the job whose status is checked. The format is as follows:

followsjob[,*jobid*]

where:

followsjob
The string from the **Follows Sched/Job** list of the job stream definition.

jobid An optional job identifier received by Tivoli Workload Scheduler in a **%CJ** response to a previous **GS** task.

Method response messages

Methods return information to Tivoli Workload Scheduler in messages written to **stdout**. Each line starting with a percent sign (%) and ending with a new line is interpreted as a message. The messages have the following format:

%CJ *state* [*mjstring* | *jobid*]

%JS [*cputime*]

%RC *rc*

%UT [*errormessage*]

where:

CJ Changes the job state.

state The state to which the job is changed. All Tivoli Workload Scheduler job states are valid except HOLD and READY. For the **GS** task, the following states are also valid:

ERROR
An error occurred.

EXTRN
Status is unknown.

mjstring
A string of up to 255 characters that Tivoli Workload Scheduler will include in any **MJ** task associated with the job.

jobid A string of up to 64 characters that Tivoli Workload Scheduler will include in any **GS** task associated with the job.

JS [*cputime***]**
Indicates successful completion of a job and provides its elapsed run time in seconds.

RC *rc* *rc* is a number that is interpreted by Tivoli Workload Scheduler as the return code of the extended agent job. The return code is taken into account only if a return code condition was specified in the definition of the extended agent job. Otherwise, it is ignored and the successful completion of the extended agent job is indicated by the presence of message **%JS** *[cputime]*. Likewise, if the method does not send the **%RC** message, then the successful completion of the extended agent job is indicated by the presence of message **%JS** *[cputime]*.

UT [*errormessage***]**
Indicates that the requested task is not supported by the method. Displays a string of up to 255 characters that Tivoli Workload Scheduler will include in its error message.

Method options file

For extended, dynamic agents, and Tivoli Workload Scheduler for z/OS agent you can use a method options file to specify login information and other options.

An options file is a text file located in the methods directory of the Tivoli Workload Scheduler installation, containing a set of options to customize the behavior of the access method. The options must be written one per line and have the following format (with no spaces included):

```
option=value
```

All access methods use two types of options files:

Extended agents

Global options file
A common configuration file created by default for each access method installed, whose settings apply to all the extended agent workstations defined for that method. When the global options file is created, it contains only the **LJuser** option, which represents the operating system user ID used to launch the access method. You can customize the global options file by adding the options appropriate to the access method.

Local options file
A configuration file that is specific to each extended agent workstation within a particular installation of an access method. The name of this file is `XANAME_accessmethod.opts`, where:

XANAME
Is the name of the extended agent workstation. The value for *XANAME* must be written in uppercase alphanumeric characters. Double-byte character set (DBCS), Single-byte character set (SBCS), and Bidirectional text are not supported.

accessmethod
Is the name of the access method.

If you do not create a local options file, the global options file is used. Every extended agent workstation, except for z/OS, must have a local options file with its own configuration options.

For example, if the installation of the access method includes two extended agent workstations, `CPU1` and `CPU2`, the names of the local options files are respectively `CPU1_accessmethod.opts` and `CPU2_accessmethod.opts`.

Tivoli Workload Scheduler reads the options file, if it exists, before running an access method. For extended agents, if the options file is modified after Tivoli Workload Scheduler is started, the changes take effect only when it is stopped and restarted.

Tivoli Workload Scheduler for z/OS agents and dynamic agents

Global options file
A common configuration file created by default for each access method installed, whose settings apply to all the agent workstations defined for that method. When the global options file is created, it contains only the **LJuser** option, which represents the

operating system user ID used to run the access method. You can customize the global options file by adding the options appropriate to the access method.

The name of the global options file is *accessmethod*.opts, where access method is the name of the method you are creating.

Local options file
A configuration file that is specific to each access method. The name of this file is *optionsfile_accessmethod*.opts,

In a distributed environment:

- If you are defining a job to run the access method by using the Dynamic Workload Console it is the options file you specified in the **New** > **Job definition** > **ERP** > **Access Method** XA Task tab.
- If you are defining the access method by using **composer** it is the options file you specified in the **target** attribute of the job definition.

If you do not create a local options file, the global options file is used.

In a z/OS environment:

- If you are defining a job to run the access method by using the Dynamic Workload Console it is the options file you specified in the **New** > **ERP** > **Access Method** XA Task tab.
- If you are defining the access method by using the **JOBREC** statement it is the name of the workstation where the access method runs.

If you do not create a local options file, the global options file is used.

If you do not specify an option in the *options_file_accessmethod*.opts file the product uses the value specified for that option in the global option file. If you do not specify them neither in the *options_file_accessmethod*.opts file nor in the global option file the product issues an error message.

The options file must have the same path name as its access method, with an .opts file extension. For example, the Windows path name of an options file for a method named netmeth is

```
TWS_home\methods\netmeth.opts
```

Tivoli Workload Scheduler reads the options file, if it exists, before running an access method.

The options recognized by Tivoli Workload Scheduler are as follows:

LJuser=*username*
Specifies the login to use for the **LJ** and **MJ** tasks. The default is the login from the job definition. See "Launch job task (LJ)" on page 543 and "Manage job task (MJ)" on page 544.

CFuser=*username*
Extended agents only. Specifies the login to use for the **CF** task. The default for UNIX is **root**, and for Windows is the user name of the account in which the product was installed. See "Check file task (CF) extended agents only" on page 544.

GSuser=*username*
: Specifies the login to use for the **GS** tasks. The default for UNIX is **root**, and for Windows is the user name of the account in which Tivoli Workload Scheduler was installed. See "Get status task (GS) extended agents only" on page 545.

GStimeout=*seconds*
: Specifies the amount of time, in seconds, Tivoli Workload Scheduler waits for a response before killing the access method. The default is **300** seconds.

nodename=*node_name*
: Specifies the host name or IP address if required by the method you are defining. For the **unixssh** access method, the host name or IP address to connect to the remote engine.

port=*port_number*
: Specifies the port number if required by the method you are defining. For the **unixssh** access method, the port to connect to the remote engine.

For Tivoli Workload Scheduler for z/OS agents and dynamic agents, you can specify the node name and port number also in the `JobManager.ini` file.

If you do not specify them in the *options_file_accessmethod*.`opts` file the product uses the value specified in the global option file. If you do not specify them neither in the *options_file_accessmethod*.`opts` file nor in the global option file the product uses the value specified in the *option_file* stanza of the `JobManager.ini` file.

Note: If the extended agent host is a Windows computer, these users must be defined as Tivoli Workload Scheduler user objects.

Running methods

The following subsections describe the interchange between Tivoli Workload Scheduler and an access method.

Launch job task (LJ)

The **LJ** task instructs the extended agent method to launch a job on an external system or application. Before running the method, Tivoli Workload Scheduler establishes a run environment. The **LJuser** parameter is read from the method options file to determine the user account with which to run the method. If the parameter is not present or the options file does not exist, the user account specified in the **Logon** field of the job's definition is used. In addition, the following environment variables are set:

HOME
: The login user's home directory.

LOGNAME
: The login user's name.

PATH
: For UNIX, it is set to`/bin:/usr/bin`. For Windows, it is set to`%SYSTEM%\SYSTEM32`.

TWS_PROMOTED_JOB
: Set to `YES`, when the job (a mission- critical job or one of its predecessors) is promoted.

TZ
: The time zone.

If the method cannot be run, the job is placed in the FAIL state.

Once a method is running, it writes messages to its **stdout** that indicate the state of the job on the external system. The messages are summarized in Table 87.

Table 87. Launch job task (LJ) messages

Task	Method Response	Tivoli Workload Scheduler Action
LJ and **MJ**	**%CJ** *state* [*mjstring*]	Sets job state to *state*. Includes *mjstring* in any subsequent **MJ** task.
	%JS [*cputime*]	Sets job state to SUCC.
	Exit code=non-zero	Sets job state to ABEND.
	%UT [*errormessage*] and Exit code=2	Sets job state to ABEND and displays *errormessage*.

A typical sequence consists of one or more **%CJ** messages indicating changes to the job state and then a **%JS** message before the method exits to indicate that the job ended successfully. If the job is unsuccessful, the method must exit without writing the **%JS** message. A method that does not support the **LJ** task, writes a **%UT** message to **stdout** and exits with an exit code of **2**.

Manage job task (MJ)

The **MJ** task is used to synchronize with a previously launched job if Tivoli Workload Scheduler determines that the **LJ** task ended unexpectedly. Tivoli Workload Scheduler sets up the environment in the same manner as for the **LJ** task and passes it the *mjstring*. See "Launch job task (LJ)" on page 543 for more information.

If the method locates the specified job, it responds with the same messages as an **LJ** task. If the method is unable to locate the job, it exits with a nonzero exit code, causing Tivoli Workload Scheduler to place the job in the ABEND state.

Killing a job

While an **LJ** or **MJ** task is running, the method must trap a SIGTERM signal (signal **15**). The signal is sent when an operator issues a **kill** command from Tivoli Workload Scheduler console manager. Upon receiving the signal, the method must attempt to stop (**kill**) the job and then exit without writing a **%JS** message.

Check file task (CF) extended agents only

The **CF** task requests the extended agent method to check the availability of a file on the external system. Before running the method, Tivoli Workload Scheduler establishes a run environment. The **CFuser** parameter is read from the method options file to determine the user account with which to run the method. If the parameter is not present or the options file does not exist, on UNIX the **root** user is used and, on Windows, the user name of the account in which Tivoli Workload Scheduler was installed is used. If the method cannot be run, the file `opens` dependency is marked as failed, that is, the file status is set to **NO** and any dependent job or job stream is not allowed to run.

Once it is running, the method runs a test command, or the equivalent, against the file using the qualifier passed to it in the **-q** command line option. If the file test is true, the method exits with an exit code of zero. If the file test is false, the method exits with a nonzero exit code. This is summarized in Table 88 on page 545.

Table 88. Check file task (CF) messages

Task	Method Response	Tivoli Workload Scheduler Action
CF	Exit code=0	Set file state to **YES**.
	Exit code=nonzero	Set file state to **NO**.
	%UT [*errormessage*] and Exit code=2	Set file state to **NO**.

A method that does not support the **CF** task writes a **%UT** message to **stdout** and exits with an exit code of **2**.

Get status task (GS) extended agents only

The **GS** task tells the extended agent's method to check the status of a job. This is necessary when another job is dependent on the successful completion of an external job. Before running the method, the **GSuser** parameter is read from the method options file to determine the user account with which to run the method. If the parameter is not present or the options file does not exist, on UNIX the **root** user is used, and, on Windows, the user name of the account in which Tivoli Workload Scheduler was installed is used. If the method cannot be run, the dependent job or job stream is not allowed to run. If a *jobid* is available from a prior **GS** task, it is passed to the method.

The method checks the state of the specified job, and returns it in a **%CJ** message written to **stdout**. It then exits with an exit code of zero. At a rate set by the ***bm check status*** local option, the method is re-run with a **GS** task until one of the following job states is returned:

abend	The job ended abnormally.
succ	The job completed successfully.
cancl	The job was cancelled.
done	The job is ended, but its success or failure is not known.
fail	The job could not be run.
error	An error occurred in the method while checking job status.
extrn	The job check failed or the job status could not be determined.

Note that **GStimeout** in the method options file specifies how long Tivoli Workload Scheduler will wait for a response before killing the method. See "Method options file" on page 541 for more information.

Method responses are summarized in Table 89:

Table 89. Get status task (GS) messages

Task	Method Response	Tivoli Workload Scheduler Action
GS	**%CJ** *state* [*jobid*]	Sets job state to *state* and includes *jobid* in any subsequent **GS** task.
	%UT [*errormessage*] and Exit code=2	Job state is unchanged.

A method that does not support the **GS** task writes a **%UT** message to **stdout** and exits with an exit code of **2**.

Cpuinfo command for extended agents only

The **cpuinfo** command can be used in an access method to return information from a workstation definition. See "Cpuinfo command for extended agents only" for complete command information.

Troubleshooting

The following topics are provided to help troubleshoot and debug extended agent and access method problems.

Job standard list error messages

All output messages from an access method, except those that start with a percent sign (%), are written to the job's standard list (**stdlist**) file. For **GS** and **CF** tasks that are not associated Tivoli Workload Scheduler jobs, messages are written to Tivoli Workload Scheduler standard list file. For information about a problem of any kind, check these files.

Method not executable

If an access method cannot be run, the following occurs:

- For **LJ** and **MJ** tasks, the job is placed in the FAIL state.
- For the **CF** task, the file dependency is unresolved and the dependent job remains in the HOLD state.
- For the **GS** task, the job dependency is unresolved and the dependent job remains in the HOLD state.

To get more information, review the standard list files (**stdlist**) for the job and for Tivoli Workload Scheduler.

Console Manager messages for extended agents only

This error message is displayed if you issue a **start**, **stop**, **link**, or **unlink** command for an extended agent:

```
AWSBHU058E The command issued for workstation: workstation_name,
           cannot be performed, because the workstation is an extended agent,
           where the command is not supported.
```

Composer and compiler messages for extended agents only

The following error messages are generated when **composer** encounters invalid syntax in a workstation definition:

```
AWSDEM045E There is an error in the workstation definition. The ACCESS keyword
           was not followed by a valid method. Valid methods correspond with
           the name of a file in the TWS_home/methods directory
           (the file need not be present when the access method is defined).
AWSDEM046E There is an error in the workstation definition. The ACCESS keyword
           has been specified more than once.
AWSDEM047E There is an error in the workstation definition. The ACCESS keyword
           was not followed by a valid method. Valid methods correspond with
           the name of a file in the TWS_home/methods directory
           (the file need not be present when the access method is defined).
```

If an extended agent is defined with an access method but without a host, the following message is displayed:

```
AWSBIA140E For an extended agent you must specify the host and the access method.
```

Jobman messages for extended agents only

For extended agents, error, warning, and information messages are written to **jobman**s **stdlist** file.

A successful job launch generates the following message:

```
AWSBDW019I Launched job job_name, #Jrun_number for user user_ID.
```

Failure to launch a job generates the following message:

```
AWSBDW057E The job  job_name was not launched for this reason:
           error_message
```

Failure of a check file task generates the following message:

```
AWSBDW062E Jobman was unable to invoke the following method file method_name
           for the extended agent. The operating system error is:
           system_error
```

Failure of a manage job task generates the following message:

```
AWSBDW066E Planman has asked jobman to run a task that is not supported on the
           targeted agent. The following method options file was used:
           method_options_file. The job identifier and monitor PID are as
           follows: job, #Jmonitor_pid
```

When a method sends a message to **jobman** that is not recognized, the following message is generated:

```
AWSBDW064E A job that jobman is monitoring has returned the following
           unrecognizable message: incorrect_message. The job identifier,
           monitor PID and method file are as follows: job_name, #Jmonitor_pid
           using method file.
```

Chapter 18. Managing internetwork dependencies

Tivoli Workload Scheduler ***internetwork dependencies*** allow jobs and job streams in the local network to use jobs and job streams in a remote network as *follows* dependencies. This chapter describes how to customize your environment to be able to define internetwork dependencies and how to manage the internetwork dependencies.

The chapter is divided into the following sections:

- "Internetwork dependencies overview"
- "Configuring a network agent" on page 551
- "Defining an internetwork dependency" on page 553
- "Managing internetwork dependencies in the plan" on page 553
- "Internetwork dependencies in a mixed environment" on page 556

Note: Depending on your needs and requirements, you can choose between internetwork dependencies and cross dependencies to establish a dependency between a job running on the local engine and a job running on a remote Tivoli Workload Scheduler engine. See "Defining dependencies" on page 14 for a description about the differences between these two types of dependencies.

Internetwork dependencies overview

Before you specify an internetwork dependency, you must create a workstation definition for the ***network agent***. A network agent is a Tivoli Workload Scheduler workstation that handles follows dependencies between its local network and a remote Tivoli Workload Scheduler network.

In the local Tivoli Workload Scheduler network there can be more than one network agent, each representing a specific Tivoli Workload Scheduler remote network where jobs and job streams referring to locally defined internetwork dependencies are defined. Internetwork dependencies are assigned to jobs and job streams in the same way as local follows dependencies, with the exception that the network agent's name is included to identify the followed job or job stream.

A special job stream named ***EXTERNAL*** is automatically created by Tivoli Workload Scheduler for each network agent in the local network. It contains placeholder jobs to represent each internetwork dependency.

An `EXTERNAL` job is created for each internetwork dependency belonging to job streams planned to start in different days with different schedule dates. This means that an `EXTERNAL` job differs from another one by:

- The script file name, which identifies the remote job or job stream the local job or job stream is dependent on.
- The date the local job stream containing the internetwork dependency is planned to start. If the dependency is defined in a job within the job stream the date the job stream is planned to start is taken into account.

The check of the internetwork dependency check does not start until the job stream matches its time dependency or it is released.

In case of two jobs belonging to different job streams and referring to the same internetwork dependency, as one of their job streams is released and the job starts the internetwork dependency is checked and possibly released. In this case when the second job starts to check its internetwork dependency it finds the dependency already solved but not necessarily on the expected day. If you want to prevent this situation from occurring you must rerun the job representing the internetwork dependency after it is solved the first time.

Tivoli Workload Scheduler checks the status of the referred jobs and job streams in the remote network and maps their status in the jobs representing the internetwork dependencies in the `EXTERNAL` job stream. The status of these jobs and job streams is checked over a fixed time interval until the remote job or job stream reaches the `SUCC`, `CANCL`, or `ERROR` state.

Understanding how an internetwork dependency is shown

This section describes a sample scenario about internetwork dependencies and how to link the job representing the internetwork dependency to the job stream where the dependency is defined. Assume that:

- You defined a job stream named `ELISCHED` running on workstation `TWS206` containing a job named `ELI` with an internetwork dependency from a job stream `TWS207#FINAL.MAKEPLAN` running in a different Tivoli Workload Scheduler network.
- `XA_MAST` is the network agent defined in the local network to manage internetwork dependencies from jobs and job streams defined in that remote network.

Use the **conman sj** command to see the internetwork dependency set:

```
                                  (Est) (Est)
CPU    Schedule SchedTime Job State Pr Start Elapse RetCode Deps

TWS206#ELISCHE 0600 03/31 **** HOLD 10 (03/31)
                          ELI  HOLD 10                    XA-MAST::"TWS207#MYJS.JOB1"
```

where `(03/31)` represents the **at** time restriction set in `TWS206#ELISCHE`. Starting from `(03/31)` the status of `TWS207#MYJS.JOB1` is checked in the remote network to see if the internetwork dependency `XA-MAST::"TWS207#MYJS.JOB1"` is satisfied.

If you run the command:

```
%sj XA-MAST#EXTERNAL;info
```

you see the list of jobs representing internetwork dependencies defined in jobs and job streams running in the local network from jobs and job streams defined in the remote network reachable through the network agent `XA-MAST`:

```
CPU      Schedule SchedTime  Job      JobFile                Opt  Job
 Prompt
XA-MAST #EXTERNAL
                             E8802332 TWS207#MYJS.JOB1
```

You can see the details about the job or job stream depending on `TWS207#MYJS.JOB1` in the internetwork dependency represented by job `E8802332` in the `EXTERNAL` job stream, by running the following command:

```
%sj @#EXTERNAL.E8802332;deps
```

The output shows the link between the dependent job and the internetwork dependency:

```
                                       (Est) (Est)
 CPU    Schedule SchedTime Job State Pr Start Elapse RetCode Deps

XA-MAST#EXTERNAL.E8802332 Dependencies are:


 TWS206#ELISCHE 0600 03/31 **** HOLD 10 (03/31)
                           ELI  HOLD 10                  XA-MAST::"TWS207#MYJS.JOB1"
```

The internetwork dependency check does not start until the job stream `TWS206#ELISCHE` matches its time dependency, `(03/31)`, or is released.

If there is another job defined within another job stream in the local network that has a dependency on `TWS2007#MYJS.JOB1` and the local job stream is planned to start on the same day, `03/31/06`, then also the dependency of this other job on `TWS2007#MYJS.JOB1` will be listed in the job `E8802332` within the `XA-MAST#EXTERNAL` job stream.

Configuring a network agent

Network agent workstations are defined as extended agents and require a hosting physical workstation and an access method. The access method for network agents is named **netmth**.

The **batchman** process on the master domain manager queries the **netmth** on the network agent at fixed time intervals to get the status of the remote predecessor job or job stream. You can customize the time interval between two consecutive checks by setting the global option *bm check status* in the `localopts` file on the master domain manager. The Tivoli Workload Scheduler continues checking until the remote job or job stream reaches the `SUCC`, `CANCL`, or `ERROR` state.

An options file named **`netmth.opts`** is created on the workstation where the network agent runs. In this file are defined the user under which the access method runs and the time to wait to get a response from the access method before shutting it down. This options file must have the same path as the access method:

```
TWS_home/methods/netmth.opts
```

The content of the `netmth.opts` file has the following structure:

```
GSuser=login_name
GStimeout=seconds
```

where:

login_name
: Is the login used to run the method. If the network agent's host is a Windows computer, this user must be defined in Tivoli Workload Scheduler.

seconds
: Is the number of seconds, Tivoli Workload Scheduler waits for a response before shutting down the access method. The default setting is 300 seconds. The next time **batchman** needs to check the status of the remote predecessor job or job stream, the access method starts up automatically.

Changes to this file do not take effect until you stop and restart Tivoli Workload Scheduler.

A sample network agent definition

The following example shows how to define a network agent workstation for a remote network, `Network A`, that allows local network, `Network B`, to use jobs and job streams in the remote network as internetwork dependencies.

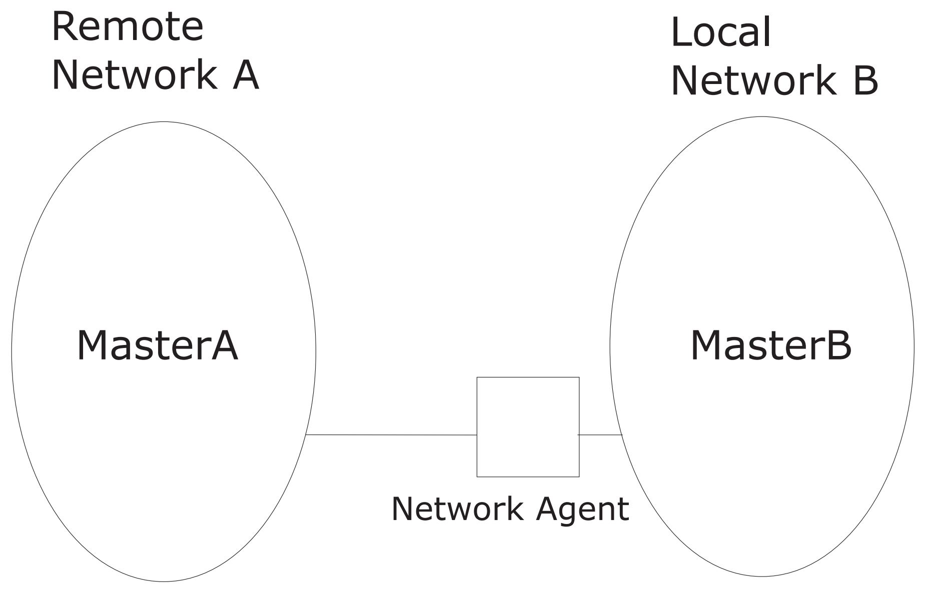


Figure 29. Local and remote networks

Assuming that:

- `MasterA` is the master domain manager of the remote network, `Network A`, and that:
  - **tws_masterA** is the *TWS_user* defined on `MasterA`.
  - The TCP port number for `MasterA` as 12345.
  - The node where `MasterA` is defined is `MasterA.rome.tivoli.com`.
- `MasterB` is the master domain manager of the local network, `Network B`, and that:
  - **tws_masterB** is the *TWS_user* defined on `MasterB`.
  - The node where `MasterB` is defined is `MasterB.rome.tivoli.com`.

A network agent workstation named `NetAgt`, defined on `MasterB` to manage internetwork dependencies on jobs or job streams defined in `Network A` can be the following:

```
CPUNAME NETAGT
  DESCRIPTION "NETWORK AGENT"
  OS OTHER
  NODE MASTERA.ROME.TIVOLI.COM
  TCPADDR 12345
  FOR maestro
    HOST MASTERB
    ACCESS netmth
END
```

Important: Write the network access name `netmth` in lowercase on case-sensitive operating systems.

The options file, **`netmth.opts`** defined on `MasterB` can be:

```
GSuser=tws_masterB
GStimeout=600
```

Note: The network agent can be defined on either the master domain manager or a fault-tolerant agent.

Defining an internetwork dependency

The syntax used to specify an internetwork dependency within a job stream definition is the following:

```
follows  Network_agent_name::remote_workstation#jobstreamname(time [date]).jobname
```

where the (*time* [*date*]) are specific to the time zone used on the workstation of the remote network the network agent is connected to; in our sample the time zone of MasterA. If (*time* [*date*]) is not specified in this syntax or if there is more than one job stream with the same (*time* [*date*]), the first job stream found is taken into account.

Assuming that:

- schedA is a job stream defined in the MasterA database.
- jobA is a job defined in the MasterA database.
- schedB is a job stream defined in the MasterB database.
- jobB is a job defined in the MasterB database.

you can define internetwork dependencies using the following **follows** statements:

To define an internetwork dependency of schedB from the job stream instance schedA(1100)

Use the following statement:

```
schedule schedB
      on everyday
      follows NetAgt::MasterA#schedA(1100)
      :
end
```

To define an internetwork dependency of jobB from jobA contained in the job stream instance schedA(1100)

Use the following statement:

```
schedule schedB
      on everyday
      :
      jobB
           :
           follows NetAgt::MasterA#schedA(1100).jobA
           :
       end
       :
end
```

You can also define internetwork dependencies of a job on a job stream or a job stream on a job.

Managing internetwork dependencies in the plan

Internetwork dependencies are managed in the plan from the **conman** command line by managing the EXTERNAL job stream. Within the EXTERNAL job stream the internetwork dependencies are listed as jobs regardless of whether they are defined for jobs or job streams. There is an EXTERNAL job stream for every network agent in the plan.

Within the `EXTERNAL` job stream, unique job names representing internetwork dependencies are generated as follows:

```
Ennnmmss
```

where:

nnn Is a random number.

mm Is the current minutes.

ss Is the current seconds.

The actual name of the job or job stream is stored in the script files specification of the job record.

Note: Remote jobs and job streams are defined and run on their local network in the standard manner. Their use as internetwork dependencies has no effect on their local behavior.

States of jobs defined in the EXTERNAL job stream

The status of the jobs defined in the `EXTERNAL` job stream is determined by the access method and listed in the Release Status field of the `EXTERNAL` job stream. The reported status refers to the last time the remote network was checked. Jobs might appear to skip states when states change in between two different checks.

All states for jobs and job streams, except `FENCE`, are listed. In addition to these there are two states that are specific to the `EXTERNAL` jobs, they are:

ERROR
An error occurred while checking for the remote status.

EXTRN
The initial state. If the job is not found in the remote network the `EXTERNAL` job state remains `EXTRN`. If it isan `EXTERNAL` job it returns to state `EXTRN`.

Note: If you cancel in the local network the instances of jobs or job streams dependent on the same instance of a job or job stream defined in a remote network, make sure you manually cancel also the job, representing that internetwork dependency in the `EXTERNAL` job stream, to prevent the `EXTERNAL` job stream from being continuously carried forward.The same consideration applies when the local job stream dependent on the job or job stream defined in the remote network is not carried forward to the new plan.

Working with jobs defined in the EXTERNAL job stream

These are the available actions you can perform against jobs in an `EXTERNAL` job stream:

Cancel
Cancels the `EXTERNAL` job, releasing the internetwork dependency for all local jobs and job streams. The status of the dependency is no longer checked.

Rerun Instructs **conman** to restart checking the state of the `EXTERNAL` job. The job state is set to `EXTRN` immediately after a **rerun** is performed.

Rerun is useful for `EXTERNAL` jobs in the `ERROR` state. For example, if an `EXTERNAL` job cannot be launched because the network access method does not grant execute permission, the job enters the `ERROR` state and its status ceases to be checked. After you correct the permissions, the method can start but **conman** will not start checking the `EXTERNAL` job state until you perform a **rerun**.

Confirm SUCC / ABEND
Sets the status of the `EXTERNAL` job to `SUCC` or `ABEND`, releasing the dependency for all depending local jobs and job streams. The status of the dependency in no longer checked.

Note: None of these commands has any effect on the remote job or job stream in the remote network. They simply manipulate the dependency for the local network.

Sample internetwork dependency management scenarios

This section provides sample scenarios describing how you can manage internetwork dependency in production using the **conman** command line commands.

Assuming that you have already defined the following:
- A local workstation called `local1`
- A job stream defined for the local workstation `local1` called `sched1`
- A job defined in `local1#sched1` called `job1`
- A network agent called `netagt` defined in the local network to manage internetwork dependency from jobs and job streams defined in the remote network.
- A workstation in the remote network called `remote1`
- A job stream defined for the remote workstation `remote1` called `rcshed`
- A job in defined in `remote1#rsched` called `rjob`

You can perform the following actions from the **conman** command line in the local network:

Adding an internetwork dependency from a remote job to a local job.
For example, to add the remote job `rjob` as an internetwork dependency for `job1`, run the following command:

```
adj local1#sched1.job1;follows=netagt::remote1#rsched.rjob
```

Adding an internetwork dependency from a remote job stream to a local job stream.
For example, to add the remote job stream `rsched` as an internetwork dependency for job stream `sched1`, run the following command:

```
ads local1#sched1;follows=netagt::remote1#rsched
```

Cancelling internetwork dependencies managed by a network agent.
For example, to cancel all `EXTERNAL` jobs for a network agent `netagt`, run one of the following two commands:

```
cj netagt#EXTERNAL.@

cj netagt::@
```

Confirming the successful completion of an internetwork dependency.
For example, to confirm that the remote job `remote1#rsched.rjob` completed successfully and so release the corresponding internetwork dependency, run the following command:

```
confirm netagt::remote1#rsched.rjob;succ
```

Deleting an internetwork dependency from a job for a job.

For example, to delete the internetwork dependency from the remote job `remote1#rsched.rjob` for the local job `local1#sched1.job1`, run the following command:

```
ddj local1#sched1.job1;follows=netagt::remote1#rsched.rjob
```

Deleting an internetwork dependency from a job for a job stream.

For example, to delete the internetwork dependency from the remote job `remote1#rsched.rjob` for the local job stream `local1#sched1`, run the following command:

```
dds local1#sched1;follows=netagt::remote1#rsched.rjob
```

Releasing a local job from an internetwork dependency from a remote job.

For example, to release a job from an internetwork dependency from a remote job, run the following command:

```
rj local1#sched1.job1;follows=netagt::remote1#rsched.rjob
```

Releasing a local job stream from an internetwork dependency from a remote job.

For example, to release a job stream from an internetwork dependency from a remote job, run the following command:

```
rs local1#sched1;follows=netagt::remote1#rsched.rjob
```

Rerunning a job in the `EXTERNAL` job stream to restart checking a dependency.

For example, to rerun a job belonging to the `EXTERNAL` job stream to restart checking the internetwork dependency from the remote job `remote1#rsched.rjob`, run one of the following two commands:

```
rr netagt#EXTERNAL.rjob

rr netagt::remote1#rsched.rjob
```

Displaying internetwork dependencies from jobs and job streams defined in a remote network.

For example, to display all the internetwork dependencies defined for a network agent with their original names and their generated job names, run the following command:

```
sj netagt#EXTERNAL.@;info
```

Submitting a job with an internetwork dependency from a job stream defined in a remote network

For example, to submit a `rm` command into the `JOBS` job stream with an internetwork dependency on a remote job stream, run the following command:

```
sbd "rm apfile";follows=netagt::remote1#rsched
```

Internetwork dependencies in a mixed environment

Table 90 on page 557 shows the supported configuration for internetwork dependencies defined in a mixed version 8.3 environment. The key to the table is as follows:

Net_A The network agent defined in the local network.

Wks_B

The workstation in the remote network that the network agent `Net_A` is connected to. `Wks_B` is the workstation that identifies and checks the state of the remote job or job stream specified in the internetwork dependency.

Sym_A

The `Symphony` file processed in the local network.

Sym_B
The Symphony file processed in the remote network.

back-level
Version 8.1, 8.2, or 8.2.1

Table 90. Internetwork dependencies in a mixed environment

	Net_A back-level Sym_A back-level	**Net_A 8.3 Sym_A back-level**	**Net_A back-level Sym_A 8.3**	**Net_A 8.3 Sym_A 8.3**
Wks_B back-level Sym_B back-level	This is not a mixed version 8.3 environment.	Net_A sends the information to Wks_B as if it had the same version as Wks_B.	Net_A sends the information to Wks_B in 8.1, 8.2, or 8.2.1 format. The use of the *schedtime* keyword in the job definition is not supported.	Net_A sends the information to Wks_B as if it had the same version as Wks_B. If defined, the *schedtime* keyword in the job definition is automatically removed by Net_A.
Wks_B 8.3 Sym_B back-level	Wks_B works as if it had the same version as Net_A.	Net_A sends the information to Wks_B. If defined, the *schedtime* keyword in the job definition is automatically removed by Wks_B.	Net_A sends the information to Wks_B. If defined, the *schedtime* keyword in the job definition is automatically removed by Wks_B.	Net_A sends the information to Wks_B. If defined, the *schedtime* keyword in the job definition is automatically removed by Wks_B.
Wks_B back-level Sym_B 8.3	Not supported.	Not supported.	Not supported.	Not supported.
Wks_B 8.3 Sym_B 8.3	Not supported.	Not supported.	Net_A sends the information to Wks_B. If defined, the *schedtime* keyword is parsed by Wks_B.	This is a version 8.3 environment.

Chapter 19. Defining and managing cross dependencies

Tivoli Workload Scheduler ***cross dependencies*** help you to integrate and automate job processing when:

- The workload is spread across different scheduling environments, because some of the activities run at different sites or involve different organizational units or require different skills to be run.
- Even if most of the batch workload is managed locally, none of these environments is completely isolated from the others because they frequently interoperate to exchange or synchronize data and activities.

More specifically, the cross dependency feature is key when you need to synchronize activities between different scheduling environments in an easy way so that you can:

- Define in one scheduling environment dependencies on batch activities that are managed by another scheduling environment.
- Monitor the status of the remote predecessor jobs as if they were running in your local environment.

Additionally, you can control the status of these dependencies by navigating from a unique user interface across the different scheduling environments.

This chapter describes how you define and use cross dependencies.

It contains the following sections:

- "An introduction to cross dependencies"
- "Processing flow across the distributed scheduling environment" on page 561
- "Defining a cross dependency" on page 563
- "How the shadow job status changes until a bind is established" on page 565
- "How a z/OS shadow job is bound" on page 568
- "How the shadow job status changes after the bind is established" on page 571

Note: Depending on your needs and requirements, you can choose between internetwork dependencies and cross dependencies to establish a dependency between a job running on the local engine and a job running on a remote Tivoli Workload Scheduler engine. See "Defining dependencies" on page 14 for a description of the differences between these two types of dependencies.

An introduction to cross dependencies

A cross dependency is, from a logical point of view, a dependency that a local job has on a job instance that is scheduled to run on a remote engine.

Use cross dependencies to integrate the workload running on different engines, which can be Tivoli Workload Scheduler for z/OS engines (controller) or Tivoli Workload Scheduler engines (master domain manager and backup master domain manager).

The following objects and terms are used to describe and implement cross dependencies:

Remote engine workstation
A new type of workstation that represents locally a remote Tivoli Workload Scheduler engine, either distributed or z/OS. This type of workstation uses a connection based on HTTP or HTTPS protocol to allow the local environment to communicate with the remote environment.

Remote job
A job scheduled to run on a remote Tivoli Workload Scheduler engine.

Shadow job
A job defined locally, on a remote engine workstation, which is used to map a remote job. The shadow job definition contains all the information necessary to correctly match, in the remote engine plan, the remote job instance.

Bind The process to associate a shadow job with a remote job instance scheduled in the remote Tivoli Workload Scheduler engine plan.

From a logical point of view in the local environment:
- The remote engine workstation is used to map the remote Tivoli Workload Scheduler engine.
- The shadow job, defined on that remote engine workstation, is used to map a remote job instance scheduled to run on that remote Tivoli Workload Scheduler engine.

You define a cross dependency when you want that a *local job* (running on your local engine) depends on a *remote job* (running on a remote engine).

To do it, you must do as follows:
1. Create a *shadow* job that runs on your local engine.
2. Define a normal dependency that makes your *local* job dependent on the *shadow* job.

When you create the shadow, consider that
- It must be defined on a workstation of remote engine type, which points to the remote engine (that is the engine where the remote job is scheduled to run).
- You must make it point to the remote job with which you are creating the cross dependency.

Figure 30 on page 561 shows the logical flow implementing cross dependencies:
1. In the bind process, the shadow job is associated to the remote job instance.
2. After the bind is established, the shadow job status is updated according to the remote job status transition.
3. When the shadow job status becomes SUCC the normal dependency of the local job is released, and so also the cross dependency of that local job on the remote job is also released.

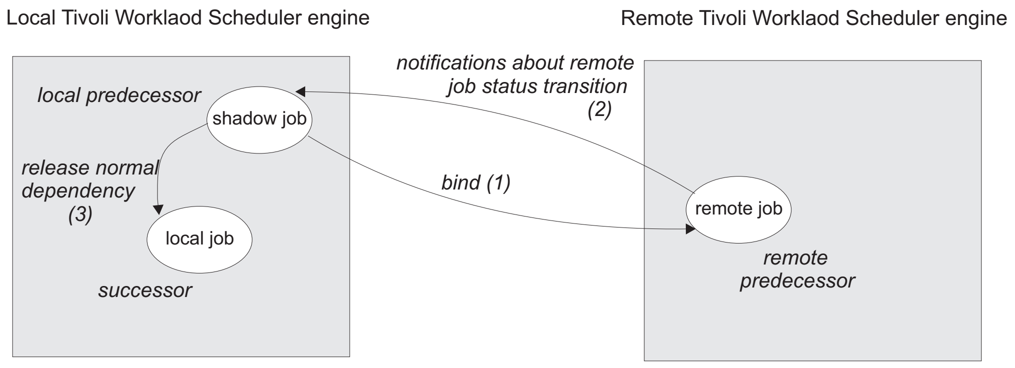

Local Tivoli Worklaod Scheduler engine
Remote Tivoli Worklaod Scheduler engine
notifications about remote job status transition (2)
local predecessor
shadow job
release normal dependency (3)
bind (1)
remote job
local job
remote predecessor
successor

Figure 30. Cross dependency logic

Processing flow across the distributed scheduling environment

Depending on whether the local engine emits or receives a bind request, the processing flow and the components involved change. In both cases, the broker workstation in the local environment must be up and running to allow the bind requests management.

Processing flow when the local engine sends a bind request to a remote engine

When you define a shadow job, you specify the information needed to establish a bind with a job in the remote engine plan.

When the shadow job scheduled time arrives, if the shadow job is free from dependencies, it is selected by the local **batchman** for submission and its status is set to INTRO.

The bind request is sent to the remote engine. The shadow job status is set to WAIT.

As soon as the bind processing completes, the remote engine sends back to the local engine a notification with the bind result.

Table 91 shows how the shadow job status changes based on:
- Whether the instance to bind exists or not in the remote engine plan.
- The status of the remote job bound.

Table 91. Shadow job status transition

Status of the shadow job in the production plan:	When on the remote engine:
BOUND	**z/OS** The remote job stream instance for the bind was found in the long term plan or in the current plan. **Distributed** The remote job stream instance for the bind was found in the preproduction plan.

Table 91. Shadow job status transition (continued)

Status of the shadow job in the production plan:	When on the remote engine:
ERROR	**z/OS** One of the following situations occurred: • The remote job stream instance for the bind exists neither in the long term plan nor in the current plan. • The remote job stream instance for the bind was found in the long term plan but, when it is included in the current plan, it does not contain the requested job instance. **Distributed** One of the following situations occurred: • The remote job stream instance to bind does not exist in the preproduction plan. • The remote job stream instance for the bind was found in the preproduction plan but, when it is included in the production plan, it does not contain the requested job instance. • The remote bind user is not authorized to access the requested job instance in the production plan.
EXEC	The status of the remote job is EXEC.
SUCC	The status of the remote job is SUCC.
FAIL	The status of the remote job is FAIL.
ABEND	The status of the remote job is ABEND.
SUCC	The status of the remote job is CANCELED.

Note: The status of the shadow job is FAIL also when its submission failed.

For more details about the shadow job status transition, see "How the shadow job status changes until a bind is established" on page 565 and "How the shadow job status changes after the bind is established" on page 571.

Processing flow when the remote engine receives a bind request from the local engine

When the remote engine receives a bind request from the local engine, the information contained in the request is used to run the bind in the remote preproduction plan.

The bind request also contains an ordered list of URLs that the remote engine uses to send notifications to the local engine. If the local engine is distributed, the list is made up of the URL specified in the JDURL property of the file named `*TDWB_HOME*/config/JobDispatcherConfig.properties`.

Note: By default the Tivoli Workload Scheduler uses the TWSUser to run the bind in the production plan. If you want to limit and control which jobs can be bound, you can specify a different user using the global option **bindUser**. The user specified does not need to be defined as a user on the operating system, or even have a password, but it must exist as entry in the security file with the following access privileges:

- DISPLAY access to the *job* and *schedule* objects that can be bound
- LIST access to the *job* objects that can be bound. This access is required only if the global option `enListSecChk` is set to `yes`.

If the required access privileges are not specified, a notification with an error is sent back to the engine that requested the bind.

The remote engine sends back to the local engine:

A notification with the status BOUND
If the preproduction plan contains at least one instance of the job stream specified in the bind request and the definition of that job stream contains the job to bind.

A notification with the status of the job instance bound
If the instance of the job to bind is found in the production plan and whenever its status changes.

A notification with an error
If the job instance to bind is not found or if the bind user is not authorized.

The remote **batchman** process writes an entry in the `PlanBox.msg` queue whenever the status of a remote job changes.

Every 30 seconds, the `PlanBox.msg` queue is scanned for new entries that document a change in the status of remote jobs that were bound. Whenever a status change is found, a notification containing the status of the remote job bound is sent back to the engine that requested the bind.

Note: To change the polling interval, specify a value, in seconds, for **com.ibm.tws.planner.monitor.checkPlanboxInterval** in the file `*WAS_HOME*/profiles/*profile_name*/properties/TWSConfig.properties` and then restart the WebSphere Application Server.

Defining a cross dependency

Perform these steps to define a cross dependency between a job running in your environment and another job running on a different Tivoli Workload Scheduler engine:

1. Create a remote engine workstation

   Create a remote engine workstation for a specific remote engine when you need to define dependencies on job instances running on that remote engine. On a remote engine workstation you can run only shadow jobs.

   As a best practice, if the remote Tivoli Workload Scheduler engine is distributed, you can define a dynamic pool containing an ordered list of remote engine workstations pointing to the remote master and to its backup masters, to ensure that failover and switch manager capabilities are applied. For more information about workstations pool, see “Workstation” on page 4.

Note: It is recommended that:

- All the distributed environments involved have the timezone feature enabled. For more information, see "Enabling time zone management" on page 531.
- You specify as TIMEZONE property of the remote engine workstations the timezone set on the operating system of the remote Master Domain Managers or Backup Master Domain Managers they point to.

For more information about the specific settings to use when defining a remote engine workstation, see "Workstation definition" on page 127.

2. Define a shadow job running on the remote engine workstation

   Create a shadow job pointing to a specific job instance defined on a remote engine when you want to track in your local environment the status of that remote job and define cross dependencies on that remote job.

   On Tivoli Workload Scheduler distributed environments, you can use alias for job stream names and job names. If you are defining a distributed shadow job, make sure that:

   - The remote job stream name specified, contains the job stream name as it is defined in the database.
   - The remote job name specified, contains the alias, if defined, of the remote job to bind.

   If you do not follow these guidelines, the bind fails and the status of the shadow job becomes ERROR.

   In the shadow job definition set COMPLETE_IF_BIND_FAILS in the *rccondsucc* field to specify if the shadow job status must be forced to SUCC or ERROR if the bind in the remote engine plan fails.

   For more information about the specific settings to use when defining a shadow job, see "Job definition" on page 145.

   Depending on whether the remote engine is z/OS or distributed, you can use different matching criteria:

   If the remote engine is distributed

   You can choose any of the these matching criteria:

Table 92. Matching criteria for distributed shadow jobs

On the Dynamic Workload Console	Corresponding keyword used in composer
Closest preceding	previous
Within a relative interval	relative from=*time_before_scheduled_time* to=*time_after_scheduled_time*
Within an absolute interval	absolute from=*interval_start* to=*interval_end*
Same scheduling date	sameDay

   For more information about these matching criteria, see "Managing external follows dependencies for jobs and job streams" on page 52.

   If the remote engine is z/OS based

   Closest preceding is the only matching criteria supported by Tivoli Workload Scheduler for z/OS.

   The scheduled time of the job stream instance containing the shadow job is used for the matching in the remote engine plan.

   The shadow job status transition is mapped to the remote job instance status transition.

3. Add a dependency on the shadow job

   You add the cross dependency for a local job on the remote job by defining a dependency for the local job on a shadow job that:

   - Points to the remote job instance.
   - Is defined on a local workstation that points to the remote engine where the remote job is defined.

   The cross dependency on the remote job instance is released when the local dependency on the shadow job is released.

Monitoring a cross dependency resolution in the production plan

Shadow jobs are added to the plan as follows:

- At run time if either of the following situations occurs:
  - A shadow job definition is submitted using the **sbj** command.
  - A job stream containing a shadow job definition is submitted using the **sbs** command.

  Note: When submitting a shadow job, specify a destination job stream with a known scheduled time to better control the remote job instance that will be bound.
- When the preproduction plan is extended or created and its time frame includes the shadow job scheduled time.

When a shadow job instance is added to the plan, you can start monitoring its status.

How the shadow job status changes until a bind is established

Figure 31 on page 566 summarizes how a shadow job status changes until the bind is established.

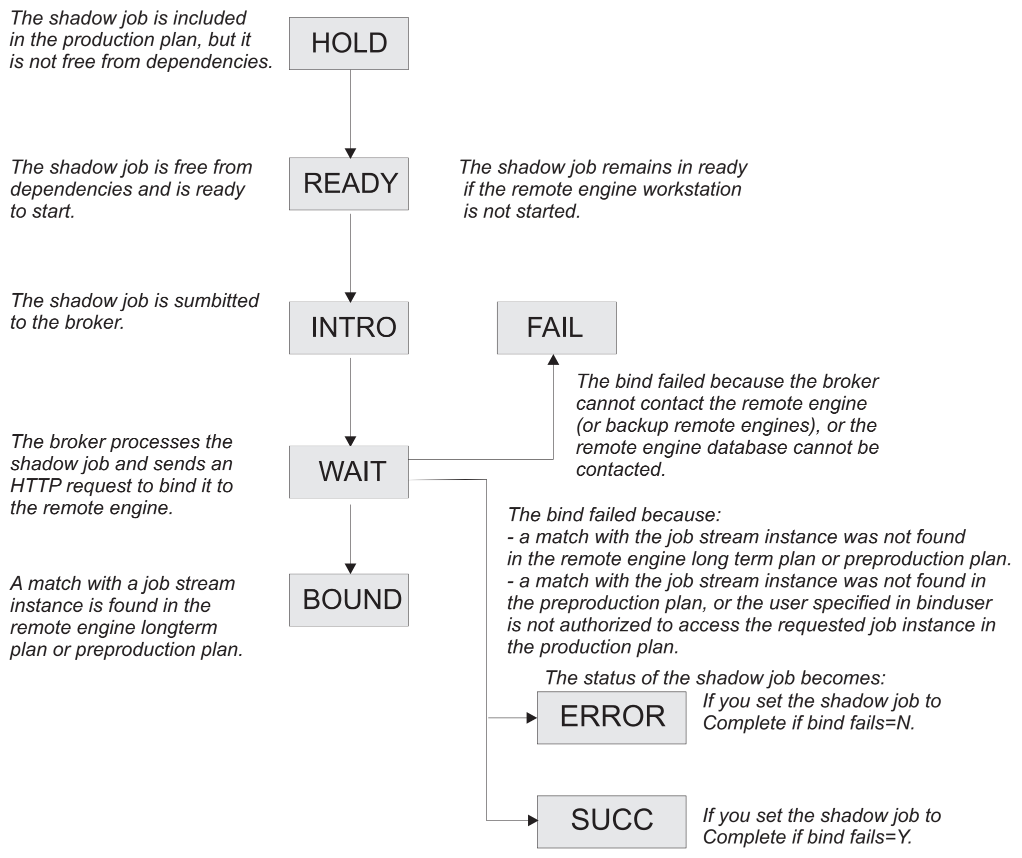

The shadow job is included in the production plan, but it is not free from dependencies.
HOLD
The shadow job is free from dependencies and is ready to start.
READY
The shadow job remains in ready if the remote engine workstation is not started.
The shadow job is sumbitted to the broker.
INTRO
FAIL
The bind failed because the broker cannot contact the remote engine (or backup remote engines), or the remote engine database cannot be contacted.
The broker processes the shadow job and sends an HTTP request to bind it to the remote engine.
WAIT
The bind failed because:
- a match with the job stream instance was not found in the remote engine long term plan or preproduction plan.
- a match with the job stream instance was not found in the preproduction plan, or the user specified in binduser is not authorized to access the requested job instance in the production plan.
A match with a job stream instance is found in the remote engine longterm plan or preproduction plan.
BOUND
The status of the shadow job becomes:
ERROR
If you set the shadow job to Complete if bind fails=N.
SUCC
If you set the shadow job to Complete if bind fails=Y.

Figure 31. Shadow job status transition until the bind is established

As for any other job, the initial status of the shadow job is HOLD and it turns to READY when the job becomes free from dependencies and is ready to start.

The scheduler then sends an HTTP request to the remote engine containing both the information to identify the shadow job in the local production plan and the information to uniquely identify the remote job instance to bind in the remote engine plan, including the matching criteria. The scheduler must also be notified about the status of the remote job instance bound.

The scheduler tries to contact the remote engine, at regular intervals, until a specific timeout expires. If, by then, the remote engine could not be reached, the shadow job status is set to FAIL. To change the timeout and the interval, specify a value, in seconds, for both MaxWaitingTime and StatusCheckInterval in the file `*TDWB_HOME*/config/ResourceAdvisorConfig.properties` and then restart the broker.

If the preproduction plan does not exist on the remote engine when the bind request is received, the distributed shadow job status remains WAIT until the preproduction plan generation is completed and the bind request is processed. This might happen, for example, when the preproduction plan is created again from scratch on the remote engine.

For more information on the reason why the shadow job status is FAIL , see "How to see why the shadow job status is FAIL" on page 572.

When the remote engine receives the HTTP request, it tries to identify the job stream instance to use for the bind in its plan; the preproduction plan if the remote engine is distributed or the long term plan if the remote engine is z/OS. The definition of the job stream must contain the definition of the remote job to bind.

For more information about how the match is made in a distributed remote engine plan, see “How a distributed shadow job is bound.”

For more information about how the match is made in a z/OS remote engine plan, see “How a z/OS shadow job is bound” on page 568.

How a distributed shadow job is bound

If the remote engine is a Tivoli Workload Scheduler master domain manager or backup master domain manager, the search for the remote job instance to bind is done in the preproduction plan. Distributed remote job instances, belonging to the JOBS or USERJOBS job streams, are not involved in the bind process. However, remote jobs that are moved to USERJOBS after binding continue to send status change notifications.

The matching interval, except for the closest preceding matching criteria that does not require interval calculation, is calculated on the remote engine using the settings specified in the distributed shadow job definition.

For example, when the **`sameDay`** matching criteria is specified, the day that is referred to is the day specified on the remote engine in terms of `[`*`startOfDay`*`,` *`startOfDay`*`+23:59]`.

When using an interval-based matching criteria, the HTTP request sent to the remote engine contains the following information to allow the remote engine to calculate the matching interval:

For absolute interval matching criteria:
The values *hhmm*, *HHMM* and, optionally, *d* and *D*, specified in the clause:

```
<dshadow:matching>
<dshadow:absolute from="hhmm [+/-d day[s]]" to="HHMM [+/-D day[s]]"/>
</dshadow:matching>
```

Boundary values are *hhmm* `-6 days` and *HHMM* `+6 days`.

The time zone used for the matching criteria is the time zone of the shadow job.

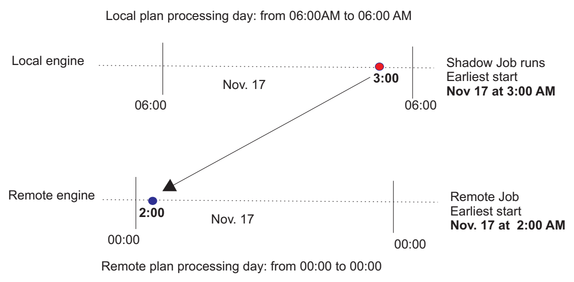
For relative matching criteria:
The shadow job scheduled time and the values [*hh*]*hmm* and [*HH*]*HMM* specified in the clause:

```
<dshadow:matching>
<dshadow:relative from="+/-[hh]hmm" to="+/-[HH]HMM"/>
</dshadow:matching>
```

Boundary values are +/-167:59 hours.

For example, to create a shadow job that matches a remote job instance whose Earliest start is November, 17 at 2:00 AM, you can specify either of the following matching criteria:

- **Same scheduled date**
- **Within an absolute interval** specifying as an offset: **1 day Before earliest start time.**



The remote job instance to match is identified on the remote engine according to the rules stated for the external follows dependencies. For more details about external follows dependencies resolution criteria, see "Managing external follows dependencies for jobs and job streams" on page 52.

For more information about defining shadow jobs, see "Job definition" on page 145.

How a z/OS shadow job is bound

If the remote engine is a Tivoli Workload Scheduler for z/OS controller, the search for the remote instance to bind is done as follows:

- First, the instance is searched in the Long Term Plan (LTP) in the part of the bind interval that follows the Current Plan (CP) end time and precedes the shadow job scheduled time.
- If no instance is found, the instance is searched in the CP in the part of the bind interval that precedes the current plan end.

Note: If the remote controller receives a bind request with a client notify URI that is not defined among the HTTP destinations, the bind request is discarded and the message EQQHT62W is logged in the MLOG.

The following sections describe the scenarios that can occur when binding a z/OS shadow job having:

- Scheduled time: 18:00
- Remote job information:
  - Application ID: **JS2**
  - Operation number: **OP2**

In the figures:

- The white box indicates the time interval covered by the LTP.
- The light grey box indicates the time interval covered by the CP.
- The dark grey box indicates the interval in the remote engine plan during which the job instance to bind must be searched.
- The **JS2** occurrence highlighted in bold is the instance selected for the bind.

Scenario 1: The CP interval contains the shadow job scheduled time and JS2 occurrences exist.

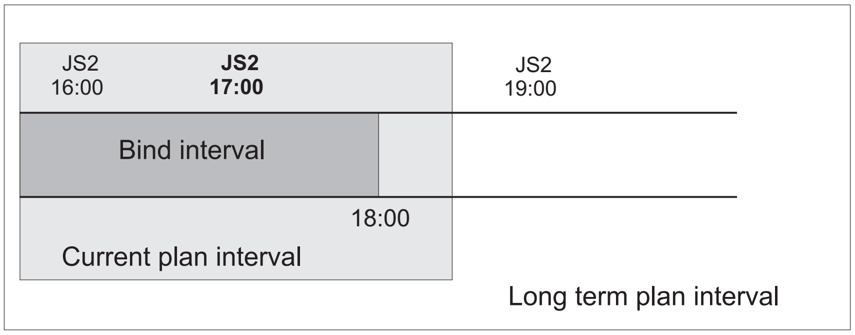


Figure 32. Instance to be bound if the shadow job scheduled time is included in the CP interval

Figure 32shows, highlighted in bold, the **JS2** instance that more closely precedes the shadow job scheduled time. This instance is selected for the bind because the scheduled time is contained in the CP. The shadow job and the remote job instance are associated. If, at a later time, a new instance of **JS2** that closest precedes the shadow job scheduled time is submitted ad hoc in the remote engine plan, the match with the **JS2** instance selected for the bind is *not* modified.

At this point, one of the following situations can occur:

The selected JS2 instance contains OP2.

The bind with **OP2** belonging to **JS2** is established and a notification containing:

- The remote job information identifying **OP2** instance in the remote engine plan
- The current status of that **OP2** instance

is sent back, the shadow job instance is updated with the remote job information, and its status is updated accordingly.

The selected JS2 instance no longer contains OP2 because either it was deleted and a daily plan removed it from the CP, or it was never contained in JS2.

The bind fails. A notification informing that the bind failed is sent back, and the shadow job status is updated according to what you set in the **Complete if bind fails** field.

The selected JS2 instance contains OP2 that was deleted but not yet removed from the CP.

The bind is established and a notification informing about the successful execution status is sent back. The shadow job instance is marked as SUCC. Its successors can start.

Scenario 2: The current plan interval contains the shadow job scheduled time, the JS2 instance that most closely precedes the shadow job scheduled time exists in the LTP but was canceled from the CP.

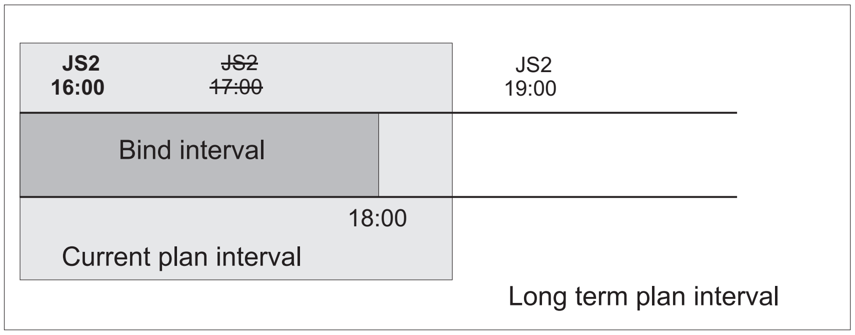


Figure 33. Instance to be bound if the instance that most closely precedes the shadow job scheduled time exists in the LTP but was canceled from the CP

Figure 33 shows, highlighted in bold, the **JS2** instance that is selected for the bind, because the occurrence that better matched was deleted.

The bind with **OP2** belonging to **JS2** is established and a notification containing:

- The remote job information identifying the **OP2** instance in the remote engine plan
- The current status of that **OP2** instance

is sent back, the shadow job instance is updated with the remote job information, and its status is updated accordingly.

Scenario 3: The CP interval contains the shadow job scheduled time but no JS2 occurrence exist.

Figure 34 shows that a **JS2** instance that closely precedes the shadow job

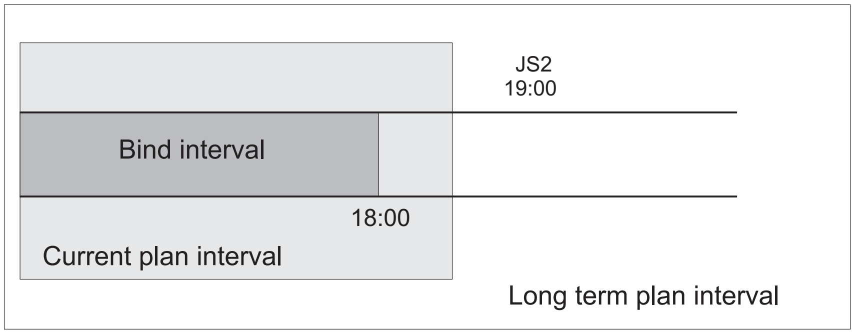


Figure 34. The scheduled time of the shadow job is included in the CP but no instance to bind exists

scheduled time does not exist.

The bind fails. A notification informing that the bind failed is sent back, and the shadow job status is updated according to what you set in the **Complete if bind fails** field.

Scenario 4: The LTP interval contains the shadow job scheduled time and the CP does not yet include the closest preceding JS2 instance.

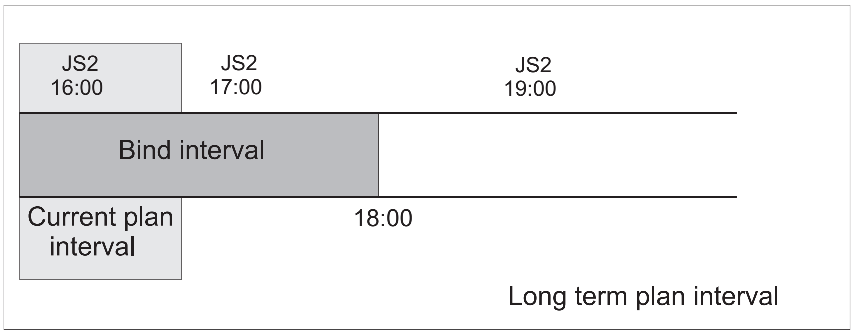


Figure 35. The instance to be bound exists but it is not yet included in the CP

Figure 35 shows the **JS2** instance that can be associated with the shadow job, even though the job **JOB2** is not yet in the CP.

A notification informing that the bind is established is sent back and the status of the shadow job is set to BOUND.

Scenario 5: The LTP interval still does not contain the shadow job scheduled time. Figure 36 shows that no **JS2** instance can be associated with the shadow

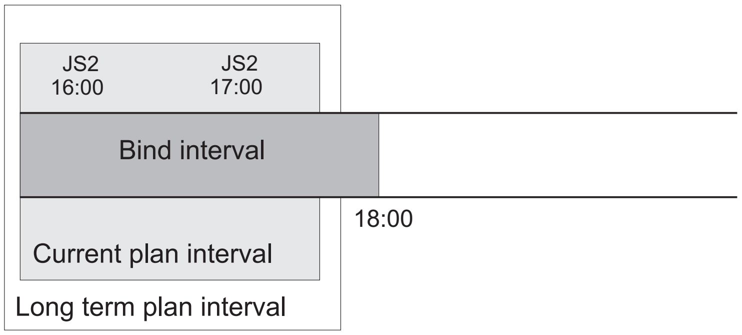


Figure 36. The LTP interval still does not contain the shadow job scheduled time

job because, until the LTP includes the shadow job scheduled time, closer preceding **JS2** instances can still be added.

In this case, the bind request is put in hold until the LTP is extended to include the shadow job scheduled time. Until then the status of the shadow job remains WAIT.

How the shadow job status changes after the bind is established

When a bind is established, the remote engine sends back an HTTP notification containing the status of the bind and, if the bind was successful, the information to identify the remote job instance bound. This information is shown in the shadow job instance details.

Depending on the type of a remote engine, the following information about the remote job instance is shown in the shadow job properties:

The remote engine type is distributed

- Job stream name
- Scheduled time
- Job stream workstation

- Job name

The remote engine type is z/OS

- Application ID
- Scheduled time
- Operation number
- Workstation
- Job name, if it was defined on the remote engine.

When the shadow job instance is mapped to an existing remote job instance, notifications about job status changes are sent asynchronously from the remote engine. These notifications are used to map remote job status transition to shadow job status transition. The store and forward mechanism ensures the delivery of the messages and the recovery in case of failures. Figure 37 shows how the status of a distributed shadow job changes, from when a bind is established until the shadow job status becomes SUCC or ERROR. Only status SUCC and ERROR are considered as the final status for a shadow job.

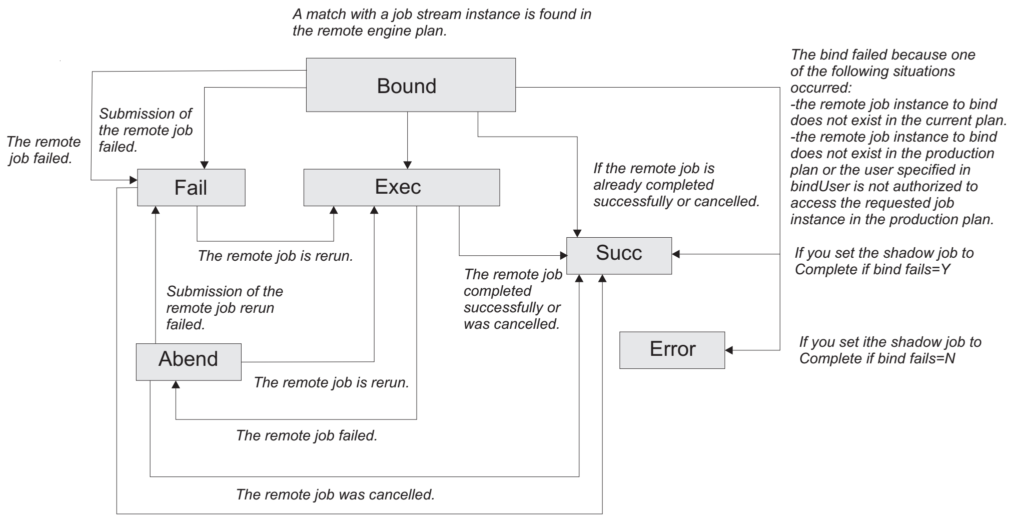

A match with a job stream instance is found in the remote engine plan.
Bound
The bind failed because one of the following situations occurred:
-the remote job instance to bind does not exist in the current plan.
-the remote job instance to bind does not exist in the production plan or the user specified in bindUser is not authorized to access the requested job instance in the production plan.
Submission of the remote job failed.
The remote job failed.
Fail
Exec
If the remote job is already completed successfully or cancelled.
Succ
If you set the shadow job to Complete if bind fails=Y
The remote job is rerun.
The remote job completed successfully or was cancelled.
Submission of the remote job rerun failed.
Abend
Error
If you set ithe shadow job to Complete if bind fails=N
The remote job is rerun.
The remote job failed.
The remote job was cancelled.

Figure 37. Shadow job status transition chain after the bind was established

If the remote job instance is already completed when the match is done, the shadow job status becomes SUCC immediately.

For more information on the reason why the shadow job status is FAIL , see "How to see why the shadow job status is FAIL."

When the shadow job status satisfies the dependency rule, the dependency of the local job on the shadow job is resolved, and the cross dependency for the local job on the remote job is also resolved.

How to see why the shadow job status is FAIL

The shadow job status can be FAIL in one of the following situations:

- The shadow job submission failed.
- The submission of the remote job failed.

To determine why the shadow job status is FAIL, see the log of the shadow job either by running the **`showjobs`** command with the `;stdlist` option, or by clicking **Job Log...** for the shadow job instance in the **Monitoring jobs** view on the Tivoli Dynamic Workload Console.

Shadow job status during the remote job recovery or rerun

After the bind is established it might happen that the remote job bound is rerun, in this case the status of the shadow job reflects the status of the rerun job. The shadow job status remains EXEC while the remote job recovery is in progress.

The shadow job status is updated only when the remote job reaches one of the following states:

ABEND
When the remote job fails to run.

SUCC When the remote job succeeds.

FAIL When the remote job submission fails.

You can see more details about the remote job in the shadow job properties. To see these details:

- Run the **conman** command **showjobs** with the **props** option against the shadow job.
- Access the shadow job properties panel in the Tivoli Dynamic Workload Console.

How carry forward applies to cross dependencies

Carry forward works the same way with shadow jobs as it does with other types of jobs. Shadow jobs in **WAIT** and **BOUND** status are treated just like jobs in **EXEC** status. Shadow jobs in **ERROR** status are treated like jobs in **FAIL** or **ABEND** status.

The status of a shadow job, bound to a remote job that is not carried forward, is set to ERROR when the remote production plan is extended.

Note: As a best practice, use cross dependencies with carry forward job streams on both local and remote distributed scheduling environments.

For more information about the **carryStates** global option, see the *Administration Guide*.

Managing shadow jobs in the production plan

Depending on the status of the shadow job, you can run the following commands:

Kill You can kill a shadow job with status BOUND, EXEC, or WAIT. The association established through the bind with the remote job is automatically canceled and the status of the shadow job is set to ABEND with return code 0.

Rerun You can rerun a shadow job with status ABEND, ERROR, SUCC, or FAIL. When you rerun a shadow job a new bind request is triggered.

Appendix A. Event-driven workload automation event and action definitions

This appendix documents the event and action providers you can use for event-driven workload automation and gives details on event and action definitions.

Event providers and definitions

This section gives details on the event types of the following event providers:
- `TWSObjectsMonitor`
- `FileMonitor`
- `TWSApplicationMonitor`

Datetime
: Contains both date and time. You can specify either one or both values in the filter.

Multiple filter predicates allowed
: You can specify multiple filter predicates for this property. The event will match the event condition if all the predicates are satisfied.

Multiple values allowed
: You can specify multiple values for this property within a single filter predicate. The filter will be satisfied when one of the values is matched.

Wildcard allowed
: Supported wildcards are asterisk (*) and question mark (?).

TWSObjectsMonitor events

`TWSObjectsMonitor` events are:
- JobStatusChanged
- JobUntil
- JobSubmit
- JobCancel
- JobRestart
- JobLate
- JobStreamStatusChanged
- JobStreamCompleted
- JobStreamUntil
- JobStreamSubmit
- JobStreamCancel
- JobStreamLate
- WorkstationStatusChanged
- ApplicationServerStatusChanged
- ChildWorkstationLinkChanged
- ParentWorkstationLinkChanged
- PromptStatusChanged
- AlertProduct

These events are generated by `batchman` (or `mailman` for the workstations) and written in a mailbox file named `monbox.msg`. The scheduling objects are monitored as follows:
- Jobs are monitored by the workstation where they run

- Job streams are monitored by the master domain manager
- Workstations monitor themselves
- Local prompts are monitored by the workstation running the job or job stream that have a dependency on the prompt
- Global prompts are monitored by the master domain manager

Click here to see the parameters of each event type.

Note: PDF users, the above parameter tables are an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

Working with WorkstationStatusChanged events

The event is sent when a workstation is started or stopped. But the following operational differences exist depending on the type of workstation that is monitored:

- For a fault-tolerant agent the event is sent when the workstation is started or stopped.
- For a dynamic workload broker workstation the event is sent also when it is linked or unlinked (as these commands also start or stop the workstation).
- For a dynamic pool workstation the event is never sent (even if the hosting dynamic workload broker is stopped) because there is no monitoring on this type of workstations.

Examples

The rule in the following example submits job stream `RJS_102739750` on workstation `NC125102` as soon as all the jobs of job stream `RCF_307577430` of workstation `NA022502` are in the RUNNING or SUCCESSFUL status.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="TWS_PLAN_EVENTS_JOB_STATUS_CHANGED" ruleType="filter" isDraft="no">
      <description>Event: Job Status Changed; Action: Submit job stream</description>
      <timeZone>Europe/Rome</timeZone>
      <validity from="2011-04-24" to="2012-04-24" />
      <activeTime start="00:00:00" end="12:00:00" />
      <eventCondition name="jobStatChgEvt1"
                      eventProvider="TWSObjectsMonitor"
                      eventType="JobStatusChanged">
      <scope>* # JOBSTREAMVALUE . * [RUNNING, SUCCESSFUL]</scope>
         <filteringPredicate>
          <attributeFilter name="JobStreamWorkstation" operator="eq">
         <value>NA022502</value>
          </attributeFilter>
         <attributeFilter name="JobStreamName" operator="eq">
         <value>RCF_307577430</value>
         </attributeFilter>
         <attributeFilter name="JobName" operator="eq">
         <value>*</value>
         </attributeFilter>
         <attributeFilter name="Priority" operator="ge">
         <value>10</value>
         </attributeFilter>
         <attributeFilter name="Monitored" operator="eq">
         <value>true</value>
         </attributeFilter>
         <attributeFilter name="Status" operator="eq">
         <value>Running</value>
```

```
            <value>Successful</value>
            </attributeFilter>
            <attributeFilter name="Login" operator="eq">
            <value>TWS_user</value>
            </attributeFilter>
            </filteringPredicate>
         </eventCondition>
         <action actionProvider="TWSAction" actionType="sbs" responseType="onDetection">
            <description>Launch an existing TWS job stream</description>
            <scope>SBS NC125102#RJS_102739750</scope>
            <parameter name="JobStreamWorkstationName">
               <value>NC125102</value>
            </parameter>
            <parameter name="JobStreamName">
               <value>RJS_102739750</value>
            </parameter>
         </action>
      </eventRule>
   </eventRuleSet>
```

The rule in the following example submits job `RJR_30411` on workstation `NC122160` as soon as job stream `RJS_102739750` of workstation `NC125102` is submitted.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                         event-management/rules/EventRules.xsd">
   <eventRule name="TWS_PLAN_EVENTS_JOB_STREAM_SUBMITTED" ruleType="filter" isDraft="no">
      <description>Event: Job Stream Submitted; Action: Submit job</description>
      <eventCondition name="jsSubEvt1"
                      eventProvider="TWSObjectsMonitor"
                      eventType="JobStreamSubmit">
      <scope>WORKSTATIONVALUE # JOBSTREAMVALUE</scope>
         <filteringPredicate>
          <attributeFilter name="JobStreamWorkstation" operator="eq">
         <value>NC125102</value>
          </attributeFilter>
         <attributeFilter name="JobStreamName" operator="eq">
         <value>RJS_102739750</value>
         </attributeFilter>
         <attributeFilter name="Priority" operator="range">
         <value>15</value>
         <value>30</value>
         </attributeFilter>
         <attributeFilter name="LatestStart" operator="le">
         <value>2011-04-26</value>
         </attributeFilter>
         </filteringPredicate>
      </eventCondition>
      <action actionProvider="TWSAction" actionType="sbj" responseType="onDetection">
         <description>Launch an existing TWS job stream</description>
         <scope>SBJ NC122160#RJR_30411 INTO NC122160#JOBS</scope>
         <parameter name="JobUseUniqueAlias">
            <value>true</value>
         </parameter>
         <parameter name="JobDefinitionName">
            <value>RJR_30411</value>
         </parameter>
         <parameter name="JobDefinitionWorkstationName">
            <value>NC122160</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

FileMonitor events

`FileMonitor` events are:
- FileCreated

- FileDeleted
- ModificationCompleted
- LogMessageWritten

When you monitor files using the `FileCreated`, `FileDeleted`, and `LogMessageWritten` events, the memory consumed by the `ssmagent.bin` and `ssmagent.exe` processes increases linearly with the number of files monitored and with the number of events created. Therefore, keep in mind that the heavier use of wildcards you make within these event types, and the consequent higher number of files monitored, will result in a heavier memory consumption by the `ssmagent.bin` and `ssmagent.exe` processes.

FileMonitor events are not supported for dynamic agents, pools, dynamic pools and remote engine workstations.

Click here to view the parameters for each event type.

Note: PDF users, the above parameter tables are an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

Using the MatchExpression property of the LogMessageWritten event rule

The `LogMessageWritten` event plug-in uses the regular expression specified in the `MatchExpression` property to perform substring matches on entries in the log files being monitored. The value of `MatchExpression` must be a valid regular expression in accordance with the regular expression syntax rules of the Netcool/SSM agent that the event plug-in uses.

The following table describes the syntax of the regular expression tokens supported by Netcool/SSM. Note that to write a valid regular expression for the `MatchExpression` property, you must write the \ (backslash) escape character before each token used in the regular expression syntax (for example, \^ or \$). When the token already specifies a backslash character, you must write two backslash characters (for example, \\< or \\b).

Table 93. Regular expression syntax.

Token	Matches
.	Any character.
^	The start of a line (a zero-length string).
$	The end of a line; a new line or the end of the search buffer.
\<	The start of a word (where a word is a string of alphanumeric characters).
\>	The end of a word (the zero length string between an alphanumeric character and a non-alphanumeric character).
\b	Any word boundary (this is equivalent to (\<¦\>)).
\d	A digit character.
\D	Any non-digit character.
\w	A word character (alphanumeric or underscore).
\W	Any character that is not a word character (alphanumeric or underscore).

Table 93. Regular expression syntax. (continued)

Token	Matches
\s	A whitespace character.
\S	Any non-whitespace character.
\c	Special characters and escaping. The following characters are interpreted according to the C language conventions: \0, \a, \f, \n, \r, \t, \v. To specify a character in hexadecimal, use the \xNN syntax. For example, \x41 is the ASCII character A.
\	All characters apart from those described above may be escaped using the backslash prefix. For example, to specify a plain left-bracket use \[.
[]	Any one of the specified characters in a set. An explicit set of characters may be specified as in [aeiou] as well as character ranges, such as [0-9A-Fa-f], which match any hexadecimal digit. The dash (-) loses its special meaning when escaped, such as in [A\-Z] or when it is the first or last character in a set, such as in [-xyz0-9]. All of the above backslash-escaping rules may be used within []. For example, the expression [\x41-\x45] is equivalent to [A-D] in ASCII. To use a closing bracket in a set, either escape it using [\]] or use it as the first character in the set, such as []xyz]. POSIX-style character classes are also allowed inside a character set. The syntax for character classes is [:class:]. The supported character classes are: • [:alnum:] - alphanumeric characters. • [:alpha:] - alphabetic characters. • [:blank:] - space and TAB characters. • [:cntrl:] - control characters. • [:digit:] - numeric characters. • [:graph:] - characters that are both printable and visible. • [:lower:] - lowercase alphabetic characters. • [:print:] - printable characters (characters that are not control characters). • [:punct:] - punctuation characters (characters that are not letters, digits, control characters, or spaces). • [:space:] - space characters (such as space, TAB and form feed). • [:upper:] - uppercase alphabetic characters. • [:xdigit:] - characters that are hexadecimal digits. Brackets are permitted within the set's brackets. For example, [a-z0-9!] is equivalent to [[:lower:][:digit:]!] in the C locale.
[^]	Inverts the behavior of a character set [] as described above. For example, [^[:alpha:]] matches any character that is not alphabetical. The ^ caret symbol only has this special meaning when it is the first character in a bracket set.
{n}	Exactly n occurrences of the previous expression, where 0 <= n <= 255. For example, a{3} matches aaa.
{n,m}	Between n and m occurrences of the previous expression, where 0 <= n <= m <= 255. For example, a 32-bit hexadecimal number can be described as 0x[[:xdigit:]]{1,8}.
{n,}	At least n or more (up to infinity) occurrences of the previous expression.

Table 93. Regular expression syntax. (continued)

Token	Matches
*	Zero or more of the previous expression.
+	One or more of the previous expression.
?	Zero or one of the previous expression.
(exp)	Grouping; any series of expressions may be grouped in parentheses so as to apply a postfix or bar (¦) operator to a group of successive expressions. For example: • ab+ matches all of abbb • (ab)+ matches all of ababab
¦	Alternate expressions (logical OR). The vertical bar (¦) has the lowest precedence of all tokens in the regular expression language. This means that ab¦cd matches all of cd but does not match abd (in this case use a(b¦c)d).

Tip: When defining regular expressions to match multi-byte characters, enclose each multi-byte character in parentheses ().

Table 94 provides a set of regular expression examples, together with sample strings as well as the results of applying the regular expression to those strings.

There are two important cases in matching regular expressions with strings. A regular expression may match an entire string (a case known as a *string match*) or only a part of that string (a case known as a *sub-string match*). For example, the regular expression `\<int\>` will generate a sub-string match for the string `int x` but will not generate a string match. This distinction is important because some subagents do not support sub-string matching. Where applicable, the results listed in the examples differentiate between string and sub-string matches.

Table 94. Regular expression examples.

This expression...	Applied to this string...	Results in...
.	a	String match
	!	String match
	`abcdef`	Sub-string match on `a`
	empty string	No match
`M..COUNT`	`MINCOUNT`	String match
	`MXXCOUNTY`	Sub-string match on `MXXCOUNT`
	`NONCOUNT`	No match
.*	empty string	String match
	`Animal`	String match
.+	Any non-empty string	String match
	empty string	No match
^	empty string	String match
	`hello`	Sub-string match of length 0 at position 0 (position 0 = first character in string)

Table 94. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
$	empty string	String match
	hello	Sub-string match of length 0 at position 5 (position 0 = first character in string)
^$	empty string	String match
	hello	No match
\bee	tee	No match
	Paid fee	No match
	feel	No match
	eel	Sub-string match on ee
.*thing.*	The thing is in here	String match
	there is a thing	String match
	it isn't here	No match
	thinxxx	No match
a*	empty string	String match
	aaaaaaaaa	String match
	a	String match
	aardvark	Sub-string match on aa
	this string	Sub-string match
((ab)*c)*	empty string	String match
	ccccccccc	String match
	ccccabcccabc	String match
a+	empty string	No match
	aaaaaaaaa	String match
	a	String match
	aardvark	Sub-string match on aa
	this string	No match
(ab)+c)*	empty string	String match
	ababababcabc	String match
(ab){2}	abab	String match
	cdabababab	Sub-string match on abab
[0-9]{4,}	123	No match
	a1234	Sub-string match on 1234
a{0}	empty string	String match
	a	No match
	hello	Sub-string match of length 0 at position 0 (position 0 = first character in string)
[0-9]{1,8}	this is not a number	No match
	a=4238, b=4392876	Sub-string match on 4238
([aeiou][^aeiou])+	Hello	Sub-string match on el
	!!! Supacalafraglistic	Sub-string match on upacalaf

Table 94. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
[+-]?1	1	String match
	+1	String match
	-1	String match
	.1	Sub-string match on 1
	value+1	Sub-string match on +1
a¦b	a	String match
	b	String match
	c	No match
	Daniel	Sub-string match on a
abcd¦efgh	abcd	String match
	efgh	String match
	abcdfgh	Sub-string match on abcd
[0-9A-F]+	BAADF00D	String match
	C	String match
	baadF00D	Sub-string match on F00D
	c	No match
	G	No match
	g	No match
x = \d+	x = 1234	String match
	x = 0	String match
	x = 1234a	Sub-string match on x = 1234
	x = y	No match
	x^=^ where ^ represents a space character	No match
\D\d	a1	String match
	a11	Sub-string match on a1
	-9	String match
	a	No match
	8	No match
	aa	No match
	4t	No match
\s+	Hello_w0rld	No match
	Hello^^^world where ^ represents a space character	Sub-string match on ^^^ where ^ represents a space character
	Widget^ where ^ represents a space character	Sub-string match on ^ where ^ represents a space character
	^^^^^ where ^ represents a space character	String match

Table 94. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
\S+	`Hello_w0rld`	Sub-string match of length 11 on `Hello_w0rld`
	`Hello^^^world` where ^ represents a space character	Sub-string match on `Hello`
	`Widget^` where ^ represents a space character	Sub-string match on `Widget`
	^^^^^ where ^ represents a space character	No match
\w+	`D4n_v4n Vugt`	Sub-string match on `D4n_v4n`
	`^^^hello` where ^ represents a space character	Sub-string match on `hello`
	`blah`	String match
	`x#1`	No match
	`foo bar`	No match
\W	`Hello there`	Sub-string match of length 1 on separating space character
	`~`	String match
	`aa`	No match
	`a`	No match
	`-`	No match
	`^^^^444 == 5` where ^ represents a space character	Sub-string match of length 1 on first ^ where ^ represents a space character
\w+\s*=\s*\d+	`x = 123`	String match
	`count0=555`	String match
	`my_var=66`	String match
	`0101010=0`	String match
	`xyz = e`	No match
	`delta=`	No match
	`==8`	No match
[[:alnum:]]+	`1234`	String match
	`...D4N13L`	Sub-string match on `D4N13L`
[[:alpha:]]+	`Bubble`	String match
	`...DANI3L`	Sub-string match on `DANI`
	`69`	No match
[:blank:]]+	`alpha^^^^and beta` where ^ represents a space character	Sub-string match on ^^^^ where ^ represents a space character
	`Animal`	No match
	empty string	No match

Table 94. Regular expression examples. (continued)

This expression...	Applied to this string...	Results in...
[[:space:]]+	alpha^^^^and beta where ^ represents a space character	Sub-string match on ^^^^ where ^ represents a space character
	Animal	No match
	empty string	No match
[[:cntrl:]]+	...Hello W0rld!	No match
	empty string	No match
[[:graph:]]+	hello world	Sub-string match on hello
	^^^^^ where ^ represents a space character	No match
	^^^!? where ^ represents a space character	Sub-string match on !?
[:lower:]]+	Animal	Sub-string match on nimal
	ABC	No match
	0123	No match
	foobar	String match
	^^^0blaH! where ^ represents a space character	Sub-string match on bla
[_[:lower:]]+	foo_bar	String match
	this_thinG!!!	Sub-string match on _thin
[[:upper:]]+	YES	String match
	#define MAX 100	Sub-string match on MAX
	f00 b4r	No match
[[:print:]]+	hello world	String match
	^^^^^ where ^ represents a space character	String match
[[:punct:]]+	didn't	Sub-string match on '
	Animal	No match
[[:xdigit:]]+	43298742432392187ffe	String match
	x = bAAdF00d	Sub-string match on bAAdF00d
	4327afeffegokpoj	Sub-string match on 4327afeffe
c:\\temp	c:\temp	String match

Example

The rule in the following example sends an email to a list of recipients as soon as file /home/book.txt is created on workstation editor_wrkstn.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="FILE_MONITOR_FILE_CREATED" ruleType="filter" isDraft="no">
      <description>Event: File Created; Action: Send mail</description>
      <validity to="2012-04-22" />
      <eventCondition name="fileCrtEvt1" eventProvider="FileMonitor" eventType="FileCreated">
      <scope>/HOME/BOOK.TXT ON EDITOR_WRKSTN</scope>
         <filteringPredicate>
          <attributeFilter name="FileName" operator="eq">
         <value>/home/book.txt</value>
          </attributeFilter>
         <attributeFilter name="SampleInterval" operator="eq">
         <value>60</value>
         </attributeFilter>
         <attributeFilter name="Workstation" operator="eq">
         <value>editor_wrkstn</value>
         </attributeFilter>
         <attributeFilter name="Hostname" operator="eq">
         <value>ceditor</value>
         </attributeFilter>
                  </filteringPredicate>
      </eventCondition>
      <action actionProvider="MailSender" actionType="SendMail" responseType="onDetection">
         <description>Send an eMail</description>
         <scope>SAUL.FELLOW@US.IBM.COM, ISAAC.LINGER@US.IBM.COM : THE EXPECTED FILE
                HAS BEEN CREATED!</scope>
         <parameter name="Cc">
            <value>william.waulkner@us.ibm.com</value>
         </parameter>
         <parameter name="Bcc">
            <value>ernest.demingway@us.ibm.com</value>
         </parameter>
         <parameter name="Body">
            <value>The expected file was created!
                   The book is ready to be published.</value>
         </parameter>
         <parameter name="To">
            <value>saul.fellow@us.ibm.com, isaac.linger@us.ibm.com</value>
         </parameter>
         <parameter name="Subject">
            <value>The expected file was created!</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

TWSApplicationMonitor events

`TWSApplicationMonitor` events concern Tivoli Workload Scheduler processes, file system, and message box. They are:

- MessageQueuesFilling
- TivoliWorkloadSchedulerFileSystemFilling
- TivoliWorkloadSchedulerProcessNotRunning

Click here to see the parameters of each event type.

Note: PDF users, the above parameter tables are an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

Example

The rule in the following example logs warning message `LOGMSG01W` as soon as either `intercom` or `mailbox` message queue files on workstation `NC122160` reach 70 percent of their size.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                         event-management/rules/EventRules.xsd">
   <eventRule name="TWS_APPL_MONITOR_MESSAGE_QUEUES_FILLING" ruleType="filter" isDraft="no">
      <description>Event: Message queues filling; Action: Message logger</description>
      <timeZone>America/Los_Angeles</timeZone>
      <validity from="2011-04-25"/>
      <activeTime end="17:00:00"/>
      <eventCondition name="twsMesQueEvt1" eventProvider="TWSApplicationMonitor"
```

```
eventType="TWSMessageQueues">
     <scope>INTERCOM, MAILBOX FILLED UP 70% ON NC122160</scope>
        <filteringPredicate>
         <attributeFilter name="MailboxName" operator="eq">
        <value>intercom</value>
        <value>mailbox</value>
         </attributeFilter>
        <attributeFilter name="FillingPercentage" operator="ge">
        <value>70</value>
        </attributeFilter>
        <attributeFilter name="Workstation" operator="eq">
        <value>NC122160</value>
        </attributeFilter>
        <attributeFilter name="SampleInterval" operator="eq">
        <value>60</value>
        </attributeFilter>
                 </filteringPredicate>
     </eventCondition>
     <action actionProvider="MessageLogger" actionType="MSGLOG" responseType="onDetection">
        <description>Write a warning message log</description>
        <scope>OBJECT=LOGMSG01W MESSAGE=MAILBOX AND/OR INTERCOM QUEUE
               HAS REACHED 70% OF FILLING</scope>
        <parameter name="ObjectKey">
           <value>LOGMSG01W</value>
        </parameter>
        <parameter name="Message">
           <value>Mailbox and/or Intercom queue has reached 70% of filling</value>
        <parameter name="Severity">
           <value>Warning</value>
     </action>
  </eventRule>
</eventRuleSet>
```

Action providers and definitions

This section gives details on the action types of the following action providers:
- GenericAction
- MailSender
- MessageLogger
- TECEventForwarder
- TWSAction
- "TWSForZosAction" on page 588

GenericAction actions

This provider implements a single action named `RunCommand` that runs non-Tivoli Workload Scheduler commands. Commands are run on the same computer where the event processor runs.

Only *TWS_user* is authorized to run the command.

Important: When the command includes output redirection (through the use of one or two > signs), insert the command in an executable file, and set the file name as the argument of the `Command` property.

Click here to see the parameters of `RunCommand`.

Note: PDF users, the above parameter tables are an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

Example

The rule in the following example runs the **`ps -ef`** command to list all the currently running processes on a UNIX workstation when an invalid parameter is found on that workstation. Note that the rule is based on a custom event developed using the GenericEventPlugIn event provider. For more information on developing custom event types, see "Defining custom events" on page 123.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                          event-management/rules/EventRules.xsd">
   <eventRule name="CUSTOM_EVENT_GENERIC_EVENT" ruleType="filter" isDraft="yes">
      <description>Event: Generic Event; Action: Run Command</description>
      <activeTime start="08:30:00" end="17:30:00"/>
      <eventCondition name="genericEvt3" eventProvider="GenericEventPlugIn"
      eventType="Event1">
      <scope>INVALID PARAMETER ON WORKSTATIONVALUE</scope>
         <filteringPredicate>
           <attributeFilter name="Param1" operator="ne">
            <value>Invalid Parameter</value>
           </attributeFilter>
           <attributeFilter name="Workstation" operator="eq">
            <value>WorkstationValue</value>
           </attributeFilter>
        </filteringPredicate>
      </eventCondition>
      <action actionProvider="GenericActionPlugin" actionType="RunCommand"
      responseType="onDetection">
         <description>Run a command</description>
         <scope>PS -EF</scope>
         <parameter name="Command">
            <value>ps -ef</value>
         </parameter>
         <parameter name="WorkingDir">
            <value>/home</value>
         </parameter>
      </action>
   </eventRule>
</eventRuleSet>
```

MailSender actions

This provider implements a single action named SendMail that connects to an SMTP server to send an email. You can use optman to customize the following related attributes:
- Mail sender
- SMTP server
- SMTP port number
- Mail user name
- Mail user password
- SSL

Click here to see the parameters of SendMail.

Note: PDF users, the above parameter table is an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

MessageLogger actions

This provider implements a single action named MSGLOG that logs the occurrence of a situation in an internal auditing database. The number of entries within the auditing database is configurable. There is an automatic cleanup based on a FIFO policy.

Click here to see the parameters of MSGLOG.

Note: PDF users, the above parameter table is an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

TECEventForwarder actions

This provider implements a single action named `TECFWD` that forwards the event to an external TEC (Tivoli Enterprise Console) server (or any other application capable of listening to events in TEC format). The provider uses a default TEC server whose host name and port can be defined with `optman`. The TEC used as recipient can be overridden by action settings.

Click here to see the parameters of `TECFWD`.

Note: PDF users, the above parameter table is an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

TWSAction actions

`TWSAction` actions are:
- SubmitJobStream
- SubmitJob
- SubmitAdHocJob
- ReplyPrompt

Click here to see the parameters of each action type.

Note: PDF users, the above parameter tables are an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

Using the `SchedTimeResolutionCriteria` property of the `SubmitJob` action

You use this property to match the job in question with a specific instance of the job stream that contains it (defined with the `JobStreamName` property) based on the job stream scheduled time. The possible values that you can set are:

Previous
The job is submitted with the closest previous job stream instance in plan.

Next The job is submitted with the closest next job stream instance in plan.

Any The job is submitted with any of the closest previous or closest next job stream instance in plan.

TWSForZosAction

This provider implements a single action named `AddJobStream` that adds an application occurrence (job stream) to the current plan on Tivoli Workload Scheduler for z/OS. This provider is for use in Tivoli Workload Scheduler end-to-end scheduling configurations.

The application description of the occurrence to be added must exist in the AD database of Tivoli Workload Scheduler for z/OS.

Click here to see the parameters of `AddJobStream`.

Note: PDF users, the above parameter table is an html file referenced by the PDF. It is not saved locally with the PDF from the infocenter. You must first view it on the infocenter before saving or printing.

Example

In this example, a pharmaceutical company uses rule ZOSRULE031 to produce a distribution schedule of the merchandise under the control of department DISTR07. As soon as the list of ordered merchandise that is up for delivery in the upcoming month is ready and placed in file MONTHLYORDERS.TXT on agent RU192298 in a branch office, the centralized system adds application (job stream) ADFIRST to the current plan. ADFIRST contains the operations (jobs) that produce an optimized delivery schedule for the next month.

```
<?xml version="1.0"?>
<eventRuleSet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://www.ibm.com/xmlns/prod/tws/1.0/event-management/rules"
      xsi:schemaLocation="http://www.ibm.com/xmlns/prod/tws/1.0/
                        event-management/rules/EventRules.xsd">
   <eventRule name="ZOSRULE031" ruleType="filter" isDraft="no">
      <eventCondition name="fileCrtEvt19" eventProvider="FileMonitor"
      eventType="FileCreated">
      <scope>/PRODORDER/MONTHLYORDERS.TXT ON RU192298</scope>
         <filteringPredicate>
           <attributeFilter name="Param1" operator="ne">
            <value>/prodorder/monthlyorders.txt</value>
           </attributeFilter>
           <attributeFilter name="SampleInterval" operator="eq">
            <value>60</value>
           </attributeFilter>
           <attributeFilter name="Workstation" operator="eq">
           <value>RU192298</value>
            </attributeFilter>
        </filteringPredicate>
      </eventCondition>
      <action actionProvider="TWSForZosAction" actionType="AddJobStream"
      responseType="onDetection">
        <scope>
          ADD JOBSTREAM ADFIRST[DEADLINE OFFSET: 0001] WITH OWNER DISTR07 IN PLAN
        </scope>
         <parameter name="HoldAll">
            <value>false</value>
         </parameter>
         <parameter name="Priority">
            <value>5</value>
         </parameter>
         <parameter name="JobStreamDeadlineOffset">
            <value>0001</value>
         </parameter>
         <parameter name="JobStreamName">
            <value>ADFIRST</value>
         </parameter>
         <parameter name="OwnerDescription">
            <value>Owner description</value>
         </parameter>
         <parameter name="Owner">
            <value>distr07</value>
         </parameter>
         <parameter name="DependenciesResolution">
            <value>All</value>
         </parameter>
         <parameter name="AuthorityGroup">
            <value>AuthGrpBase</value>
         </parameter>
         <parameter name="Parm_1">
            <value>var1=value1</value>
         </parameter>
         <parameter name="Parm_2">
```

```
          <value>var2=value2</value>
       </parameter>
       <parameter name="JCLVariableTable">
          <value>VarTableZos01</value>
       </parameter>
       <parameter name="JobStreamDescription">
          <value>This job stream contains jobs that process orders for
                      owner DISTR07.</value>
       </parameter>
       <parameter name="Group">
          <value>GroupBase</value>
       </parameter>
    </action>
 </eventRule>
</eventRuleSet>
```

Appendix B. Job Submission Description Language schema reference

This reference section specifies the semantics and structure of the Job Submission Description Language (JSDL) that apply specifically for use with dynamic workload broker. The JSDL schema is used to describe the job requirements for submission to resources. dynamic workload broker analyzes the IT environment and assigns the best available resource to run the job, based on the requirements you specify.

Introduction

The Job Submission Description Language (JSDL) is a language for describing the job requirements for submission to resources. The JSDL language contains a vocabulary and normative XML schema that facilitate the expression of those requirements as a set of XML elements.

JSDL files adhere to the XML syntax and semantics as defined in the JSDL schema.

Job Submission Description Language document structure

A JSDL file is described using the XML syntax and adheres to the XML syntax and semantics. The XML syntax is an industry standard and is not explained in this manual. The JSDL file also adheres to specific JSDL syntax rules, as explained in "Job Submission Description Language element types" on page 594 and in "JSDL elements" on page 597.

The JSDL file consists of elements (either complex or simple) and types. Complex elements contain other elements while simple elements do not contain any other elements. A type specification performs a syntax check on the value specified for the element it refers to. For example, the **physicalMemory** element adheres to the jsdl:NumericRangeType type. The jsdl:NumericRangeType type specifies that you can assign to this element either a specific numeric value or a numeric range value. No other value types are supported for the **physicalMemory** element.

The JSDL file is arranged in a hierarchical structure where the **jobDefinition** element is the root element. The **jobDefinition** element contains all the elements that describe the job and their attributes.

The pseudo schema definition looks like this:

```
< jobDefinition  >
  <annotation ... />?
  <category>... />*
  <variables ... />?
  <application ... />
  <resources ... />?
  <relatedResources ... />*
  <optimization ... >?
  <scheduling ...>?
</jobDefinition>
```

Table 95 on page 592 provides a table view of the JSDL file indicating the hierarchical relationships between the elements contained in the **jobDefinition** element.

Table 95. Hierarchical structure of the JSDL file

First level	Second level	Third level	Fourth level
annotation			
category			
variables	stringVariable		
	uintVariable		
	doubleVariable		
application	script		
	arguments	value	
	environment	variable name	
	credential	username	
		groupname	
		password	
	j2ee	invoker	type
		jms	connFactory
			destination
			message
		ejb	jndiHome
		credential	userName
			password
			JAASalias

Table 95. Hierarchical structure of the JSDL file (continued)

First level	Second level	Third level	Fourth level
resources	candidateHosts	hostName	
	candidateCPUs	cpu	speed
	physicalMemory		
	virtualMemory		
	candidateOperating Systems	operatingsystems	
	fileSystem		
	logicalResource		
	group		
	properties	and	and
			or
			requirement
		or	and
			or
			requirement
		requirement	and
			or
			requirement
	allocation		
	relationship		
	candidateResources (reserved for internal use)	endpointReference (reserved for internal use)	
relatedResources	logicalResource		
	group		
	properties	and	and
			or
			requirement
		or	and
			or
			requirement
		requirement	and
			or
			requirement
	allocation		
	relationship		
	candidateResources (reserved for internal use)	endpointReference (reserved for internal use)	
optimization	objective		
	ewlm		

Table 95. Hierarchical structure of the JSDL file (continued)

First level	Second level	Third level	Fourth level
scheduling	maximumResource WaitingTime		
	estimatedDuration		
	priority		
	recoveryActions	action	parameters
			credential
			tpmaddress
			workflow

The JSDL syntax uses the BNF-style conventions for elements and attributes:

?	Indicates that the element or attribute is optional and can be specified once.
*****	Indicates that the element or attribute is optional and can be specified zero or more times.
+	Indicates that the element or attribute is required and can be specified one or more times.
[...]	Indicate that the elements or attributes contained within the brackets form a group.
|	Indicates that two or more elements or attributes are mutually exclusive.

Job Submission Description Language element types

The JSDL specification uses a number of standard XML Schema types. It also uses a number of types specific to the description of job requirements.

Both types perform a syntax check on the value that can be assigned to each element in the JSDL file. For example, the **physicalMemory** element adheres to the jsdl:NumericRangeType type. The jsdl:NumericRangeType type specifies that you can assign to this element either a specific numeric value or a numeric range value. No other value types are supported for the **physicalMemory** element.

Normative XML schema types

The JSDL specification adopts the normative XML schema (xsd) types listed below. The XML syntax is an industry standard and is not explained in this manual.

- xsd:any
- xsd:anyURI
- xsd:boolean
- xsd:double
- xsd:DoubleVariableType
- xsd:duration
- xsd:IDREF
- xsd:NCName
- xsd:PriorityType
- xsd:QName

- xsd:string
- xsd:unsignedInt
- xsd:UnsignedIntVariableType

JSDL types

The following types are specific to the JSDL syntax:

StringVariableExpressionType
A string variable expression type is a simple type in which you can specify a variable expression that might contain one ore more variable references, such as ${var}, any character, and any string. The following is the syntax schema for this type:

```
<...>
<xsd:simpleType name="StringVariableExpressionType">
  <xsd:union>
   <xsd:simpleType>
    <xsd:restriction base='xsd:string' />
   </xsd:simpleType>
   <xsd:simpleType>
    <xsd:restriction base='xsd:string'>
     <xsd:pattern
      value=".*\t*\r*\n*((\$\{[a-zA-Z_]+
          [0-9a-zA-Z_\.\-]*\})+[^\{]*[.\n]*)+" />
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:union>
 </xsd:simpleType>
</...>
```

DoubleVariableExpressionType
A double variable expression type is a simple type in which you can specify a variable expression that might contain one variable reference, such as ${var}, or a double value. The following is the syntax schema for this type:

```
<...>
 <xsd:simpleType name="DoubleVariableExpressionType">
  <xsd:union>
   <xsd:simpleType>
    <xsd:restriction base='xsd:double' />
   </xsd:simpleType>
   <xsd:simpleType>
    <xsd:restriction base='xsd:string'>
     <xsd:pattern value="[\n\r\t ]*($\{[a-zA-Z_]+
              [0-9a-zA-Z_\.\-]*\})[\n\r\t ]*" />
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:union>
 </xsd:simpleType>

</...>
```

UnsignedIntVariableExpressionType
An unsigned variable expression type is a simple type in which you can specify a variable expression that might contain one variable reference, such as ${var}, or an unsigned integer value. The following is the syntax schema for this type:

```
<...>
<xsd:simpleType name="UnsignedIntVariableExpressionType">
  <xsd:union>
   <xsd:simpleType>
    <xsd:restriction base='xsd:unsignedInt' />
   </xsd:simpleType>
   <xsd:simpleType>
    <xsd:restriction base='xsd:string'>
     <xsd:pattern value="[\n\r\t ]*($\{[a-zA-Z_]+
               [0-9a-zA-Z_\.\-]*\})[\n\r\t ]*" />
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:union>
 </xsd:simpleType>

</...>
```

NotEmptyStringVariableExpressionType

A string variable expression type is a simple type that allows the specification of a variable expression that might contain one or more variable references such as ${var}, optionally in association with any character or with a simple string. This variable expression cannot be empty. The following is the syntax schema for this type:

```
<xsd:simpleType name="NotEmptyStringVariableExpressionType">
  <xsd:union>
   <xsd:simpleType>
    <xsd:restriction base='xsd:string'>
     <xsd:minLength value="1"/>
    </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType>
    <xsd:restriction base='xsd:string'>
     <xsd:pattern
      value=".*\t*\r*\n*((\$\{[a-zA-Z_]+
                     [0-9a-zA-Z_\.\-]*\})+[^\{]*[.\n]*)+" />
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:union>
 </xsd:simpleType>
```

NumericRangeOnlyType

A numeric range value is a complex type that allows the definition of intervals and ranges higher than, smaller than, or contained within the specified value. All numbers given are double variable expressions. The following is the syntax schema for this type:

```
<...>
 <minimum>jsdl:DoubleVariableExpressionType</minimum> ?
 <maximum> jsdl:DoubleVariableExpressionType</maximum> ?
</...>
```

NumericRangeType

A numeric range value is a complex type that allows the definition of exact values or ranges. All numbers given are double variable expressions. The following is the syntax schema for this type:

```
<...>
 <exact>jsdl:DoubleVariableExpressionType</exact> |
 <range>jsdl:NumericRangeOnlyType</range>
</...>
```

StringRangeOnlyType

A string range value is a complex type that allows the definition of intervals and ranges higher than, smaller than, or contained within the specified value. All numbers and strings given are string variable expressions. The following is the syntax schema for this type:

```
<...>
 <minimum>jsdl:StringVariableExpressionType</minimum> ?
 <maximum>jsdl:StringVariableExpressionType</maximum> ?
</...>
```

StringRangeType

A string range value is a complex type that allows the definition of exact values as string variable expressions or ranges that can be applied to integer or string types. The following is the syntax schema for this type:

```
<...>
 <exact>jsdl:StringVariableExpressionType</exact> |
 <range>jsdl:StringRangeOnlyType</range>
</...>
```

JSDL elements

The JSDL core element set contains the semantics for elements that are defined by JSDL.

The JSDL file consists of elements (either complex or simple) and types. Complex elements contain other elements while simple elements do not contain any other elements. A type specification performs a syntax check on the value specified for the element it refers to.

The following is a list of the elements contained in the JSDL syntax:

jobDefinition element

Definition

This element describes the job and its requirements. It is the root element of the JSDL document. This attribute is required.

Type

The type of this element is jsdl:JobDefinitionType. It can contain the following elements:

- annotation
- category
- variables
- application
- resources
- relatedResources
- optimization
- scheduling

Attributes

name
The name of the job specified by the user. The type of this attribute is **xsd:NCName**. The name must start with an alphabetical character, and can contain underscore symbols (_), minus symbols (-), and periods (.). Spaces, special characters, accented characters, and numbers are not supported. This attribute is required. The name you define for this field uniquely identifies the job definition when it is saved in the Job Repository database. After saving the job definition in the database, you can submit the job definition using the Dynamic Workload Console or the command line.

description
A string specifying a short description of the job definition. The type of this attribute is **xsd:string**. This attribute is optional.

targetNamespace
A URI specifying the target namespace of the job definition. The type of this attribute is **xsd:anyURI**. This attribute is required.

Pseudo Schema

```
<jobDefinition
 name="xsd:NCName"
 description="xsd:string"?
 xsd:anyAttribute##other>
 <annotation ... />?
 <category>.../>*
 <variables ... />?
 <application ... />
 <resources ... />?
 <relatedResources .../>*
 <optimization ...>?
 <scheduling ...>?
 <xsd:any##other/>*
</jobDefinition>
```

annotation element

Definition
This element provides descriptive, human-readable information about the job definition. This element is optional and can be specified once.

Type The type of this element is xsd:string.

Attributes
No attributes are defined.

Pseudo Schema

```
<annotation
 xsd:anyAttribute##other>
 xsd:string
 <xsd:any##other/>*
</application>
```

category element

Definition
This element describes job category that help you categorize the job. One job may have multiple categories, for example: Education_DB, Financial_Dept, Asset_Management. The value can be any string value. This element is optional and can be specified zero or more times.

Type The type of this element is xsd:string.

Attributes
No attributes are defined.

Pseudo Schema

```
<category>
 xsd:string
</category>
```

variables element

Definition
This element describes the list of variables that are defined in the JSDL file. The following three variable types are supported:

- String
- Double
- Integer

The variable value can be referred in other parts of the JSDL document specifying: ${variable name} A referred variable can be one defined in the JSDL file with the **variable** element or it can be defined while submitting the job. The substitution can be performed by the dynamic workload broker server in different phases of the job processing. In every phase, the dynamic workload broker server tries to match all the variable references still not substituted with defined variables. This element is optional and can be specified once.

Type The type of this element is jsdl:VariablesType. It can contain the following elements:

- stringVariable
- uintVariable
- doubleVariable

Attributes
No attributes are defined.

Pseudo Schema
```
<variables
 xsd:anyAttribute##other>
 <stringVariable ...>*
 <uintVariable ...>*
 <doubleVariable ...>*
 <xsd:any##other/>*
</variables>?
```

stringVariable element

Definition
This element describes a variable specifying the variable name and the assigned default string value. This element is optional and can be specified zero or more times.

Type The type of this element is jsdl:StringVariableType.

Attributes
The following attributes are defined:

name This attribute specifies the name of the variable. The type of this attribute is xsd:NCName. This attribute is required.

description
This attribute specifies the description of the variable. The type of this attribute is xsd:string. This attribute is optional.

Pseudo Schema
```
<stringVariable
 name="xsd:NCName"
 description="xsd:string"?
 xsd:anyAttribute##other>
 xsd:string
 <xsd:any##other/>*
</stringVariable>
```

doubleVariable element

Definition
This element describes a variable specifying the variable name and the assigned default double value. This element is optional and can be specified zero or more times.

Type The type of this element is xsd:DoubleVariableType.

Attributes
The following attributes are defined:

name This attribute specifies the name of the variable. The type of this attribute is xsd:NCName. This attribute is required.

description
This attribute specifies the description of the variable. The type of this attribute is xsd:string. This attribute is optional.

Pseudo Schema
```
<doubleVariable
 name="xsd:NCName"
 description="xsd:string"?
 xsd:anyAttribute##other>
 xsd:double
 <xsd:any##other/>*
</doubleVariable>
```

uintVariable element

Definition
This element describes a variable specifying the variable name and the assigned default unsigned integer value. This element is optional and can be specified zero or more times.

Type The type of this element is xsd:UnsignedIntVariableType.

Attributes
The following attributes are defined:

name This attribute specifies the name of the variable. The type of this attribute is xsd:NCName. This attribute is required.

description
This attribute specifies the name of the variable. The type of this attribute is xsd:string. This attribute is optional.

Pseudo Schema
```
<uintVariable
 name="xsd:NCName"
 description="xsd:string"?
 xsd:anyAttribute##other>
 xsd:unsignedInt
 <xsd:any##other/>*
</uintVariable>
```

application element

Definition
This element describes the application to be run the related parameters. This element is required and can be specified once.

Type The type of this element is jsdl:ApplicationType.

Attributes
The following attributes are defined:

name Specifies the type of the application. Supported values are as follows:

Executable
A file which is used to perform various functions or operations on a computer.

J2EE An application based on Java 2 Platform Enterprise Edition (J2EE).

This attribute is mandatory and can be specified once.

description
Specifies the name of the application. The type of this attribute is xsd:string. This attribute is optional.

version
Specifies the version of the application. The type of this attribute is xsd:string. This attribute is optional.

Pseudo Schema

```
<application
 name="xsd:NCName"
 description="xsd:string"?
 version="xsd:string"?
 xsd:anyAttribute##other>
 <xsd:any##other/>*
</application>
```

resources element

Definition
This element contains the resource requirements of the job that must be matched on target computer system in order for a job to be assigned to that system. The contained resource requirement elements are combined in an AND relationship. This means that each requirement adds to the other ones to refine the matching resource requirement and all requirements must be met for the allocation to succeed. This element is optional and can be specified once. If this element is not present, the dynamic workload broker server can choose any set of resources to run the job.

Type
The type of this element is jsdl:ResourceType. The supported types for this element are listed in Table 96 on page 631. It can contain the following elements:

- candidateHosts
- candidateCPUs
- physicalMemory
- virtualMemory
- candidateOperatingSystems
- fileSystem
- logicalResource
- group
- properties
- allocation
- relationship
- candidateResources

Attributes
No attributes are defined.

Pseudo Schema

```
<resources
 xsd:anyAttribute##other>
 <candidateHosts .../>?
 <candidateCPUs .../>?
 <physicalMemory .../>?
 <virtualMemory .../>?
 <candidateOperatingSystems .../>?
 <fileSystem .../>*
 <logicalResource ...>*
 <group ...>*
 <properties ...>
 <allocation ...>*
 <relationship ...>*
 <candidateResources ...>?
 <xsd:any##other>*
</resources>
```

relatedResources element

Definition
This element is a unique identifier for the resource requirement that must be unique in the document. The requirements defined in this element apply to both logical resources and computer systems. It can be referred to by **relationship** elements. The contained resource requirement elements are combined in an AND relationship. This means that each requirement adds to the other ones to refine the matching resource requirement and all requirements must be met for the allocation to succeed. This element is optional and can be specified zero or more times.

Type The type of this element is jsdl:RelatedResourceType. It can contain the following elements:
- logicalResource
- group
- properties
- allocation
- relationship
- candidateResources

Attributes
The following attributes are defined:

id Specifies the internal ID of the resource you want to associate to the resources element. This ID is used only for internal reference within the JSDL file. The type for this attribute is xsd:ID. This attribute is required.

type Specifies the type of the required resource. Supported types are ComputerSystem and Logical Resource. If this attribute is not present, the resource type ComputerSystem is assumed. A resource type is identified by a unique type name and describes the properties that each resource instance provides. For more information on the available resource properties, see Table 96 on page 631. The type of this attribute is xsd:NCName. This attribute is optional.

Pseudo Schema
```
<relatedResources
 id="xsd:ID"
 type="xsd:NCName"?
 xsd:anyAttribute##other>
 <logicalResource ...>*
 <group ...>*
 <properties ...>
 <allocation ...>*
 <relationship ...>*
 <candidateResources ...>?
 <xsd:any##other>*
</relatedResources>
```

See Also

"resources element" on page 601

candidateHosts element

Definition
This element specifies the set of named hosts which must be selected for running the job. dynamic workload broker assigns the job to one of the hosts in this list. The hosts specified are in OR, that is at least one of them must be matched by the Operating System resource contained in the target resource. If none of the hosts you specify is available when the job is submitted, the job waits for one of them to become available. If no host becomes available before the timeout expires, the job fails. This attribute is optional.

Type The type of this element is jsdl:CandidateHostsRequirementType. It can contain the following element:

- hostName

Attributes
No attributes are defined.

Pseudo Schema

```
<candidateHosts
 xsd:anyAttribute##other>
 <hostName>jsdl:StringVariableExpressionType<hostName/>+
 <xsd:any##other>*
</candidateHosts>
```

orderedCandidatedWorkstations element

Definition
This element specifies the ordered list of workstations that are candidate for being selected based on the information specified in the requirements field of the dynamic pool definition. The first workstation that matches the requirements is selected for processing.

Type The type of this element is jsdl:OrderedCandidatedWorkstationsType. It can contain the following element:

- workstation

Attributes
No attributes are defined.

Pseudo Schema

```
<orderedCandidatedWorkstations
 xsd:anyAttribute##other>
 <workstation>jsdl:StringVariableExpressionType<workstation/>+
 <xsd:any##other>*
</orderedCandidatedWorkstations>
```

hostName element

Definition
This element specifies a string variable expression containing the name of a single host which may be selected for running the job. If none of hosts you specify is available when the job is submitted, the job waits for one of them to become available. If no host becomes available before the timeout expires, the job fails. To specify the host name, you can use a variable expression that can contain one ore more variable references, such as

${var}, any character, and any string. Wildcards are supported. This attribute is required and can be specified one or more times.

Type The type of this attribute is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema
```
 <hostName>jsdl:StringVariableExpressionType<hostName/>


 <hostName>lab145674.example.com<hostName/>

<hostName>${my_preferred_host}<hostName/>
```

candidateCPUs element

Definition
This element specifies the set of CPU characteristics that must be satisfied by hosts which may be selected for running the job. The characteristics combinations specified are in OR, that is at least one of them must be matched by the target resource. If none of the CPU characteristics you specify is available when the job is submitted, the job waits for one of them to become available. If no CPU becomes available before the timeout expires, the job fails. This element is optional and can be specified zero or more times.

Type The type of this element is jsdl:CandidateCPUsRequirementType. It can contain the following element:
- cpu

Attributes
No attributes are defined.

Pseudo Schema
```
<candidateCPUs
 xsd:anyAttribute##other>
 <cpu ...>*
 <xsd:any##other/>*
</candidateCPUs >?
```

cpu element

Definition
This element specifies the CPU characteristics that must be satisfied by hosts which may be selected for running the job. The characteristics combinations specified are in OR, that is at least one of them must be matched by the target resource. At least one of the CPUs with the specified characteristics must be available for the job to run. If none of the CPUs you specify is available when the job is submitted, the job waits for one of them to become available. If no CPU becomes available before the timeout expires, the job fails. This element is required and can be specified one or more times.

Type The type of this element is CPURequirementType.

Attributes
The following attributes are defined:

architecture
Specifies the CPU architecture required for running the job. This attribute is optional. Supported values are as follows:

- parisc
- powerpc
- powerpc_64
- s390
- s390x
- sparc
- x86
- x86_64

quantity
Specify the number of processors that must be available on the computer. Setting the Quantity to 0 indicates that the requirement is met by computers with any number of processors. The type of this attribute is xsd:unsignedInt. This attribute is optional.

Pseudo Schema

```
<cpu>
 architecture="xsd:string"?
 speed="jsdl:NumericRangeType"?
 quantity="xsd:unsignedInt"?
 xsd:anyAttribute##other>
 <xsd:any##other>*
</cpu>
```

speed element

Definition
This element specifies the CPU speed range in MHz required for running the job The unit of measure is megahertz. This element is optional and can be specified once.

Type The type of this element is jsdl:NumericRangeType.

Attributes
No attributes are defined.

Pseudo Schema

```
<speed>
 xsd:anyAttribute##other>
 <jsdl:NumericRangeType>*
<speed>
```

physicalMemory element

Definition
This element specifies a range or exact value indicating the amount of free physical memory required for the job. The amount is expressed in bytes. This attribute is optional.

Type The type of this element is jsdl:NumericRangeType.

Attributes
No attributes are defined.

Pseudo Schema

```
<physicalMemory
 xsd:anyAttribute##other>
 jsdl:NumericRangeType
 <xsd:any##other>*
</physicalMemory>
```

virtualMemory element

Definition
This element specifies a range or exact value indicating the amount of free virtual memory required for the job. The amount is expressed in bytes. This attribute is optional.

Type The type of this element is jsdl:NumericRangeType.

Attributes
No attributes are defined.

Pseudo Schema
```
<virtualMemory
 xsd:anyAttribute##other>
 jsdl:NumericRangeType
 <xsd:any##other>*
</virtualMemory>
```

candidateOperatingSystems element

Definition
This element specifies the set of operating system characteristics that must be satisfied by hosts which may be selected for running the job. The characteristics combinations are matched in OR, that is at least one of them must be matched by the Operating System resource contained in the target resource. At least one of the operating systems listed must be available for the job to run. If none of the operating systems you specify is available when the job is submitted, the job waits for one of them to become available. If no operating system becomes available before the timeout expires, the job fails. This attribute is optional.

Type The type of this element is jsdl:OperatingSystemsRequirementType. It can contain the following element:

- operatingSystem

Attributes
No attributes are defined.

Pseudo Schema
```
<candidateOperatingSystems
 <xsd:anyAttribute##other>
 <operatingSystem...>*
 <xsd:any##other/>*
</ candidatOperatingSystems>?
```

operatingSystem element

Definition
This element specifies the operating system characteristics that are required for running the job. This attribute is required and can be specified one or more times.

Type The type for this element is OperatingSystemRequirementType

Attributes
The following attributes are defined:

type This attribute defines the name of operating system required for running the job. The type of this attribute is xsd:string. This attribute is required. Supported values are as follows:

- AIX
- Linux

- Windows 2000
- Windows 2003
- Windows XP
- Windows Vista
- HPUX
- Solaris

version
Specify the operating system version. You can specify the exact operating system version or sub version, for example 5.2 or 5.2.3.30, or you can specify a part of the version, for example 5.2.3. In this case, the requirement applies to all operating systems having the 5.2.3 version and any sub versions, such as fix packs and maintenance levels. The type of this attribute is xsd:string. This attribute is optional.

Pseudo Schema

```
<operatingSystem
 type="xsd:string"
 version="xsd:string"?
 xsd:anyAttribute##other>
 <xsd:any##other>*
</operatingSystem>
```

fileSystem element

Definition
This element describes the set of file system characteristics which may be selected for running the job. Each set of characteristics specifies the location where the file system is available, the required amount of disk space and the type of the file system. The file system may be local to the resource, for example on a local disk, or may be remote, for example on an NFS mount. The requirement is satisfied if for a given target all listed file system characteristics are satisfied. The characteristics combinations are matched in AND, that is all must be matched by the File System resources contained in the target resource. All of the file systems listed must be present for the job to run. If any of the file systems you specify is not available when the job is submitted, the job waits for it to become available. If it does not become available before the timeout expires, the job fails. This element is optional and can be specified zero or more times.

Type The type for this element is jsdl:FileSystemRequirementType. It contains the **diskSpace** element.

Attributes
The following attributes are defined:

type Is a token specifying the type of file system of the containing fileSystem element. The type of this attribute is jsdl:FileSystemTypeEnumeration. This attribute is optional. Supported values are as follows:

Unkonwn
The file system is not specified.

No Root Directory
Is a local file system that is not the root directory.

Removable Disk
Is a file system mounted on a removable hard disk.

Local Disk
Is a file system mounted on a local disk.

Remote Drive
Is a file system mounted on a remote drive.

CD-ROM
Is a file system mounted on a CD ROM drive.

RAM Disk
Is a file system mounted on a RAM disk.

mountPoint
Is a string variable expression specifying the local mapping where the file system is available for the job. The type of this attribute is jsdl:StringVariableExpressionType. To specify the mount point, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. Wildcards are supported. This attribute is optional.

Pseudo Schema

```
<fileSystem
 type="jsdl:FileSystemTypeEnumeration"?
 mountPoint="jsdl:StringVariableExpressionType"?
 xsd:anyAttribute##other>
 <diskSpace>jsdl:NumericRangeType</diskSpace>?
 <xsd:any##other/>*
</fileSystem>
```

diskSpace element

Definition
This element specifies the amount of disk space required on the containing file system element for running the job. The amount of disk space is given in kilobytes. This element is optional and can be specified once.

Type The type of this element is jsdl:NumericRangeType.

Attributes
No attributes are defined.

Pseudo Schema

```
<fileSystem
 type="jsdl:FileSystemTypeEnumeration"?
 mountPoint="jsdl:StringVariableExpressionType"?
 xsd:anyAttribute##other>
 <diskSpace>jsdl:NumericRangeType</diskSpace>?
 <xsd:any##other/>*
</fileSystem>
```

logicalResource element

Definition
This element specifies one or more logical resources are required for running the job. The characteristics combinations are matched in AND, that is all must be matched by the Logical Resources associated with the target resource. All of the logical resources listed must be available for the job to run. If one of the logical resources you specify is unavailable when the job is submitted, the job waits for it to become available. If it does not become available before the timeout expires, the job fails. This element is optional and can be specified zero or more times.

Type The type of this element is LogicalResourceRequirementType.

Attributes
The following attributes are defined:

name Is a string variable expression specifying the name of the requested logical resource. To specify the logical resource name, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. 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. This attribute is optional.

subType
Is a string variable expression specifying the type of the requested logical resource. To specify the logical resource type, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. This attribute is optional.

quantity
The integer value specifying the required quantity of the logical resource. The specified quantity is allocated exclusively to the job while it runs. To specify the amount of the resource, you can use a variable expression that can contain one variable reference, such as ${var}, or an unsigned integer value. This attribute is optional.

Pseudo Schema
```
<logicalResource
 name="jsdl:StringVariableExpressionType"?
 subType="jsdl:StringVariableExpressionType"?
 quantity="jsdl:UnsignedIntVariableExpressionType"?
 xsd:anyAttribute##other>
 <xsd:any##other/>*
</logicalResource>
```

group element

Definition
This element specifies that the resources required for running the job have to belong to the specified group of resources. The groups are matched in AND, that is the target resource must be in all the groups specified. This element is optional and can be specified zero or more times.

Type The type of this element is jsdl:GroupRequirementType.

Attributes
The following attribute is defined:

name Specifies the name of the group as a string variable expression. To specify the resource group, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. This attribute is required and can be specified once.

Pseudo Schema
```
<group
 name="jsdl:StringVariableExpressionType"
 xsd:anyAttribute##other>
 <xsd:any##other/>*
</group>
```

properties element

Definition
This element specifies the properties of the resource required for running the job. The requirement is expressed as a set of conditions on resource properties combined with AND/OR operators. This is based on the resource model that describes the resources available in the environment as instances of resource types. A resource type is identified by a unique type name and describes the properties that each resource instance provides. Use this element to specify advanced requirements. For more information on available resource types and properties, see Table 96 on page 631. This element is optional and can be specified once.

Type The type of this element is jsdl:RequirementCompositorType. It can contain the following elements:
- and
- or
- requirement

Attributes
No attributes are defined.

Pseudo Schema
```
<properties
 xsd:anyAttribute##other>
 <and .../>?
 <or .../>?
 <requirement .../>?
 <xsd:any##other/>*
</properties>
```

and element

Definition
This element specifies an AND condition on the containing requirement specifications. This element is optional and can be specified once.

Type The type of this element is jsdl:RequirementCompositorType. It can contain the following elements:
- and
- or
- requirement

Attributes
No attributes are defined.

Pseudo Schema
```
<and
 xsd:anyAttribute##other>
 <and .../>?
 <or .../>?
 <requirement .../>?
 <xsd:any##other/>*
</and>
```

or element

Definition
This element specifies an OR condition on the containing requirement specifications. This element is optional and can be specified once.

Type The type of this element is jsdl:RequirementCompositorType. It can contain the following elements:

- and
- or
- requirement

Attributes
No attributes are defined.

Pseudo Schema
```
<or
 xsd:anyAttribute##other>
 <and .../>?
 <or .../>?
 <requirement .../>?
 <xsd:any##other/>*
</or>
```

requirement element

Definition
This element is a range value specifying a requirement on the capabilities of a resource for running the job. This element is optional and can be specified once.

Type The type of this element is jsdl:RequirementType.

Attributes
The following attribute is defined:

propertyName
Is a string specifying the resource property that the requirement applies to. The available properties vary depending on resources you selected in the resources element. For more information on resource types and properties, see Table 96 on page 631. The type of this attribute is xsd:NCName. This attribute is required.

Pseudo Schema
```
<requirement
 propertyName="xsd:NCName"
 xsd:anyAttribute##other>
 jsdl.StringRangeType
 <xsd:any##other/>*
</and>
```

allocation element

Definition
This element specifies an exclusive allocation requirement on a given property of a resource. You can define an allocation requirement on the resource consumable attributes. For more information on consumable attributes, see Table 96 on page 631. The job can be run on the resource if the required value is available. While the job runs, it uses exclusively the required value of the resource property. When the job completes, it releases the property value. This element is optional and can be specified once.

Type The type of this element is jsdl:AllocationRequirementType.

Attributes
The following attribute is defined:

propertyName
: Is a string specifying the resource property that the requirement applies to. The type of this attribute is xsd:QName. This attribute is required.

Quantity
: Specify the quantity of the property which is to be allocated exclusively to the job. To specify the quantity of the property which is to be allocated, you can use a variable expression that can contain one variable reference, such as ${var}, or a double value.

Pseudo Schema

```
<allocation
 propertyName="xsd:QName"
 xsd:anyAttribute##other>
 jsdl:DoubleVariableExpressionType
 <xsd:any##other/>*
</and>
```

relationship element

Definition

This element specifies a requirement that the resource selected for running the job has a relationship with other resources that match certain additional criteria. A relationship is a direct association between a source and a target resource. This element is optional and can be specified once.

Type The type of this element is jsdl:RelationshipRequirementType.

Attributes

The following attributes are defined:

type Is a string specifying the required relationship type. The type of this attribute is xsd:NCName. This attribute is optional.

source Is a string specifying the ID of a **relatedResources** element. If this attribute is specified, the relationship requirement specifies that the resource must have at least one relationship directed from one or more resources matched by the **relatedResources** element to itself. If this attribute is not specified, then a target attribute must be present. The type of this attribute is xsd:IDREF. This attribute is optional.

target Is a string specifying the ID of a **relatedResources** element. If this attribute is specified, the relationship requirement specifies that the resource must have at least one relationship directed from itself to one or more resources matched by the **relatedResources** element. If this attribute is not specified, then a source attribute must be present. The type of this attribute is xsd:IDREF. This attribute is optional.

Pseudo Schema

```
<relationship
 type="xsd:NCName"
 source="xsd:IDREF"?
 target="xsd:IDREF"?
 xsd:anyAttribute##other>
 <xsd:any##other/>*
</relationship>
```

candidateResources element

This element is reserved for internal use.

Definition
This element specifies the set of resources, which may be selected from for running the job. If this element is specified, one or more resources from the set must be chosen to run the job. The resources are identified using the Endpoint Reference address (WS-Addressing EPR) of the Job Factory service managing the resource. The requirement combinations are matched in OR, that is at least one of them must be matched by the resource contained in the target resource. At least one of the resources listed must be available for the job to run. If none of the resources you specify is available when the job is submitted, the job waits for one of them to become available. If no resource becomes available before the timeout expires, the job fails. This element is optional and can be specified once.

Type The type of this element is jsdl:CandidateResourcesRequirementType. It contains the **endpointReference** element.

Attributes
No attributes are defined.

Pseudo Schema
```
<candidateResources
 xsd:anyAttribute##other>
 <endpointReference>wsa:EndpointReferenceType<endpointReference/>+
 <xsd:any##other>*
</candidateResources>
```

endpointReference element

This element is reserved for internal use.

Definition
This element specifies the Web Services Addressing endpoint reference of the Job Factory service managing the resource. This element is required and can be specified one or more times. This element is reserved for internal use.

Type The type of this element is wsa:EndpointReferenceType.

Attributes
No attributes are defined.

Pseudo Schema
```
<candidateResources
 xsd:anyAttribute##other>
 <endpointReference>wsa:EndpointReferenceType<endpointReference/>+
 <xsd:any##other>*
</candidateResources>
```

optimization element

Definition
This element specifies the optimization policies to be applied to the job. Depending on whether you select the objective or ewlm element, the resource selection policy changes. This element is optional and can be specified once. If no value is specified then the default load balancing policy to equalize the number jobs running on each resource is applied.

Type The type for this element is jsdl:OptimizationType. It can contain only one the following elements:

- objective
- ewlm

Attributes

The following attribute is defined:

name Specifies the name of the optimization policy. Supported values are as follows:

JPT_JSDLOptimizationPolicyType

If you use this option, you must specify the **objective** element. Using the **objective** element, you can specify resource properties to be maximized or minimized. When you define the **objective** element, dynamic workload broker runs the job on the resource matching the optimization requirement. For more information on the **objective** element, see “objective element.” This is the default value. See “Resources in the job definition” on page 630 for more information.

JPT_BestResource

Use this option so that the resource pool for the job is made up of only the best resources among the pool of resources that meet the specified policy. If you use this option, you must specify the **objective** element. Using the **objective** element, you can specify resource properties to be maximized or minimized. When you define the **objective** element, dynamic workload broker runs the job on the resource matching the optimization requirement. For more information on the **objective** element, see “objective element.” This is the default value. See “Resources in the job definition” on page 630 for more information.

JPT_EWLM

If you use this option, the **ewlm** element is automatically inserted. For more information on the **ewlm** element, see “ewlm element” on page 616.

Pseudo Schema

```
<optimization
 name="xsd:NCName"
 xsd:anyAttribute##other>
 <objective .../> |
 <ewlm .../>
 <xsd:any##other>*
</optimization>
```

objective element

Definition

This element specifies the objective to reach when performing optimization for the job. For example, if you select the **resourcePropertyName** CPU Utilization for **resourceType** Computer System with **propertyObjective** minimize, dynamic workload broker will try to run the job on the resource where the CPU utilization is lowest. This element is mutually exclusive with the **ewlm** element.

Type The type of this element is PropertyObjectiveType.

Attributes

The following attributes are defined:

propertyObjective

Is a string specifying the objective type. This attribute is required. Supported values are as follows:

minimize

Resources are assigned to the job with the objective to minimize the value of the specified resource type property. For example, you can choose this objective to assign the job to the resource where the cpu utilization is lowest.

maximize

Resources are assigned to the job with the objective to maximize the value of the specified resource type property. For example, you can choose this objective to assign the job to the resource where the processing speed is highest.

minimize.utilization

Resources are assigned to the job with the objective to minimize the usage of the specified resource type property. This attribute is available only for consumable properties. For a list of all consumable properties, see Table 96 on page 631. If you choose to minimize the utilization of the property consumption, the job is assigned to a resource where a lower quantity of the property is used.

maximize.utilization

Resources are assigned to the job with the objective to maximize the usage of the specified resource type property. This attribute is available only for consumable properties. For a list of all consumable properties, see Table 96 on page 631. For example, you might want to perform a stress test on a workstation by creating jobs where the CPU Utilization resource property for the Computer System resource type is set to Maximize Utilization. This would cause all jobs with this setting to be assigned to the workstation where the CPU utilization is higher, generating a loop.

resourceType

Is a string specifying the type of the resource that the policy applies to. If this element is not specified, the resource type ComputerSystem is assumed. The type of this attribute is xsd:QName. This attribute is optional.

resourcePropertyName

Is a string specifying the resource property that the policy applies to. The type of this attribute is xsd:QName. This attribute is required.

Note: When specifying an optimization requirement on any property of a resource type, you must have previously defined a requirement on that resource type. For example, if you want to optimize the total physical memory on an operating system, you must previously define a requirement on the Operating System resource type. This procedure does not apply to the Computer System resource type, because Computer System is the default resource type.

Pseudo Schema

```
<objective
 propertyObjective="minimize" | "maximize"
 resourceType="xsd:QName"?
 resourcePropertyName="xsd:QName"
 xsd:anyAttribute##other>
 <xsd:any##other>*
</objective>
```

ewlm element

Definition
This element specifies the optimization based on Enterprise Workload Manager resource weight calculation. dynamic workload broker will run the job on the best available resources as indicated by Enterprise Workload Manager. This element is optional and can be specified once. This element is mutually exclusive with the **objective** element.

Type The type of this element is jsdl:PropertyObjectiveType

Attributes
No attributes are defined.

Pseudo Schema

```
<ewlm>
 xsd:anyAttribute##other>
 <xsd:any##other>*
</ewlm>
```

scheduling element

Definition
This element specifies the scheduling parameters to be applied in running the job. This element is optional and can be specified once.

In the dynamic workload broker, this element corresponds to the Scheduling page. For more information on the Scheduling page, see the online help documentation.

Type The type of this element is jsdl:SchedulingType. It can contain the following elements:
- maximumResourceWaitingTime
- estimatedDuration
- priority

Attributes
No attributes are defined.

Pseudo Schema

```
<scheduling
 xsd:anyAttribute##other>
 <maximumResourceWaitingTime>xsd:duration<maximumResourceWaitingTime>?
 <estimatedDuration>xsd:duration<estimatedDuration>?
 <priority>xsd:unsignedint<priority>?
 <xsd:any##other>*
</objective>
```

maximumResourceWaitingTime element

Definition
This element specifies how long the dynamic workload broker server must

wait since the job submission before deciding there are no resources matching the requirements. This element is optional and can be specified once.

Type The type of this element is xsd:duration.

Attributes
No attributes are defined.

Pseudo Schema
```
<maximumResourceWaitingTime>?
 xsd:anyAttribute##other>
 <xsd:duration>*
<maximumResourceWaitingTime>
```

estimatedDuration element

Definition
This element specifies the job run estimated duration. This can be used by the dynamic workload broker server to plan the resource assignment. This element is optional and can be specified once.

Type The type of this element is xsd:duration.

Attributes
No attributes are defined.

Pseudo Schema
```
<estimatedDuration>?
 <xsd:anyAttribute##other>
 <xsd:duration>*
<estimatedDuration>
```

priority element

Definition
This element specifies the job priority as an integer between 0 and 100. Higher values mean higher priority. This element is optional and can be specified once.

Type The type of this element is xsd:PriorityType. It can contain the following elements:
- maximumResourceWaitingTime
- estimatedDuration
- priority

Attributes
No attributes are defined.

Pseudo Schema
```
<scheduling
 xsd:anyAttribute##other>
 <maximumResourceWaitingTime>xsd:duration<maximumResourceWaitingTime>?
 <estimatedDuration>xsd:duration<estimatedDuration>?
 <priority>xsd:unsignedint<priority>?
 <xsd:any##other>*
</objective>
```

recoveryActions

Definition
This element describes the list of recovery actions that dynamic workload broker must perform if the time interval specified in the

maximumResourceWaitingTime element expires and no resources matching the requirements have been found. This element is optional and can be specified once. The recovery actions defined in the this element are performed by starting a Tivoli Provisioning Manager workflow. For more information on the maximumResourceWaitingTime element, see "maximumResourceWaitingTime element" on page 616.

Type The type of this element is jsdl:RecoveryActionList. It can contain the action element.

Attributes
No attributes are defined.

Pseudo Schema
```
<recoveryActions
 xsd:anyAttribute##other>
 <action...>+
 <xsd:any##other/>*
</recoveryActions >
```

action

Definition
This element specifies the recovery action(s) to be performed. dynamic workload broker performs the actions listed sequentially based on the order in which they have been specified. Any subsequent recovery action is run only if the previous action completed successfully. If the **recoveryActions** element is specified, at least one **action** element must be specified.

Type The type of this element is jsdl:RecoveryActionType.

Attributes
The following attributes are defined:

name Specifies the name of the recovery action to be performed. This attribute is optional.

additionalTimeOnCompletion
Specifies the time interval dynamic workload broker must wait for the recovery action to become effective after completing. If this attribute is specified for a recovery action, the subsequent recovery action is performed only after the specified time interval expires. If the required resource becomes available before the interval expires, dynamic workload broker might decide to run the job before the action completes.

maximumExecutionTime
Specifies the expected time period dynamic workload broker waits for the recovery action to complete. If the recovery action is not completed when this timeout expires, the recovery procedure fails and the recovery sequence is stopped.

Pseudo Schema
```
< action>
 name="xsd:NCName"
 additionalTimeOnCompletion ="xsd:duration"?
 maximumExecutionTime ="xsd:duration"?
 xsd:anyAttribute##other>
 <xsd:any##other/>*
</ action >
```

tpmaction element

Definition
This element specifies the parameters required for running a Tivoli Provisioning Manager recovery action. This attribute is optional and can be specified once.

Type The type of this attribute is jsdltpm:TPMActionType. It can contain the following elements:
- parameters
- credential
- tpmaddress
- workFlow

Attributes
No attributes are defined.

Pseudo Schema
```
<tpmaction
 <parameters... />?
 <credential.../>?
 <tpmaddress.../>?
 workFlow="jsdl:StringVariableExpressionType"
</tpmaction>
```

parameters element

Definition
This element specifies the arguments to be used when running the Tivoli Provisioning Manager workflow. This element is optional and can be specified once.

Type The type of this attribute is jsdltpm:ParametersType.

Attributes
The following attributes is supported:

name Specifies the name of the attribute to be used when running the Tivoli Provisioning Manager workflow. This attribute is required and can be specified one or more times.

Pseudo Schema
```
<parameters
 <parameter... />+
</parameters>
```

credential element

Definition
This element specifies the credentials required for running the Tivoli Provisioning Manager workflow. This element is optional and can be specified once.

Type The type of this element is jsdl:CredentialType. It can contain the following elements:
- userName
- password

Attributes
No attributes are defined.

Pseudo Schema

```
<credential
 <userName> jsdl:NotEmptyStringVariableExpressionType</userName>
 <password> jsdl:StringVariableExpressionType </password>
</credential>
```

userName element

Definition
: This element specifies a user name defined on the target system, that is used to run the Tivoli Provisioning Manager workflow. This element is required if you use the credential element and can be specified once.

Type The type of this element is jsdl:NotEmptyStringVariableExpressionType.

Attributes
: No attributes are defined.

Pseudo Schema

```
<userName>
 xsd:anyAttribute##other>
 <jsdl:NotEmptyStringVariableExpressionType*
<userName>
```

password element

Definition
: This element specifies the password of the specified user that is used to run the Tivoli Provisioning Manager workflow on the target system. This element is required if you use the credential element and can be specified once.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
: No attributes are defined.

Pseudo Schema

```
<password>?
 xsd:anyAttribute##other>
 <jsdl:StringVariableExpressionType
<password>
```

tpmaddress element

Definition
: This element specifies the Tivoli Provisioning Manager address that must be used to call the Tivoli Provisioning Manager web service necessary for running the workflow. This element is optional and can be specified once.

Type The type of this element is jsdltpm:TPMAddressType.

Attributes
: The following attributes are defined:

host Specifies the host name of the Tivoli Provisioning Manager server to be used when running the recovery action.

port Specifies the port number of the Tivoli Provisioning Manager server to be used when running the recovery action.

Pseudo Schema

```
< tpmaddress
 <host... />
 <port... />
</ tpmaddress >
```

workflow element

Definition
This element specifies the name of the Tivoli Provisioning Manager workflow to be run. To specify the workflow name, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. This element is required.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema

```
<workflow>?
 xsd:anyAttribute##other>
 <jsdl:StringVariableExpressionType
<workflow>
```

executable element

Definition
This element specifies the parameters for running a native system command, that is executable files and scripts. You can also embed scripts in this element. This element is required.

Note: The following restrictions apply:

- On Windows systems, you can run scripts containing batch commands. Supported formats for scripts are:
  - .cmd
  - .bat
- On UNIX and Linux systems, only shell scripts are supported. At the beginning of the shell script, you must specify the command interpreter.
- On UNIX and Linux systems, commands contained in scripts must run in foreground. This means that you cannot use the "&" parameter in association with the command.
- On all supported platforms, you cannot include in jobs any commands starting a program with a graphic interface.

Type The type of this element is jsdle:ExecutableType. It can contain the following elements:

- script
- arguments
- environment
- credential

Attributes
The following attributes are defined:

path Is a string variable expression specifying the path name of the executable file to run. If the **script** element is not present, the **path** attribute must be specified. If the **script** element is present, the

path attribute cannot be specified. To specify the path, you can use a variable expression that can contain one or more variable references, such as ${var}, any character, and any string.

You must specify the file extension. If you want to run an executable file without specifying its extension, you can specify the executable file name in the **script** element, so that the file is run within the shell.

input Is a string variable expression specifying the standard input for the command. This attribute is an absolute path name or a path name relative to the working directory. To specify the path, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. This attribute is optional.

output
Is a string variable expression specifying the standard output for the command. This attribute is an absolute path name or a path name relative to the working directory. To specify the path, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. This attribute is optional.

error Is a string variable expression specifying the standard error for the command. This attribute is an absolute path name or a path name relative to the working directory. To specify the path, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. This attribute is optional.

workingDirectory
Is a string variable expression specifying the working directory required by the job to run. To specify the directory, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. This attribute is optional. If you do not specify this attribute, the job runs in the following directories, depending on the operating system:

On UNIX systems
The following cases apply:
- The job runs in the $HOME_DIRECTORY of the user who submits the job, if existing.
- If this directory does not exist, it runs on /root, if the user who submits the job has the required rights.
- If the user does not have the required rights, the job runs in the Tivoli Workload Scheduler agent working directory.

On Windows systems
The job runs in the Tivoli Workload Scheduler agent working directory.

script Specifies the script code to be run. To specify special characters required by scripting languages, the content of the script element can be specified with a CDATA section

Pseudo Schema

```
<executable
 path="jsdl:StringVariableExpressionType"
 input="jsdl:StringVariableExpressionType"?
 output="jsdl:StringVariableExpressionType"?
 error="jsdl:StringVariableExpressionType"?
 workingDirectory="jsdl:StringVariableExpressionType"?
 xsd:anyAttribute##other>
 <script ... />?
 <arguments .../>?
 <environment .../>?
 <credential .../>?
 <xsd:any##other>*
</executable>
```

script element

Definition
This element specifies the script code to be run. This element is optional and can be specified once.

Type The type of this element is xsd:string.

Attributes
No attributes are defined.

Pseudo Schema

```
<script>?
 xsd:anyAttribute##other>
 <xsd:string
<script>
```

arguments element

Definition
This element specifies the list of arguments as string variable expressions that are concatenated to produce the argument string to be passed to the command. This element is optional and can be specified once.

Type The type of this element is jsdle:ArgumentsType. It can contain the following element:

- value

Attributes
No attributes are defined.

Pseudo Schema

```
<arguments
 xsd:anyAttribute##other>
 <value>jsdl:StringVariableExpressionType</value>+
 <xsd:any##other>*
</arguments>
```

value element

Definition
This element specifies the value of the **arguments** element. To specify the value, you can use a variable expression that can contain one or more variable references, such as ${var}, any character, and any string. This element is required and can be specified one or more times.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema

```
<arguments
 xsd:anyAttribute##other>
 <value>jsdl:StringVariableExpressionType</value>+
 <xsd:any##other>*
</arguments>
```

Note: If you need to specify that a value consists of a blank space, you must enclose it within double quotes.

environment element

Definition
This element specifies a string variable expression of environment variables that will be defined for the job in the running environment. This element is optional and can be specified once.

Type The type of this element is jsdl:jsdle:EnvironmentType. It can contain the following element:

- variable

Attributes
No attributes are defined

Pseudo Schema

```
<environment
 xsd:anyAttribute##other>
 <variable name="xsd:string">jsdl:StringVariableExpressionType</variable>+
 <xsd:any##other>*
</environment>
```

Note: If you need to specify that a value consists of a blank space, you must enclose it within double quotes.

variable element

Definition
This element specifies a string variable expression of environment variables that will be defined for the job in the running environment. This element is optional and can be specified once.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
The following attributes are defined:

name Specifies the name of the variable.

value Specifies the value of the variable. To specify the variable value, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string.

credential element

Definition
This element specifies the security credential for running the command. Include this element when you want to specify a user or group name under which the executable or script runs on the target system that is different from the user or group name under which the workload agent runs. This element is optional and can be specified once.

Type The type of this element is jsdle:CredentialType. It can contain the following elements:

- userName
- groupName
- password

Attributes
No attributes are defined.

Pseudo Schema
```
<credential
 xsd:anyAttribute##other>
 <userName> jsdl:StringVariableExpressionType </userName>
 <groupName> jsdl:StringVariableExpressionType </groupName>
 <password> jsdl:StringVariableExpressionType </password>
 <xsd:any##other>*
</credential>
```

userName element

Definition
Is a string variable expression that specifies the user name of a user defined on the target system. The command runs using this user name. This element is required if you use the credential element and can be specified once. This might be either a UNIX or Windows user ID. To specify the user name, you can use a variable expression that can contain one or more variable references such as ${var}, optionally, in association with any character or with a simple string. If the application runs on a Windows system as a Windows domain user, specify the user name as follows:
```
domain_name\user_name
```

If the application runs as a local user, you can use the following format:
```
user_name
```

Type The type of this element is jsdl:NotEmptyStringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema
```
<credential
 xsd:anyAttribute##other>
 <userName> jsdl:StringVariableExpressionType </userName>
 <groupName> jsdl:StringVariableExpressionType </groupName>
 <password> jsdl:StringVariableExpressionType </password>
 <xsd:any##other>*
</credential>
```

groupName element

Definition
Is a string variable expression that specifies the name of the group to which the user belongs that is defined on the target system where the command runs. This element is optional and can be specified once. This element is ignored on Windows target systems. To specify the group name, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema

```
<credential
 xsd:anyAttribute##other>
 <userName> jsdl:StringVariableExpressionType </userName>
 <groupName> jsdl:StringVariableExpressionType </groupName>
 <password> jsdl:StringVariableExpressionType </password>
 <xsd:any##other>*
</credential>
```

password element

Definition
Is a string variable expression that defines the password of the specified user name that is used to run the command on the target system. This element is optional and can be specified once. This element is ignored on UNIX target systems. To specify the password, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema

```
<credential
 xsd:anyAttribute##other>
 <userName> jsdl:StringVariableExpressionType </userName>
 <groupName> jsdl:StringVariableExpressionType </groupName>
 <password> jsdl:StringVariableExpressionType </password>
 <xsd:any##other>*
</credential>
```

j2ee element

Definition
This element specifies the J2EE application information needed for the job. This element is optional and can be specified once. The J2EE operations you can perform vary depending on the scheduler type (direct or indirect) you select, and on whether or not you enable WebSphere Application Server or J2EE security.

Type The type of this element is jsdlj:J2EEType. It can contain the following elements:
- invoker
- jsm
- ejb
- credential

Attributes
No attributes are defined.

Pseudo Schema

```
<j2ee>?
 xsd:anyAttribute##other>
 <jsdlj:J2EEType
<j2ee>
```

invoker element

Definition
This element specifies whether indirect or direct invoker is to be used for

the J2EE application. This element is required and can be specified once. Selecting a direct invoker means that the Tivoli Workload Scheduler agent immediately forwards the job to the WebSphere Application Server instance components (EJB or JMS). Selecting an indirect invoker means that the Tivoli Workload Scheduler agent leverages an existing WebSphere® scheduling infrastructure already configured on the target WebSphere Application Server.

Type The type of this element is jsdlj:InvokerType.

Attributes
No attributes are defined.

Pseudo Schema
```
<invoker>?
 xsd:anyAttribute##other>
 <jsdlj:InvokerType
<invoker>
```

jms element

Definition
This element specifies the target Java Message System (JMS) queue and the message to be sent. This element is optional and can be specified once. It is mutually exclusive with the **ejb** element.

Type The type of this element is jsdlj:JMSActionType.

Attributes
No attributes are defined.

Pseudo Schema
```
<jms>?
 xsd:anyAttribute##other>
 <jsdlj:JMSActionType
<jms>
```

ejb element

Definition
This element specifies the characteristics of the JNDI home of the EJB to be called. It is mutually exclusive with the **jms** element. The EJB must be already installed in the target WAS and must implement the TaskHandler interface.

Type The type of this element is jsdlj:EJBActionType. It can contain the following elements:
- jndiHome
- credential

Attributes
No attributes are defined.

Pseudo Schema
```
<ejb>?
 xsd:anyAttribute##other>
 <jsdlj:EJBActionType
<ejb>
```

jndiHome element

Definition
This element specifies the home directory of the Java Naming and Directory Interface (JNDI) application programming interface. This element is required and can be specified once.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema
```
<jndiHome>?
 xsd:anyAttribute##other>
 <jsdl:StringVariableExpressionType
<jndiHome>
```

ejb element

Definition
This element specifies the characteristics of the EJB action.

Type The type of this element is jsdlj:JMSActionType. It can contain the following elements:
- connFactory
- destination
- message
- credential

Attributes
No attributes are defined.

Pseudo Schema
```
<jms>?
 xsd:anyAttribute##other>
 <jsdl:JMSActionType
<jms>
```

connFactory element

Definition
This element specifies an administered object that a client uses to create a connection to the JMS provider. This element is required and can be specified once.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema
```
<connFactory>
 xsd:anyAttribute##other>
 <jsdl:StringVariableExpressionType
<connFactory>
```

destination element

Definition
This element specifies an administered object that encapsulates the identity of a message destination, which is where messages are delivered and consumed. This element is required and can be specified once.

Multiplicity

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema

message element

Definition
This element specifies an object that is sent from one application to another. This element is required and can be specified once.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema
```
<message>
 xsd:anyAttribute##other>
 <jsdl:StringVariableExpressionType
<message>
```

credential element

Definition
This element specifies the credentials required for running the J2EE application. Include this element when you want to specify a user name under which the application runs on the target system that is different from the user name under which the workload agent runs. This element is optional and can be specified once.

Type The type of this element is jsdl:CredentialType. It can contain the following elements:
- userName
- password
- JAASAuthenticationAlias

Attributes
No attributes are defined.

Pseudo Schema
```
<credential>?
 xsd:anyAttribute##other>
 <jsdl:CredentialType
<credential>
```

userName element

Definition
This element specifies the user name of a user defined on the target system. The J2EE application runs using this user name. This element is required if you use the credential element and can be specified once. To specify the user name, you can use a variable expression that can contain one or more variable references such as ${var}, optionally in association with any character or with a simple string. If you choose an indirect invoker, use this element to specify the user name required to connect to the WebSphere Application Server scheduler.

Type The type of this element is jsdl:NotEmptyStringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema
```
<userName>
 xsd:anyAttribute##other>
 <jsdl:NotEmptyStringVariableExpressionType
<userName>
```

password element

Definition
This element specifies the password of the specified user name that is used to run the J2EE application on the target system. This element is optional and can be specified once. To specify the password, you can use a variable expression that can contain one ore more variable references, such as ${var}, any character, and any string. If you choose an indirect invoker, use this element to specify the password required to connect to the WebSphere Application Server scheduler.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema
```
<password>
 xsd:anyAttribute##other>
 <jsdl:StringVariableExpressionType
<password>
```

JAASAuthenticationAlias element

Definition
This element specifies the JAAS authentication alias. This element is optional and can be specified once. It is required only when using an indirect invoker. To specify the alias, you can use a variable expression that can contain one or more variable references such as ${var}, optionally in association with any character or with a simple string.

Type The type of this element is jsdl:StringVariableExpressionType.

Attributes
No attributes are defined.

Pseudo Schema
```
<JAASAuthenticationAlias>
 xsd:anyAttribute##other>
 <jsdl:NotEmptyStringVariableExpressionType
<JAASAuthenticationAlias>
```

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

Table 96. 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
FileSystem	DisplayName	No	No	Yes
	FileSystemRoot	No	No	Yes
	FileSystemType	No	No	No
	FreeStorageCapacity	No	Yes	No
	TotalStorageCapacity	Yes	Yes	No

Table 96. Resource types and properties (continued)

Resource Type	Available properties	Is consumable	Can be optimized	Supports wildcards
NetworkSystem	NetworkAddress	No	No	No
	NetworkSystem HostName	No	No	Yes

Appendix C. Quick reference for commands

This appendix is divided into four sections:

- "Managing the plan"
- "Managing objects in the database" on page 634
- "Managing objects in the plan" on page 642
- "Utility commands" on page 647
- "Report commands" on page 650

Managing the plan

This section describes the operations you can perform against the plan using the **JnextPlan** script and the **planman** command line:

Table 97. Commands used against the plan

Command or script syntax	**Action performed**
JnextPlan [**-from** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]*] {**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* | **-for** *[h]hhmm* [**-days** *n*] | **-days** *n*}	Creates or extends the production plan.
planman [*connection_parameters*] **crt** [**-from** *mm/dd/*[*yy*]*yy* [*hhmm* [**tz** | **timezone** *tzname*]]] {**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* | **-for** *[h]hhmm* [**-days** *n*] | **-days** *n*}	Creates an intermediate production plan.
planman [*connection_parameters*] **deploy** [**-scratch**]	Deploys all rules that are not in draft state.
planman [*connection_parameters*] **ext** {**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* | **-for** *[h]hhmm* [**-days** *n*] | **-days** *n*}	Creates an intermediate plan for a plan extension.
planman [*connection_parameters*] **showinfo**	Retrieves the production plan information.
planman [*connection_parameters*] **crttrial** *file_name* [**-from** *mm/dd/*[*yy*]*yy* [*hhmm* [**tz** | **timezone** *tzname*]]] {**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* | **-for** *[h]hhmm* [**-days** *n*] | **-days** *n*}	Creates a trial plan.
planman [*connection_parameters*] **exttrial** *file_name* {**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* | **-for** *[h]hhmm* [**-days** *n*] | **-days** *n*}	Creates a trial plan of a production plan extension.

Table 97. Commands used against the plan (continued)

Command or script syntax	Action performed
planman [*connection_parameters*] **crtfc** *file_name* [**-from** *mm/dd/[yy]yy* [*hhmm* [**tz** | **timezone** *tzname*]]] {**-to** *mm/dd/[yy]yy[hhmm[**tz** | **timezone** tzname]]* | **-for** *[h]hhmm* [**-days** *n*] | **-days** *n*}	Creates a forecast plan.
planman [*connection_parameters*] **unlock**	Unlocks the production plan.
ResetPlan [*connection_parameters*] [**-scratch**]	Resets the production plan.

where *connection_parameters* are the following:

```
[-file filename]
[-host hostname]
[-port port_name]
[-protocol protocol_name][-proxy proxy_name]
[-proxyport proxy_port_number]
[-password user_password]
[-timeout seconds]
[-username user_name]
```

For more information, see "Creating and extending the production plan" on page 70.

Managing objects in the database

The section is divided into the following subsections:
- "General purpose commands"
- "Scheduling objects" on page 635
- "Composer commands" on page 639

General purpose commands

This section describes the names, the syntax of general purpose commands that are run from the **composer** program, and the user authorization, when needed, that is necessary to run them.

Table 98. General purpose commands

Command	Syntax	User Authorization
continue	**continue**&*command argument*&*command argument*	Authorization for using composer
edit	**edit** *filename*	Authorization for using composer
exit	**exit**	Authorization for using composer
help	**help** *commandname*	Authorization for using composer
redo	**redo** *directives*	Authorization for using composer

Table 98. General purpose commands (continued)

Command	Syntax	User Authorization
validate	**validate** *filename* [;syntax]	Authorization for using composer
version	**version**	Authorization for using composer

Scheduling objects

This section contains all scheduling objects definition syntax.

In the table displaying the list of commands that can be used against the scheduling object, *filename* indicates an existing file when used in the syntax for the **add** and **replace** commands, it indicates a not existing file when used in the syntax for the **create/extract** command.

Calendar

File definition syntax:

```
$calendar
calendarname ["description"]
    date [...]
```

Domain

File definition syntax:

```
domain domainname[description "description"]
   * manager workstation
    [parent domainname | ismaster]
end
```

Event rule

XML definition syntax:

- eventRule name=" " ruleType=" " isDraft=" " (1, 1)
  - description (0, 1)
  - timeZone (0, 1)
  - validity from=" " to=" " (0, 1)
  - activeTime start=" " end=" " (0, 1)
  - timeInterval amount=" " unit=" " (0, 1)
  - eventCondition eventProvider=" " eventType=" " (1, n)
    - scope (0, 1)
    - filteringPredicate (0, 1)
      - attributeFilter name=" " operator="eq" (0, n)
        - value (1, n)
      - attributeFilter name=" " operator="ne" (0, n)
        - value (1, n)
      - attributeFilter name=" " operator="le" (0, n)
        - value (1, 1)
      - attributeFilter name=" " operator="ge" (0, n)
        - value (1, 1)
      - attributeFilter name=" " operator="range" (0, 1)
        - value (1, 2)
  - correlationAttributes (0, 1)

    - attribute name=" " (1, n)
  - action actionProvider=" " actionType=" " responseType=" " (0, n)
    - description (0, 1)
    - scope (0, 1)
    - parameter name=" "(1, n)
    - value (1, 1)

Job

File definition syntax:

```
$jobs
[workstation#]jobname
   {scriptname filename | docommand "command" | task job_definition}
   streamlogon username
   [description "description"]
   [tasktype tasktype]
   [interactive]1
   [rccondsucc "Success Condition"]
   [recovery
        {stop | continue | rerun}
        [after [workstation#]jobname]
        [abendprompt "text"] ]
```

Note:

1. This keyword is available on Windows platforms only.

Job stream

File definition syntax:

```
schedule [workstation#]jobstreamname
   # comment
   [validfrom date]
   [timezone|tz tzname]
   [description "text"]
   [draft]
   [vartable table_name]
   [freedays calendarname [-sa] [-su]]
   [on [runcycle name]
      [validfrom date] [validto date]
      [description "text"]
      [vartable table_name]
     {date|day|calendar|request|"icalendar"} [,...]
      [fdignore|fdnext|fdprev]
      [({at time [+n day[s]] |
      schedtime time [+n day[s]]}
      [until time [+n day[s]] [onuntil action]]
      [deadline time [+n day[s]]])]]
   [,...]
   [except [runcycle name]
         [validfrom date] [validto date]
         [description "text"]
         {date|day|calendar|request|"icalendar"} [,...]
         [fdignore|fdnext|fdprev]
         [{(at time [+n day[s]])] |
         (schedtime time [+n day[s]])}]
   [,...]
```

[{**at** *time* [***timezone*|*tz*** *tzname*] [+*n* **day**[s][**absolute**|**abs**]] |
schedtime *time* [***timezone*|*tz*** *tzname*] [+*n* **day**[s]]}]
[**until** *time* [**timezone**|**tz** *tzname*] [+*n* **day**[s]] [**onuntil** *action*][**absolute**|**abs**]]
[**deadline** *time* [**timezone**|**tz** *tzname*] [+*n* **day**[s]]]
[**carryforward**]
[**matching** {**previous**|**sameday**|**relative from** [+ | -] *time* **to** [+ | -] *time*|
from *time* [+ | -*n* day[s]] **to** *time* [+ *n* day[s]] [**,...**]}]
[**follows** {[*netagent::*][*workstation#*]*jobstreamname*[**.***jobname* |@] [**previous**|
sameday|**relative from** [+|-] *time* **to** [+|-] *time*|
from *time* [+|-*n* day[s]] **to** *time* [+|-*n* day[s]]
]}] [,...]] [...]
[**keysched**]
[**limit** *joblimit*]
[**needs** { [*n*] [*workstation#*]*resourcename* } [,...]] [...]
[**opens** { [*workstation#*]"*filename*" [(*qualifier*)] [,...] }] [...]
[**priority** *number* | **hi** | **go**]
[**prompt** {*promptname*|"[**:**|**!**]*text*"} [,...]] [...]
:
job-statement
# *comment*
[{**at** *time* [**timezone**|**tz** *tzname*] [+*n* **day**[s]] |
schedtime *time* [**timezone**|**tz** *tzname*] [+*n* **day**[s]]}][**,...**]
[**until** *time* [**timezone**|**tz** *tzname*] [+*n* **day**[s]] [**onuntil** *action*]
[**deadline** *time* [**timezone**|**tz** *tzname*] [+*n* **day**[s]]] [**,...**]
[**every** *rate*]
[**follows** {[*netagent::*][*workstation#*]*jobstreamname*{.*jobname* @} [**previous**|
sameday|**relative from** [+|-] *time* **to** [+|-] *time* |
from *time* [+|-*n* day[s]] **to** *time* [+|-*n* day[s]]
]}] [,...]] [...]
[**confirmed**]
[**critical**]
[**keyjob**]
[**needs** { [*n*] [*workstation#*]*resourcename* } [,...]] [...]
[**opens** { [*workstation#*]"*filename*" [(*qualifier*)] [,...] }] [...]
[**priority** *number* | **hi** | **go**]
[**prompt** {*promptname*|"[**:**|**!**]*text*"} [,...]] [...]

[*job-statement***...**]
end

Parameter

File definition syntax:

$parm
[*tablename.*]*variablename* "*variablevalue*"

Prompt

File definition syntax:

$prompt
promptname "[**:** | **!**]*text*"

Resource

File definition syntax:

$resource
*workstation***#***resourcename units* [*"description"*]

Variable table

File definition syntax:

vartable *tablename*
[**description** *"description"*]
[**isdefault**]
members
[*variablename "variablevalue"*]
...
[*variablename "variablevalue"*]
end

Workstation

File definition syntax:

cpuname *workstation* [**description** "*description*"]
[**vartable** *table_name*]
os *os-type*
[**node** *hostname*] [**tcpaddr** *port*]
[**secureaddr** *port*] [**timezone|tz** *tzname*]
[**domain** *domainname*]
[**for maestro** [**host** *host-workstation* [**access** *method*]]
    [**type fta** | **s-agent** | **x-agent** | **manager** | **broker** | **agent** |
    pool | **d-pool** | **rem-engine**]
    [**ignore**]
    [**autolink on** | **off**]
    [**behindfirewall on** | **off**]
    [**securitylevel enabled** | **on** | **force**]
    [**fullstatus on** | **off**]
    [**server** *serverid*]]
    [**protocol** *http* | *htpps*]
    [**members** [*workstation*] [...]]
[**requirements** *jsdl_definition*]]
end

cpuname *workstation* [**description** *description*]
[**vartable** *table_name*]
os *os-type*
node *hostname* [**tcpaddr** *port*]
[**secureaddr** *port*] [**timezone|tz** *tzname*]
[**domain** *domainname*]
[**for maestro** [**host** *host-workstation* [**access** *method*]]
    [**type fta** | **s-agent** | **x-agent** | **manager**]
    [**ignore**]
    [**autolink on** | **off**]
    [**behindfirewall on** | **off**]
    [**securitylevel enabled** | **on** | **force**]
    [**fullstatus on** | **off**]
    [**server** *serverid*]]
end

Workstation class

File definition syntax:

```
cpuclass workstationclass [description "description"]
                              [ignore]
      members [workstation | @] [...]
end
```

User definition

File definition syntax:

```
username[workstation#][domain\]username
      password "password"end
```

Composer commands

This section describes the operations you can perform in the database using the **composer** command line interface program with syntax:

```
composer [connection_parameters] [-defaultws twscpu]
       ["command[&[command]][...]"]
```

where *connection_parameters*, if they are not supplied in the `localopts` or `useropts` files, are the following:

```
[-file filename]|
[-host hostname]
[-port port_name]
[-protocol protocol_name]
[-proxy proxy_name]
[-proxyport proxy_port_number]
[-password user_password]
[-timeout seconds]
[-username user_name]
```

See "Setting up options for using the user interfaces" on page 45 for more details.

These operations can only be run from any composer client command line installed.

In Table 99 displaying the list of commands that can be used against the scheduling object, *filename* indicates an existing file when used in the syntax for the **add** and **replace** commands, it indicates a not existing file when used in the syntax for the **create/extract** command.

Table 99. Composer commands

Command	Syntax	User Authorization
add	{**add** | **a**} *filename* [**;unlock**]	*add* or *modify*
authenticate	{**authenticate** | **au**} [**username=***username* **password=***password*]	
continue	{**continue** | **c**}	

Table 99. Composer commands (continued)

Command	Syntax	User Authorization
create extract	{**create** | **cr** | **extract** | **ext**} *filename* **from** {[**calendars** | **calendar** | **cal**=*calname*] | [**eventrule** | **erule** | **er**=*eventrulename*] | [**parms** | **parm** | **vb**=[*tablename.*]*variablename*] | [**vartable** | **vt**=*tablename*] | [**prompts** | **prom**=*promptname*] | [**resources** | **resource** | **res**=[*workstationname#*]*resourcename*] | [**cpu**={*workstationname* | *workstationclassname* | *domainname*}] | [**workstation** | **ws**=*workstationname*] | [**workstationclass** | **wscl**=*workstationclassname*] | [**domain** | **dom**=*domainname*] | [**jobs** | **jobdefinition** | **jd**=[*workstationname#*]*jobname*] | [**sched** | **jobstream** | **js**= [*workstationname#*]*jstreamname* *[**valid from** date|**valid to** date |**valid in** date date]* *[**;full**]*] | [**users** | **user**=[*workstationname#*]*username* [;password]]} [**;lock**]	*display*
delete	{**delete** | **de**} {[**calendars** | **calendar** | **cal**=*calname*] | [**eventrule** | **erule** | **er**=*eventrulename*] | [**parms** | **parm** | **vb**=[*tablename.*]*variablename*] | [**vartable** | **vt**=*tablename*] | [**prompts** | **prom**=*promptname*] | [**resources** | **resource** | **res**=[*workstationname#*]*resourcename*] | [**cpu**={*workstationname* [**;force**] | *workstationclassname* [**;force**]| *domainname*}] [**workstation** | **ws**=*workstationname*] [**;force**] | [**workstationclass** | **wscl**=*workstationclassname*] [**;force**] | [**domain** | **dom**=*domainname*] | [**jobs** | **jobdefinition** | **jd**=[*workstationname#*]*jobname*] | [**sched** | **jobstream** | **js**= [*workstationname#*]*jstreamname* *[**valid from** date|**valid to** date |**valid in** date date]*] | [**users** | **user**=[*workstationname#*]*username*]} [**;noask**]	*delete*
display	{**display** | **di**} {[**calendars** | **calendar** | **cal**=*calname*] | [**eventrule** | **erule** | **er**=*eventrulename*] | [**parms** | **parm** | **vb**=*variablename.*]*variablename*] | [**vartable** | **vt**=*tablename*] | [**prompts** | **prom**=*promptname*] | [**resources** | **resource** | **res**=[*workstationname#*]*resourcename*] | [**cpu**={*workstationname* | *workstationclassname* | *domainname*}] [**workstation** | **ws**=*workstationname*] | [**workstationclass** | **wscl**=*workstationclassname*] | [**domain** | **dom**=*domainname*] | [**jobs** | **jobdefinition** | **jd**=[*workstationname#*]*jobname*] | [**sched** | **jobstream** | **js**= [*workstationname#*]*jstreamname* *[**valid from** date|**valid to** date |**valid in** date date]* *[**;full**]*] | [**users** | **user**=[*workstationname#*]*username*]} [**;offline**]	*display*
edit	{**edit** | **ed**} *filename*	
exit	{**exit** | **e**}	
list print	{**list** | **l**} {[**calendars** | **calendar** | **cal**=*calname*] | [**eventrule** | **erule** | **er**=*eventrulename*] | [**parms** | **parm** | **vb**=[*tablename.*]*variablename*] | [**vartable** | **vt**=*tablename*] | [**prompts** | **prom**=*promptname*] | [**resources** | **resource** | **res**=[*workstationname#*]*resourcename*] | [**cpu**={*workstationname* | *workstationclassname* | *domainname*}] [**workstation** | **ws**=*workstationname*] | [**workstationclass** | **wscl**=*workstationclassname*] | [**domain** | **dom**=*domainname*] | [**jobs** | **jobdefinition** | **jd**=[*workstationname#*]*jobname*] | [**sched** |**jobstream** | **js**= [*workstationname#*]*jstreamname* *[**valid from** date|* ***valid to** date |**valid in** date date]* | [**users** | **user**=[*workstationname#*]*username*]} [**;offline**]	*display*

Table 99. Composer commands (continued)

Command	Syntax	User Authorization
lock	{**lock** | **lo**} {[**calendars** | **calendar** | **cal**=*calname*] | [**eventrule** | **erule** | **er**=*eventrulename*] [**parms** | **parm** | **vb**=[*tablename.*]*variablename*] | [**vartable** | **vt**=*tablename*] | [**prompts** | **prom**=*promptname*] | [**resources** | **resource** | **res**=[*workstationame#*]*resourcename*] | [**cpu**={*workstationame* | *workstationclassname* | *domainame*}] [**workstation** | **ws**=*workstationame*] | [**workstationclass** | **wscl**=*workstationclassname*] | [**domain** | **dom**=*domainame*] | [**jobs** | **jobdefinition** | **jd**=[*workstationame#*]*jobname*] | [**sched** | **jobstream** | **js**= [*workstationame#*]*jstreamname* [***valid from*** *date* | ***valid to*** *date* | ***valid in*** *date date*]] | [**users** | **user**=[*workstationame#*]*username*]}	*modify*
modify	{**modify** | **m**} {[**calendars** | **calendar** | **cal**=*calname*] | [**eventrule** | **erule** | **er**=*eventrulename*] | [**parms** | **parm** | **vb**=[*tablename.*]*variablename*] | [**vartable** | **vt**=*tablename*] | [**prompts** | **prom**=*promptname*] | [**resources** | **resource** | **res**=[*workstationame#*]*resourcename*] | [**cpu**={*workstationame* | *workstationclassname* | *domainame*}] [**workstation** | **ws**=*workstationame*] | [**workstationclass** | **wscl**=*workstationclassname*] | [**domain** | **dom**=*domainame*] | [**jobs** | **jobdefinition** | **jd**=[*workstationame#*]*jobname*] | [**sched** | **jobstream** | **js**= [*workstationame#*]*jstreamname* [***valid from*** *date* | ***valid to*** *date* | ***valid in*** *date date*] [***;full***]] | [**users** | **user**=[*workstationame#*]*username*]}	*modify* or *add*
new	**new** [**calendar** | **domain** | **eventrule** | **job** | **jobstream** | **parameter** | **prompt** | **resource** | **user** | **vartable** | **workstation** | **workstation_class**]	*add* or *modify*
rename	{**rename** | **rn**} {**calendars** | **calendar** | **cal** | **parms** | **parm** | **vb** | **vartable** | **vt** | **prompts** | **prom** | **resorces** | **resource** | **res** | **workstation** | **ws** | **workstationclass** | **wscl** | **domain** | **dom** | **jobs** | **jobdefinition** | **jd** | **jobsched** | **jb** | **eventrule** | **erule** | **er** **sched** | **jobstream** | **js** | **users** | **user** } *old_object_identifier* *new_object_identifier*	*add* and *delete*
replace	{**replace** | **rep**} *filename* [**;unlock**]	*modify* or *add*

Table 99. Composer commands (continued)

Command	Syntax	User Authorization
unlock	{**unlock** | **u**} {[**calendars** | **calendar** | **cal**=*calname*] | [**eventrule** | **erule** | **er**=*eventrulename*] | [**parms** | **parm** | **vb**=[*tablename.*]*variablename*] | [**vartable** | **vt**=*tablename*] | [**prompts** | **prom**=*promptname*] | [**resources** | **resource** | **res**=[*workstationame#*]*resourcename*] | [**cpu**={*workstationame* | *workstationclassname* | *domainame*}] [**workstation** | **ws**=*workstationame*] | [**workstationclass** | **wscl**=*workstationclassname*] | [**domain** | **dom**=*domainame*] | [**jobs** | **jobdefinition** | **jd**=[*workstationame#*]*jobname*] | [**sched**|**jobstream**|**js**= [*workstationame#*]*jstreamname* [***valid from*** *date*|***valid to*** *date* |***valid in*** *date date*]] | [**users** | **user**=[*workstationame#*]*username*]} [**;forced**]	*modify* and *unlock*
validate	{**validate** | **val**} *filename* [**;syntax**]	

Managing objects in the plan

This section describes the operations you can perform against the plan using the **conman** command line interface program with syntax:

```
conman ["command[&[command]...] [&]"]
```

Conman commands

This section lists the commands you can run from the **conman** program.

This is how you access to the **conman** command line:

```
conman [connection_parameters] ["command[&[command]...] [&]"]
```

where *connection_parameters*, if they are not supplied in the `localopts` or `useropts` files, are the following:

```
[-file filename]
[-host hostname]
[-port port_name]
[-protocol protocol_name]
[-proxy proxy_name]
[-proxyport proxy_port_number]
[-password user_password]
[-timeout seconds]
[-username user_name]
```

See "Setting up options for using the user interfaces" on page 45 for more details.

This is how you select jobs in commands:

```
[workstation#]
{jobstreamname(hhmm[ date]) job|jobnumber}
[{+|~}jobqualifier[...]]
```

or:

```
[workstation#]
jobstream_id.
job
[{+|~}jobqualifier[...]]
;schedid
```

This is how you select job streams in commands:

```
[workstation#]
jobstreamname(hhmm[ date])
[{+|~}jobstreamqualifier[...]]
```

or:

```
[workstation#]
jobstream_id
 ;schedid
```

You can run these commands from different types of workstations. In this table:

F stands for domain managers and fault-tolerant agents.

S stands for standard agents.

For each command you find the name, the syntax, the type of workstations from where you can issue the command, and the needed authorization, if any.

Table 100. Commands that can be run from conman

Command	Syntax	Workstation types	User Authorization
adddep job	{**adddep job** | **adj**} *jobselect* ;*dependency*[;...] [;**noask**]	F	*adddep* - (*use* when using prompts and needs)
adddep sched	{**adddep sched** | **ads**} *jstreamselect* ;*dependency*[;...] [;**noask**]	F	*adddep* - (*use* when using prompts and needs)
altpass	**altpass** [*workstation*#] *username* [;"*password*"]	F	*altpass*
altpri	{**altpri** | **ap**} *jobselect* | *jstreamselect* [;*pri*] [;**noask**]	F	*altpri*
bulk_discovery	{**bulk_discovery** | **bulk**}	F	*display*
cancel job	{**cancel job** | **cj**} *jobselect* [;**pend**] [;**noask**]	F	*cancel*
cancel sched	{**cancel sched** | **cs**} *jstreamselect* [;**pend**] [;**noask**]	F	*cancel*
checkhealthstatus	{**checkhealthstatus** | **chs**} [*workstation*]	M,F,S	
confirm	{**confirm** | **conf**} *jobselect* ;{**succ** | **abend**} [;**noask**]	F	*confirm*
console	{**console** | **cons**} [**sess** | **sys**] [;**level**=*msglevel*]	F-S	*console*
continue	{**continue** | **cont**}	F-S	
deldep job	{**deldep job** | **ddj**} *jobselect* ;*dependency*[;...] [;**noask**]	F	*deldep*
deldep sched	{**deldep sched** | **dds**} *jstreamselect* ;*dependency*[;...] [;**noask**]	F	*deldep*

Table 100. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
deployconf	{**deployconf** | **deploy**} [*domain***!**]*workstation*	F,S	Permission to *start* actions on *cpu* objects
display	{**display file** | **df**} *filename* [**;offline**] {**display job** | **dj**} *jobselect* [**;offline**] {**display sched** | **ds**} *jstreamselect* [**valid** {**at** *date* | **in** *date date*} [**;offline**]	F-S[1]	*display*
exit	{**exit** | **e**}	F-S	
fence	{**fence** | **f**} *workstation* **;***pri* [**;noask**]	F	*fence*
help (UNIX only)	{**help** | **h**} {*command*|*keyword*}	F-S	
kill	{**kill** | **k**} *jobselect* [**;noask**]	F	*kill*
limit cpu	{**limit cpu** | **lc** } *workstation* **;***limit* [**;noask**]	F	*limit*
limit sched	{**limit sched** | **ls** } *jstreamselect* **;***limit* [**;noask**]	F	*limit*
link	{**link** | **lk**} [*domain***!**]*workstation* [**;noask**]	F-S	*link*
listsym	{**listsym** | **lis**} [**trial** | **forecast**] [**;offline**]	F	
recall	{**recall** | **rc**} [*workstation*] [**;offline**]	F	*display*
redo	{**redo** | **red**}	F-S	
release job	{**release job** | **rj**} *jobselect* [**;***dependency*[**;**...]] [**;noask**]	F	*release*
release sched	{**release sched** | **rs**} *jstreamselect* [**;***dependency*[**;**...]] [**;noask**]	F	*release*

reply	{**reply** | **rep**}
	{ *promptname* | [*workstation***#**]*msgnum*}
	;*reply*
	[**;noask**]
		F	*reply*

rerun	{**rerun** | **rr**} *jobselect* [**;from=**[*wkstat***#**]*job*[**;at=***time*] [**;pri=***pri*]] [**;noask**] {**rerun**|**rr**} *jobselect* [**;step=***step*] [**;noask**]	F	*rerun*

Table 100. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
resource	{**resource** | **reso**} [*workstation*#] *resource*;*num* [;**noask**]	F	*resource*
setsym	{**setsym** | **set**} [**trial** | **forecast**] [*filenum*]	F	
showcpus	{**showcpus** | **sc**} [[*domain*!]*workstation*] [;**info**|;**link**] [;**offline**]	F-S	*list*2
showdomain	{**showdomain** | **showd**} [*domain*] [;**info**] [;**offline**]	F-S	*list*2
showfiles	{**showfiles** | **sf**} [[*workstation*#]*file*] [;*state*[;...]] [;**keys**] [;**offline**] {**showfiles** | **sf**} [[*workstation*#]*file*] [;*state*[;...]] [;**deps**[;**keys** | **info** | **logon**]] [;**offline**]	F	
showjobs	{**showjobs** | **sj**} [*jobselect*] [;**deps**[;**keys** | **info** | **logon**]] [;**short** | **single**] [;**offline**] [;**showid**] [;**props**] {**showjobs** | **sj**} [*jobselect* | [**workstation#**]*jobnumber.hhmm*] [;**stdlist**[;**keys**]] [;**short** | **single**] [;**offline**] [;**showid**] [;**props**]	F	*list*2
showprompts	{**showprompts** | **sp**} [*promptselect*] [;**keys**] [;**offline**] {**showprompts** | **sp**} [*promptselect*] [;**deps**[;**keys** | **info** | **logon**]][;**offline**]	F	*list*2
showresources	{**showresources** | **sr**} [[*workstation*#]*resourcename*] [;**keys**] [;**offline**] {**showresources** | **sr**} [[*workstation*#]*resourcename*] [;**deps**[;**keys** | **info** | **logon**]] [;**offline**]	F	*list*2
showschedules	{**showscheds** | **ss**} [*jstreamselect*] [;**keys**] [;**offline**] [;**showid**] {**showscheds** | **ss**} [*jstreamselect*] [;**deps**[;**keys** | **info** | **logon**]] [;**offline**] [;**showid**]	F	*list*2
shutdown	{**shutdown** | **shut**} [;**wait**]	F-S	*shutdown*

Table 100. Commands that can be run from conman (continued)

Command	**Syntax**	**Workstation types**	**User Authorization**
start	**start** [*domain*!]*workstation* [**;mgr**] [**;noask**] [**;demgr**]	F-S	*start*
startappserver	**startappserver** [*domain*!]*workstation* [**;wait**]	F-S	Permission to *start* actions on *cpu* objects
startevtp	{**starteventprocessor** | **startevtp**} [*domain*!]*workstation*	M[4]	Permission to *start* actions on *cpu* objects
startmon	{**startmon** | **startm**} [*domain*!]*workstation* [**;noask**]	F-S	Permission to *start* actions on *cpu* objects
status	{**status** | **stat**}	F-S	appserver
stop	**stop** [*domain*!]*workstation* [**;wait**] [**;noask**]	F-S	*stop*
stop ;progressive	**stop ;progressive**		*stop*
stopappserver	{**stopappserver** | **stopapps**} [*domain*!]*workstation* [**;wait**]	F-S	Permission to *stop* actions on *cpu* objects
stopevtp	{**stopeventprocessor** | **stopevtp**} [*domain*!][*workstation*]	M[4]	Permission to *stop* actions on *cpu* objects
stopmon	{**stopmon** | **stopm**} [*domain*!]*workstation* [**;wait**] [**;noask**]	F-S	Permission to *stop* actions on *cpu* objects
submit docommand	{**submit docommand** | **sbd**} [*workstation*#]"*cmd*" [**;alias**[=*name*]] [**;into=**[*workstation*#] {*jobstream_id***;schedid** |*jobstreamname* (*hhmm*[*date*])}] [;*joboption*[**;**...]]	F-S	*submit* - (*use* when using prompts and needs)
submit file	{**submit file** | **sbf**} "*filename*" [**;alias**[=*name*]] [**;into=**[*workstation*#]{*jobstream_id* **;schedid** |*jobstreamname* (*hhmm*[*date*])}] [;*joboption*[**;**...]] [**;noask**]	F-S	*submit* - (*use* when using prompts and needs)
submit job	{**submit job** | **sbj**} [*workstation*#]*jobname* [**;alias**[=*name*]] [**;into=**[*workstation*#]{*jobstream_id* **;schedid** | *jobstreamname*(*hhmm*[*date*])}] [;*joboption*[**;**...]] [**;vartable=***tablename*] [**;noask**]	F-S[3]	*submit* - (*use* when using prompts and needs)
submit sched	{**submit sched** | **sbs**} [*workstation*#]*jstreamname* [**;alias**[=*name*]] [;*jstreamoption*[**;**...]] [**;vartable=***tablename*] [**;noask**]	F-S[3]	*submit* - (*use* when using prompts and needs)

Table 100. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
switchevtp	{**switcheventprocessor** | **switchevtp**} *workstation*	M[4]	Permission to *start* and *stop* actions on *cpu* objects
switchmgr	{**switchmgr** | **switchm**} *domain***;***newmgr*	F	*start stop*
system	[**:** | **!**] *system-command*	F-S	
tellop	{**tellop** | **to**} [*text*]	F-S	
unlink	**unlink** [*domain***!**]*workstation* [**;noask**]	F-S	*unlink*
version	{**version** | **v**}	F-S	

where:

(1) Indicates that you can only display files on a standard agent.

(2) You must have **list** access to the object being shown if the ***enListSecChk*** option was set to **yes** on the master domain manager when the production plan was created or extended.

(3) Indicates that you can use submit job (**sbj**) and submit sched (**sbs**) on a standard agent by using the connection parameters or specifying the settings in the `useropts` file when invoking the **conman** command line.

(4) You can use this command on master domain managers and backup masters as well as on workstations installed as backup masters but used as ordinary fault-tolerant agents.

Utility commands

This section contains the list of the utility commands that you can run from the operating system command prompt. The utility commands are divided into three groups, those you can run on both UNIX and Windows operating systems, those you can run only on UNIX, and those you can run only on Windows.

Utility commands available for both UNIX and Windows operating systems

Table 101. Utility commands available for both UNIX and Windows

Command	Syntax
at	**at -V** | **-U** **at -s** *jstream* | **-q** *queue time-spec*
batch	**batch -V** | **-U** **batch** [**-s** *jstream*]
cpuinfo	**cpuinfo -V** | **-U** **cpuinfo** *workstation* [*infotype*] [...]

Table 101. Utility commands available for both UNIX and Windows (continued)

Command	**Syntax**
datecalc	**datecalc -V | -U** **datecalc** *base-date* [*offset*] [**pic** *format*][**freedays** *Calendar_Name* [**-sa**] [**-su**]] **datecalc -t** *time* [*base-date*] [*offset*] [**pic** *format*] **datecalc** *yyyymmddhhtt* [*offset*] [**pic** *format*]
delete	**delete -V | -U** **delete** *filename*
evtdef	**evtdef -U | -V** **evtdef** [*connection parameters*] **dumpdef** *file-path* **evtdef** [*connection parameters*] **loaddef** *file-path*
evtsize	**evtsize -V | -U** **evtsize** *filename size* **evtsize -compact** *filename* [*size*] **evtsize -info** *filename* **evtsize -show** *filename* **evtsize -info | -show pobox**
jobinfo	**jobinfo -V | -U** **jobinfo** *job-option* [...]
jobstdl	**jobstdl -V | -U** **jobstdl** [**-day** *num*] [{**-first** | **-last** | **-num** *n* | **-all**}] [**-twslog**] [{**-name** ["*jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"|*jobnum* | **-schedid** *jobstream_id.jobname*}]
maestro	**maestro** [**-V | -U**]
makecal	**makecal** [**-c** *name*] **-d** *n* | **-e** | {**-f 1 | 2 | 3 -s** *date*} | **-l** | **-m** | **-p** *n* | {**-r** *n* **-s** *date*} | **-w** *n* [**-i** *n*] [**-x | -z**][**-freedays** *Calendar_Name* [**-sa**] [**-su**]]
morestdl	**morestdl -V | -U** **morestdl** [**-day** *num*] [**-first** | **-last** | **-num** *n* | **-all**] [**-twslog**] [{**-name** ["*jobstreamname* [(*hhmm date*),(*jobstream_id*)].]*jobname*"|*jobnum* | **-schedid** *jobstream_id.jobname*}]

Table 101. Utility commands available for both UNIX and Windows (continued)

Command	**Syntax**
param	**param -u | -V** **param {-c | -ec}** [*file.section.*|*file.*|*section.*] *variable* [*value*] **param** [*file.section.*|*file.*|*section.*] *variable* **param {-d | -fd}** [*file.section.*|*file.*|*section.*] *variable*
parms	**parms** {[**-V | -U**] | **-build**} **parms {-replace | -extract}** *filename* **parms** [**-d**]*parametername***parms -c** *parametername value*
release	**release -V | -U** **release** [**-s**] [*workstation***#**]*resourcename* [*count*]
rmstdlist	**rmstdlist -V | -U** **rmstdlist** [**-p**] [*age*]
sendevent	**sendevent -V | ? | -help | -U | -usage** **sendevent** [**-hostname** *hostname*][**-port** *port*] *eventType source* [[*attribute=value*]...]
showexec	**showexec** [**-V | -U | INFO**]
ShutDownLwa	**ShutDownLwa**
StartUp	**StartUp** [**-V | -U**]
StartUpLwa	**StartUpLwa**
tws_inst_pull_info	**tws_inst_pull_info -twsuser** *userid* **-log_dir_base** *path* [**-u** | [**-run_db2_module** [y|n] | **-extract_db_defs** [y|n] | **-date** *yyyymmdd*]

Utility commands available for UNIX operating system only

Table 102. Utility commands available for UNIX only

Command	**Syntax**
at	**at -V | -U** **at -s***jstream* | **-q***queuetime-spec*
batch	**batch -V | -U** **batch** [**-s** *jstream*]
showexec	**showexec** [**-V | -U | -info**]
version	**version -V | -U | -h** **version** [**-a**] [**-f** *vfile*] [*file* [...]]

Utility commands available for Windows operating system only

Table 103. Utility commands available for Windows only

Command	Syntax
listproc **(UNSUPPORTED)**	**listproc**
killproc **(UNSUPPORTED)**	**killproc** *pid*
shutdown	**shutdown [-V | -U] [-appsrv]**

Report commands

This section contains a list and syntax of report commands and report extract programs. These commands are run from the operating system command prompt.

Report commands

Table 104. Report commands

Name	Output produced	Syntax
rep1	Reports 01 - Job Details Listing	**rep**[*x*] [**-V**|**-U**]
rep2	Report 02 - Prompt Listing	**rep**[*x*] [**-V**|**-U**]
rep3	Report 03 - Calendar Listing	**rep**[*x*] [**-V**|**-U**]
rep4a	Report 04A - Parameter Listing	**rep**[*x*] [**-V**|**-U**]
rep4b	Report 04B - Resource Listing	**rep**[*x*] [**-V**|**-U**]
rep7	Report 07 - Job History Listing	**rep7 -V**|**-U** **rep7** [**-c** *wkstat*] [**-s** *jstream_name*] [**-j** *job*] [**-f** *date* **-t** *date*] [**-l**]
rep8	Report 08 - Job Histogram	**rep8 -V**|**-U** **rep8** [**-f** *date* **-b** *time* **-t** *date* **-e** *time*] [**-i** *file*] [**-p**] **rep8** [**-b** *time* **-e** *time*] [**-i** *file*] [**-p**]
rep11	Report 11 - Planned Production Schedule	**rep11 -V**|**-U** **rep11** [**-m** *mm*[*yy*] [...]] [**-c** *wkstat* [...]] [**-s** *jstream_name*] [**-o** *output*]

Table 104. Report commands (continued)

Name	Output produced	Syntax
reptr	Report 09A - Planned Production Summary Report 09B - Planned Production Detail Report 10A - Actual Production Summary Report 10B - Actual Production Detail	**reptr** [**-V**|**-U**] **reptr -pre** [-{**summary** | **detail**}] [*symfile*] **reptr -post** [-{**summary** | **detail**}] [*logfile*]
xref	Report 12 - Cross Reference Report	**xref** [**-V**|**-U**] **xref** [**-cpu** *wkstat*] [**-s** *jstream_name*] [**-depends**|**-files**|**-jobs**|**-prompts**|**-resource**|**-schedules**|**-when** [...]]

Report extract programs

Table 105. Report extract programs

Extract Program	Used to generate	Syntax
jbxtract	Report 01 Report 07	**jbxtract** [**-V** | **-U**] [**-j** *job*] [**-c** *wkstat*] [**-o** *output*]
prxtract	Report 02	**prxtract** [**-V** | **-U**] [**-o** *output*] **prxtract** [**-V** | **-U**] [**-m** *mm*[*yyyy*]] [**-c** *wkstat*] [**-o** *output*]
caxtract	Report 03	**caxtract** [**-V** | **-U**] [**-o** *output*]
paxtract	Report 04A	**paxtract** [**-V** | **-U**] [**-o** *output*]
rextract	Report 04B	**rextract** [**-V** | **-U**] [**-o** *output*]
r11xtr	Report 11	**r11xtr** [**-V** | **-U**] [**-m** *mm*[*yyyy*]] [**-c** *wkstat*] [**-o** *output*] [**-s** *jstream_name*]
xrxtrct	Report 12	**xrxtrct** [**-V** | **-U**]

Appendix D. Accessibility

Accessibility features help users with physical disabilities, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in this product enable users to do the following:

- Use assistive technologies, such as screen-reader software and digital speech synthesizer, to hear what is displayed on the screen. Consult the product documentation of the assistive technology for details on using those technologies with this product.
- Operate specific or equivalent features using only the keyboard.
- Magnify what is displayed on the screen.

In addition, the product documentation was modified to include features to aid accessibility:

- All documentation is available in both HTML and convertible PDF formats to give the maximum opportunity for users to apply screen-reader software.
- All images in the documentation are provided with alternative text so that users with vision impairments can understand the contents of the images.

Navigating the interface using the keyboard

Standard shortcut and accelerator keys are used by the product and are documented by the operating system. Refer to the documentation provided by your operating system for more information.

The Event Rule Editor panel is the only one that does not allow keyboard-only operations and CSS cannot be disabled. However, as an alternative, you can perform all the operations available in this panel by launching the composer command from the command line interface.

Magnifying what is displayed on the screen

You can enlarge information on the product windows using facilities provided by the operating systems on which the product is run. For example, in a Microsoft Windows environment, you can lower the resolution of the screen to enlarge the font sizes of the text on the screen. Refer to the documentation provided by your operating system for more information.

Notices

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 publication 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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

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

Adobe, and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Index

Product Number: 5698-WSH

Printed in USA

SC32-1274-12

Spine information:

IBM Tivoli Workload Scheduler **Version 8.6 (Revised June 2012)** **User's Guide and Reference**