

Tivoli Workload Scheduler
Version 8.5.1 (Revised October 2010)

User's Guide and Reference

IBM

Tivoli Workload Scheduler
Version 8.5.1 (Revised October 2010)

User's Guide and Reference

IBM

Note

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

This edition applies to version 8, release 5 modification level 1, 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 SC32-1274-09.

© **Copyright IBM Corporation 1999, 2010.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	vii
--------------------------	------------

Tables	ix
-------------------------	-----------

About this publication xi

What is new in this release	xi
What is new in this publication	xi
What is new in this publication for version 8.5.1	xi
What this guide contains	xii
Publications	xiii
Accessibility	xiii
Tivoli technical training	xiii
Support information	xiv
Conventions used in this publication	xiv
Typeface conventions	xiv
Operating system-dependent variables and paths	xv
Command syntax	xv

Chapter 1. Tivoli Workload Scheduler overview 1

Understanding basic concepts	1
Tivoli Workload Scheduler database objects	1
The Tivoli Workload Scheduler network	10
Configuring your Tivoli Workload Scheduler runtime environment	10
Defining scheduling activities using Tivoli Workload Scheduler	11
Controlling job and job stream processing	12
Managing production scheduling activities with Tivoli Workload Scheduler	14
Automating workload using event rules	14
Tivoli Workload Scheduler user interfaces	15
Starting production	16

Chapter 2. Understanding basic workstation processes 21

Tivoli Workload Scheduler workstation processes	21
Starting and stopping processes on a workstation	26
Workstation inter-process communication	28
Tivoli Workload Scheduler network communication	29
Support for Internet Protocol version 6	31

Chapter 3. Configuring the job environment 33

Job environment overview	33
Environment variables exported by jobman	34
Customizing date formatting in the stdlist	36
Customizing job processing on a UNIX workstation - jobmanrc	37
Customizing the MAIL_ON_ABEND section of jobmanrc	38
Customizing job processing for a user on UNIX workstations - jobmanrc	39

Customizing job processing on a Windows workstation - jobmanrc.cmd	41
Setting up options for using the user interfaces	41

Chapter 4. Managing the production cycle 43

Plan management basic concepts	43
Preproduction plan	44
Identifying job stream instances in the plan	46
Managing external follows dependencies for jobs and job streams	46
Production plan	56
Understanding carry forward options	56
Trial plan	58
Forecast plan	59
Customizing plan management using global options	60
Generating a production plan - JnextPlan	63
Planman command line	66
Creating an intermediate production plan	68
Creating an intermediate plan for a plan extension	69
Retrieving the production plan information	70
Creating a trial plan	71
Creating a trial plan of a production plan extension	72
Creating a forecast plan	73
Deploying rules	74
Unlocking the production plan	75
Resetting the production plan	75
The stageman command	76
Managing concurrent accesses to the Symphony file	78
Scenario 1: Access to Symphony file locked by other Tivoli Workload Scheduler processes	78
Scenario 2: Access to Symphony file locked by stageman	78
Managing follows dependencies using carry forward prompt	78
The logman command	79
Starting production plan processing	81
Automating production plan processing	82

Chapter 5. Using workload service assurance 85

Enabling and configuring workload service assurance	86
Planning critical jobs	89
Processing and monitoring critical jobs	91
Workload service assurance scenario	93

Chapter 6. Customizing your workload using variable tables 95

The default variable table	96
Data integrity for variable tables	96
Locking mechanism for variable tables	97

Variable table security	97
Variable resolution	98

Chapter 7. Running event-driven workload automation 101

The event rule management process	104
Using the involved interfaces and commands	106
Defining event rules	107
Event rule examples	109
Rule operation notes	113
Triggered rule elements	116
Defining custom events	116

Chapter 8. Defining objects in the database 119

Defining scheduling objects.	119
Workstation definition	121
Workstation class definition	131
Domain definition	133
Job definition	134
Windows user definition	141
Calendar definition	143
Variable and parameter definition	144
Variable table definition	147
Prompt definition	149
Resource definition	151
Job stream definition	152
Job stream definition keyword details	157
Event rule definition	188

Chapter 9. Managing objects in the database - composer 199

Using the composer command-line program	199
Setting up the composer environment	199
Running the composer program	201
Running commands from composer	204
Filters and wild cards	204
Delimiters and special characters.	207
Command descriptions	207
Referential integrity check	209
add.	213
authenticate	215
continue	215
create - extract	216
delete	219
display	223
edit.	227
exit.	228
help	228
list, print	229
lock	233
modify	237
new	241
redo	243
rename	244
replace	247
system command	248
unlock.	248
validate	251
version	252

Chapter 10. Managing objects in the plan - conman 253

Using the conman command line program	253
Setting up the conman environment.	253
Running conman	254
Running commands from conman	257
Wildcards	258
Delimiters and special characters	258
Conman commands processing	259
Selecting jobs in commands	259
Syntax.	259
Arguments	260
Selecting job streams in commands	267
Syntax.	268
Arguments	268
Managing jobs and job streams from back-level agents	274
Command descriptions	274
adddep job	277
adddep sched	278
altpass.	280
altpri	281
bulk_discovery	282
cancel job.	282
cancel sched.	284
checkhealthstatus	285
confirm	286
console	287
continue	288
deldep job	288
deldep sched	290
deployconf	291
display	292
exit.	294
fence	295
help	296
kill	297
limit cpu	297
limit sched	299
link.	299
listsym	302
recall	303
redo	304
release job	305
release sched	307
reply	308
rerun	309
resource	312
setsym	313
showcpus	313
showdomain	319
showfiles	320
showjobs	322
showprompts	335
showresources	337
showschedules	340
shutdown	345
start	345
startappserver	348
starteventprocessor	348
startmon	349

status	349
stop	350
stop ,progressive	352
stopappserver	353
stopeventprocessor	354
stopmon	355
submit docommand	355
submit file	359
submit job	362
submit sched	365
switcheventprocessor	368
switchmgr	369
system	370
tellop	370
unlink	371
version	374

Chapter 11. Using utility commands 375

Command descriptions	375
at and batch	376
cpuinfo	379
datecalc	380
delete	384
evtdef	385
evtsize	388
jobinfo	389
jobstdl	391
maestro	394
makecal	394
metronome	396
morestdl	396
parms	398
release	400
rmstdlist	401
sendevent	402
showexec	403
shutdown	404
StartUp	405
1 tws_inst_pull_info	406
version	406
Unsupported commands	407

Chapter 12. Getting reports and statistics 409

Setup for using report commands	409
Changing the date format	410
Command descriptions	410
rep1 - rep4b	410
rep7	411
rep8	413
rep11	414
reptr	415
xref	416
Sample report outputs	417
Report 01 - Job Details Listing:	417
Report 02 - Prompt Listing:	419
Report 03 - Calendar Listing:	420
Report 04A - Parameter Listing:	420
Report 04B - Resource Listing:	421
Report 07 - Job History Listing:	421

Report 08 - Job Histogram:	422
Report 9B - Planned Production Detail:	422
Report 10B - Actual Production Detail:	423
Report 11 - Planned Production Schedule:	424
Report 12 - Cross Reference Report:	425
Report extract programs	427
jbxtract	427
prxtract	429
caxtract	429
paxtract	430
reextract	431
r11xtr	431
rxtrct	432
Running reports on the Tivoli Dynamic Workload Console	437
Historical reports	438
Production reports	441

Chapter 13. Managing time zones. 445

Enabling time zone management	445
How Tivoli Workload Scheduler manages time zones	446
Moving to daylight saving time on	448
Moving to daylight saving time off	448
General rules	448
Backward compatibility time zone table	449
Complete table of time zones with variable length notation	450

Chapter 14. The auditing feature 467

Auditing overview	467
Enabling the auditing feature	467
Audit log format	468
Audit log header format	468
Audit log body format	469
Sample audit log files	472

Chapter 15. Managing extended agents 475

What are extended agents?	475
Workstation definition	476
Access method interface	476
Method command line syntax	476
Method response messages	478
Method options file	479
Running methods	480
Launch job task (LJ)	480
Manage job task (MJ)	481
Check file task (CF)	481
Get status task (GS)	482
Cpuinfo command	483
Troubleshooting	483
Job standard list error messages	483
Method not executable	483
Console Manager messages	483
Composer and compiler messages	483
Jobman messages	484

Chapter 16. Managing internetwork dependencies 485

Internetwork dependencies overview	485
Understanding how an internetwork dependency is shown	486
Configuring a network agent	487
A sample network agent definition	487
Defining an internetwork dependency	489
A sample internetwork dependency definition	489
See Also	489
Managing internetwork dependencies	490
States of jobs defined in the EXTERNAL job stream.	490
Working with jobs defined in the EXTERNAL job stream	490
Sample internetwork dependency management scenarios	491
Internetwork dependencies in a mixed environment.	493

Appendix A. Event-driven workload automation event and action definitions. 495

Event providers and definitions	495
TWSObjectsMonitor events	495
FileMonitor events	520
TWSApplicationMonitor events	523
Action providers and definitions	525
TECEventForwarder actions	525

MailSender actions	526
MessageLogger actions	526
TWSAction actions	527
GenericAction actions	529

Appendix B. Web services interface 531

Appendix C. Quick reference for commands 533

Managing the plan	533
Managing objects in the database.	534
General purpose commands	534
Scheduling objects.	535
Composer commands	538
Managing objects in the plan	541
Conman commands	541
Utility commands	545
Report commands	547

Appendix D. Accessibility 551

Notices 553	
Trademarks	554

Index 557

Figures

1. Single-domain network	7	16. Sameday matching criteria - Step 1: at Start of	
2. Multiple-domain network	8	Day (SOD) on a Thursday	53
3. Process tree in UNIX	24	17. Sameday matching criteria - Step 2: at 9:00	53
4. Process tree in Windows	25	18. Sameday matching criteria - Step 3: at 15:00	53
5. Inter-process communication.	28	19. Relative Interval matching criteria - at start of	
6. Closest preceding matching criteria	47	day on Thursday.	54
7. Within a relative interval matching criteria	47	20. Absolute interval matching criteria - at start of	
8. Within an absolute interval matching criteria	48	day on Thursday.	56
9. Sameday matching criteria	48	21. Critical path	92
10. Closest preceding predecessor job	49	22. Network links	301
11. Pending predecessor instance	50	23. Started workstations in network	347
12. Closest preceding matching criteria - Step 1:		24. Stopped workstations in network	351
before 08:00	51	25. Unlinked network workstations	373
13. Closest preceding matching criteria - Step 2: at		26. Example when start of day conversion is not	
08:00 on weekdays except Thursdays and		applied	447
Fridays	51	27. Example when start of day conversion is	
14. Closest preceding matching criteria - Step 3: at		applied	447
09:00 on Thursdays and Fridays.	52	28. Local and remote networks	488
15. Closest preceding matching criteria - Step 4: at			
15:00 on every day	52		

Tables

1. Command syntax	xv	39. Scheduling objects filtering criteria	205
2. Starting and stopping Tivoli Workload Scheduler on a workstation	26	40. Delimiters and special characters for composer	207
3. Job environment variables for Windows	34	41. List of composer commands	208
4. Job environment variables for UNIX	35	42. Object identifiers for each type of object defined in the database	209
5. Variables defined by default in the jobmanrc file	37	43. Object definition update upon deletion of referenced object	210
6. Variables defined by default in the jobmanrc.cmd file	41	44. Referential integrity check when deleting an object from the database	210
7. Carry forward global options settings	57	45. Output formats for displaying scheduling objects	225
8. Resulting carry forward settings	57	46. Output formats for displaying scheduling objects	231
9. Workload service assurance global options	86	47. Delimiters and special characters for conman	258
10. Workload service assurance local options	88	48. List of conman commands	274
11. The relationship between variable tables and their enclosed variables in the Tivoli Workload Scheduler security file	98	49. State change after confirm command	286
12. Simple event rule scenarios	102	50. Opened links	301
13. conman commands for managing monitoring engines	104	51. Started workstations	347
14. conman commands for managing the event processing server	105	52. Stopped workstations	352
15. Interfaces and commands for managing event-driven workload automation	106	53. Stopped workstations with stop ;progressive	353
16. Event rule definition for scenario1	109	54. Unlinked workstations	373
17. Event rule definition for scenario2	110	55. List of utility commands	375
18. Event rule definition for scenario3	111	56. Date formats	410
19. Event rule definition for scenario4	112	57. List of report commands	410
20. Rule operation notes	114	58. Report extract programs	427
21. List of supported scheduling object keywords	120	59. Jbextract output fields	428
22. List of reserved words when defining jobs and job streams	120	60. Prxtract output fields	429
23. List of reserved words when defining workstations	121	61. Caxtract output fields	430
24. List of reserved words when defining users	121	62. Paxtract output fields	430
25. Attribute settings for management workstation types	122	63. Rextract output fields	431
26. Attribute settings for target workstation types	123	64. R11xtr output fields	432
27. Type of communication depending on the security level value	129	65. Xdep_job output fields	433
28. Comparison operators	137	66. Xdep_job output fields (continued)	434
29. Logical operators	138	67. Xdep_sched output fields	434
30. Recovery options and actions	139	68. Xfile output fields	435
31. How to handle a backslash in variable substitution	145	69. Xjob output fields	435
32. Keywords that can take local parameters in submit commands	146	70. Xprompts output fields	435
33. Required access keyword on variable table in the security file (vartable object) and allowed actions on its variable entries	149	71. Xresource output fields	436
34. List of scheduling keywords	153	72. Xsched output fields	436
35. Explanation of the notation defining the number of occurrences for a language element	189	73. Xwhen output fields	437
36. TWSObjectsMonitor events	192	74. Summary of historical reports	438
37. FileMonitor events	193	75. Summary of production reports	442
38. Action types by action provider	195	76. Backward compatibility time zone table	449
		77. Variable length notation time zones list	450
		78. Method command task options	477
		79. Launch job task (LJ) messages	481
		80. Check file task (CF) messages	482
		81. Get status task (GS) messages	482
		82. Internetwork dependencies in a mixed environment	493
		83. Parameters of JobStatusChanged event types	496
		84. Parameters of JobUntil event types	498
		85. Parameters of JobSubmit event types	500
		86. Parameters of JobCancel event types	501
		87. Parameters of JobRestart event types	502

88. Parameters of JobLate event types.	504	105. Parameters of TWSPProcessMonitor event	
89. Parameters of JobStreamStatusChanged event		types.	524
types.	506	106. Parameters of TWSDiskMonitor event types.	525
90. Parameters of JobStreamCompleted event		107. Parameters of TECFWD action types.	525
types.	508	108. Parameters of SendMail action types.	526
91. Parameters of JobStreamUntil event types.	509	109. Parameters of MSGLOG action types.	526
92. Parameters of JobStreamSubmit event types.	511	110. Parameters of SubmitJobStream action types.	527
93. Parameters of JobStreamCancel event types.	512	111. Parameters of SubmitJob action types.	528
94. Parameters of JobStreamLate event types.	513	112. Parameters of SubmitAdHocJob action types.	529
95. Parameters of WorkstationStatusChanged		113. Parameters of ReplyPrompt action types.	529
event types.	515	114. Parameters of RunCommand action types.	530
96. Parameters of		115. Available messages in the Web services	
ApplicationServerStatusChanged event types..	516	interface	532
97. Parameters of ChildWorkstationLinkChanged		116. Commands used against the plan.	533
event types.	517	117. General purpose commands	534
98. Parameters of ParentWorkstationLinkChanged		118. Commands used against scheduling objects in	
event types.	517	the database	539
99. Parameters of PromptStatusChanged event		119. Commands that can be run from conman	542
types.	518	120. Utility commands available for both UNIX	
100. Parameters of FileCreated event types.	520	and Windows	545
101. Parameters of FileDeleted event types.	521	121. Utility commands available for UNIX only	547
102. Parameters of ModificationCompleted event		122. Utility commands available for Windows only	547
types.	521	123. Report commands	547
103. Parameters of LogMessageWritten event		124. Report extract programs	548
types.	522		
104. Parameters of TWSMMessageQueues event			
types.	523		

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&tuid=swg24018908>.

What is new in this publication

This section describes what has changed in this publication since version 8.5.

Note: Throughout the document, changed or added text is marked by a revision bar in the left margin.

What is new in this publication for version 8.5.1

For version 8.5.1, the following changes were made:

- The commands to start and stop the Tivoli Workload Scheduler agent locally were added in Table 2 on page 26.
- Throughout the guide, and especially in the sections describing the definition of workstations in the Tivoli Workload Scheduler database, the term *workload broker agent* was changed to *workload broker workstation* to reflect the new name for the Tivoli Dynamic Workload Broker Bridge.

What is new in this publication for version 8.5.1.1

For version 8.5.1.1, the following changes were made:

- The command to launch the *chkhltst* service was added. The service helps you verify that the workstation mailbox can be successfully read, and check if there are errors in the mailbox header. Information about this command was added in the following sections: "checkhealthstatus" on page 285, "Command descriptions" on page 274, and "Conman commands" on page 541.
- Description of the "stopappserver" on page 353 command was improved with better documentation about the user credentials required to run the command.
- Description of the "submit job" on page 362 command was improved with better documentation about the authorization required in the security file for running the command.
- The documentation of the "evtsize" on page 388 was enhanced with new options.

- 1 • More information was added for the “jobstdl” on page 391 command.
 - 1 • The parameters of the “ModificationCompleted” on page 521 event type were
 - 1 updated.

 - 1 New and changed lines are marked by a 1 (one) on the page left margin
-

What this guide contains

This manual contains the following chapters and appendixes:

- Chapter 1, “Tivoli Workload Scheduler overview,” on page 1
Provides you with a quick introduction to Tivoli Workload Scheduler, its scheduling objects and its user interfaces, and explains the basic steps a new user should follow to begin to use the product for the first time.
- Chapter 2, “Understanding basic workstation processes,” on page 21
Describes workstation processes and network communication in a Tivoli Workload Scheduler environment during job processing.
- Chapter 3, “Configuring the job environment,” on page 33
Describes how to customize the way job management is performed on a workstation.
- Chapter 4, “Managing the production cycle,” on page 43
Describes how Tivoli Workload Scheduler performs plan management.
- Chapter 5, “Using workload service assurance,” on page 85
Describes how you can ensure that your mission-critical workload completes on time.
- Chapter 6, “Customizing your workload using variable tables,” on page 95
Is an overview of how you can use variable tables to manage variable substitution in run cycles, job streams, and workstations.
- Chapter 7, “Running event-driven workload automation,” on page 101
Provides an overview of event rule management.
- Chapter 8, “Defining objects in the database,” on page 119
Describes the scheduling objects, including job streams with the related keywords, that you can define in the Tivoli Workload Scheduler DB2 database.
- Chapter 9, “Managing objects in the database - composer,” on page 199
Explains the usage and the syntax of the commands used in the **composer** command-line interface to manage scheduling objects in the database.
- Chapter 10, “Managing objects in the plan - conman,” on page 253
Describes the usage and the syntax of the commands used in the **conman** command-line interface to monitor and manage job and job stream instances in production.
- Chapter 11, “Using utility commands,” on page 375
Describes the Tivoli Workload Scheduler utility commands that manage the environment.
- Chapter 12, “Getting reports and statistics,” on page 409
Describes how to use different types of reports.
- Chapter 13, “Managing time zones,” on page 445
Describes how to manage the time zones with Tivoli Workload Scheduler.
- Chapter 14, “The auditing feature,” on page 467

Describes how to enable and use the auditing option to track changes applied to the database and to the plan.

- Chapter 15, “Managing extended agents,” on page 475

Describes how to create and use extended agents, that is logical definitions, each hosted by a physical workstation, used to perform job processing where an agent is not installed.

- Chapter 16, “Managing internetwork dependencies,” on page 485

Describes how to create and use a network agent workstation to manage dependencies of locally defined jobs and job streams from jobs and job streams running in a remote Tivoli Workload Scheduler network.

- Appendix A, “Event-driven workload automation event and action definitions,” on page 495

Provides a complete description of all event and action types for use when defining event rules.

- Appendix B, “Web services interface,” on page 531

Gives an introduction to the use of the Tivoli Workload Scheduler Web Services Interface.

- Appendix C, “Quick reference for commands,” on page 533

Contains a quick reference for commands. It also includes utility and report commands.

- Appendix D, “Accessibility,” on page 551

Is an overview of the accessibility features of Tivoli Workload Scheduler.

Publications

Full details of Tivoli Workload Scheduler publications can be found in *Tivoli Workload Automation: Publications*. This document also contains information on 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 551.

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.	<i>[-file definition_file]</i>
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.	<i>prompt_name</i>
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
<i>Italic</i>	<i>Italic</i> text is variable and must be replaced by whatever it represents. In the example to the right, the user would replace <i>file_name</i> with the name of the specific file.	<i>file_name</i>

Table 1. Command syntax (continued)

Syntax convention	Description	Example
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.	<p><code>[-x file_name]...</code></p> <p>An ellipsis outside the brackets indicates that <code>-x file_name</code> is optional and may be repeated as follows: <code>-x file_name1 -x file_name2 -x file_name3</code></p> <p><code>[-x file_name...]</code></p> <p>An ellipsis inside the brackets indicates that <code>-x file_name</code> is optional, and the file variable can be repeated as follows: <code>-x file_name1 file_name2 file_name3</code></p> <p><code>-x file_name [-x file_name]...</code></p> <p>An ellipsis used with this syntax indicates that you must specify <code>-x file_name</code> at least once.</p>

Chapter 1. Tivoli Workload Scheduler overview

IBM Tivoli 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 15
- “Starting production” on page 16

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 10
- “Configuring your Tivoli Workload Scheduler runtime environment” on page 10
- “Defining scheduling activities using Tivoli Workload Scheduler” on page 11
- “Managing production scheduling activities with Tivoli Workload Scheduler” on page 14

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 3
- Workstation, see “Workstation” on page 4
- Workstation class, see “Workstation class” on page 5
- Domain, see “Domain” on page 6
- Event rule, see “Event rule” on page 9
- Resource, see “Resource” on page 9
- Parameter, see “Parameter” on page 9
- Variable table, see “Variable table” on page 9

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.

For information about how to define jobs, see “Job definition” on page 134.

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 152.

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 143.

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 149

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 5, and for domains, see “Domain” on page 6.

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. For example, the IP address of the workstation, whether it is behind a firewall, whether communications with it should be secure, what time zone it is in, and the identity of its domain manager.

Workstations in the Tivoli Workload Scheduler scheduling network can not 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.

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.

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.

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.

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.

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 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 Chapter 15, "Managing extended agents," on page 475 and the *Tivoli Workload Scheduler for Applications User's Guide*.

Workload broker agent

A workstation that manages the lifecycle of Tivoli Workload Scheduler "Workload Broker" type jobs in Tivoli Dynamic Workload Broker. It is installed and configured as a dynamic workload broker workstation in the database.

Tip: You can schedule jobs and job streams to run on any workstation, whatever its role.

For information about how to define workstations, see "Workstation definition" on page 121.

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 131.

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. All 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.

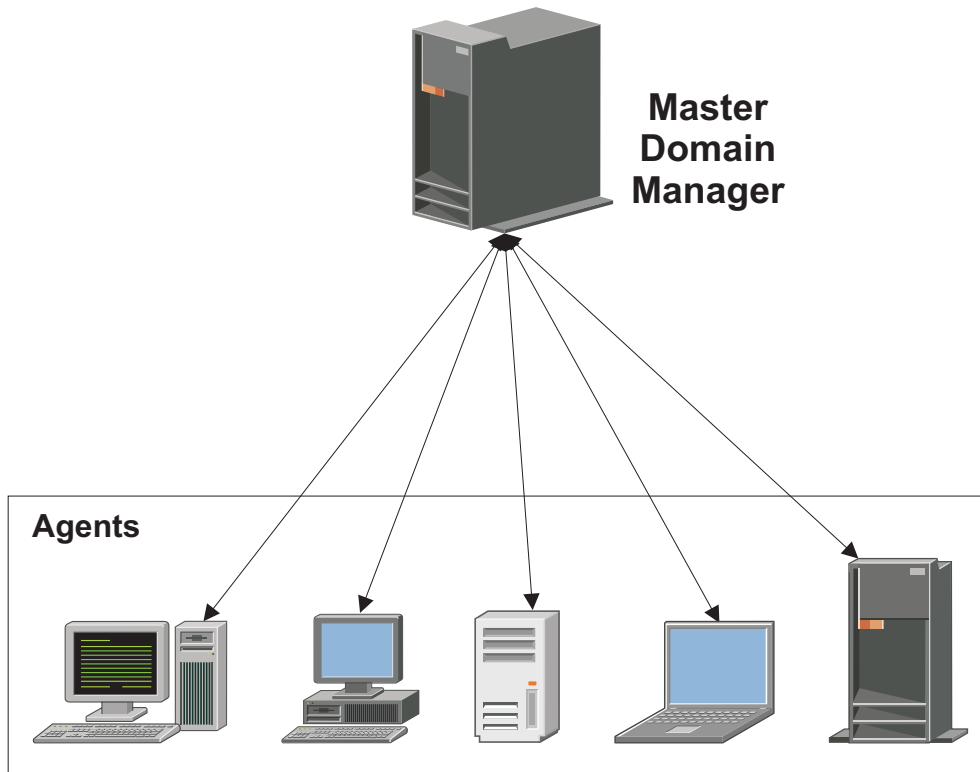


Figure 1. Single-domain network

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.

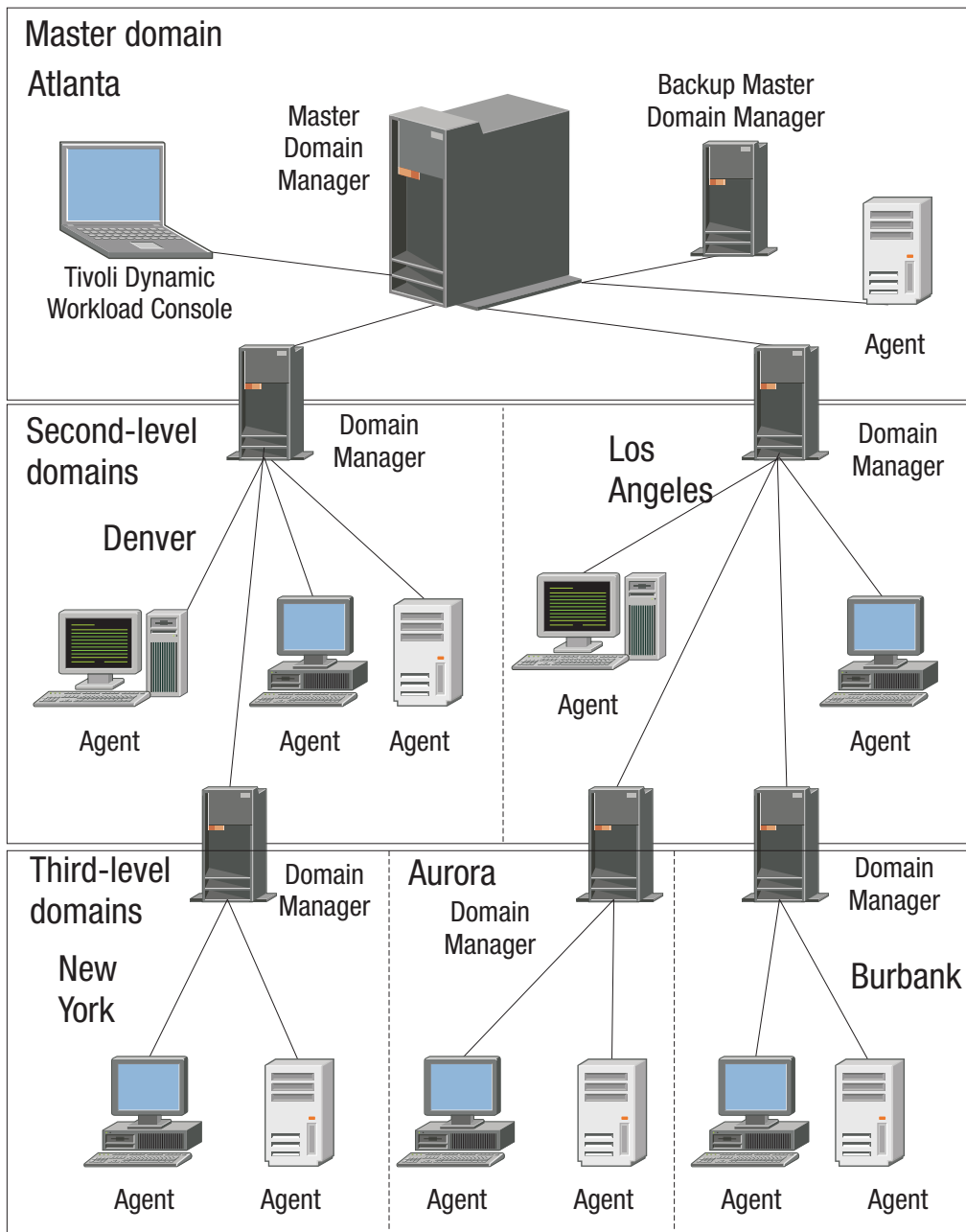


Figure 2. Multiple-domain network

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 5

For information about how to define domains, see “Domain definition” on page 133.

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 107.

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 151.

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 144.

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 147.

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.

For information about workstations, refer to “Workstation definition” on page 121.

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 43.

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 41.

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 one of the graphical user interfaces, the **Tivoli Dynamic Workload Console** or the **Job Scheduling 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 Tivoli Job Scheduling Console.
- The Tivoli 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 119.

For more information about report utility commands, refer to Chapter 12, “Getting reports and statistics,” on page 409.

For more information about the Job Scheduling Console and the Tivoli 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. You can choose from four different types of dependencies:

- **On completion of jobs and job streams:** a job or a job stream must not begin processing until other jobs and job streams have completed successfully. For more information, refer to “follows” on page 165.
- **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 172.
- **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 178.
- **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 149 and “prompt” on page 181.

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 host site1, dependent on the successful completion of job2, which runs on host site2.

Setting time constraints

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 157, “deadline” on page 160, “every” on page 162, “schedtime” on page 182, and “until” on page 185.

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 295 and “priority” on page 180.

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 297, and “limit sched” on page 299.

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 151.

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 286.

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 535.

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 e-mail
- 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 101.

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:

Tivoli 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 Tivoli 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 Tivoli 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

Tivoli 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 119 and Chapter 9, “Managing objects in the database - composer,” on page 199.

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 253.

Java API

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. Documentation for the API is provided on all product CDs. Mount the CD for your platform and open the following file with your Internet browser: *CD_drive/API/doc/index.html*. Click on the **Help** topic in the page header to learn more about how the documentation is structured.

Job Scheduling Console

An interactive graphical interface used to create, modify, and delete objects in the product database and in the plan. This interface is described in the *IBM Tivoli Workload Scheduler Job Scheduling Console: User's Guide*.

The Job Scheduling Console is being replaced by the Tivoli Dynamic Workload Console as the primary graphical user interface. It will not support any new functions beyond those of Tivoli Workload Scheduler version 8.4.

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 66.

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 Appendix B, "Web services interface," on page 531.

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 41.

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 Tivoli Dynamic Workload Console or the Job Scheduling 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 41.

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 133.

- **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 121. 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 “Windows user definition” on page 141.

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 143.

6. Optionally define parameters, prompts, and resources

For additional information refer to “Variable and parameter definition” on page 144, “Prompt definition” on page 149, and “Resource definition” on page 151.

7. Define jobs and job streams

For additional information refer to “Job” on page 535, and to “Job stream definition” on page 152.

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 82.

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 297 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 253 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 26
- “Workstation inter-process communication” on page 28
- “Tivoli Workload Scheduler network communication” on page 29

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 26. 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 299) and are stopped by unlink requests (see “unlink” on page 371) 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 121.

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 33. 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 297).

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 535.

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 fail, the job ends in the FAIL state.

1
1
1
1

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 24 shows the process tree on Tivoli Workload Scheduler workstations, other than standard agents, installed on UNIX:

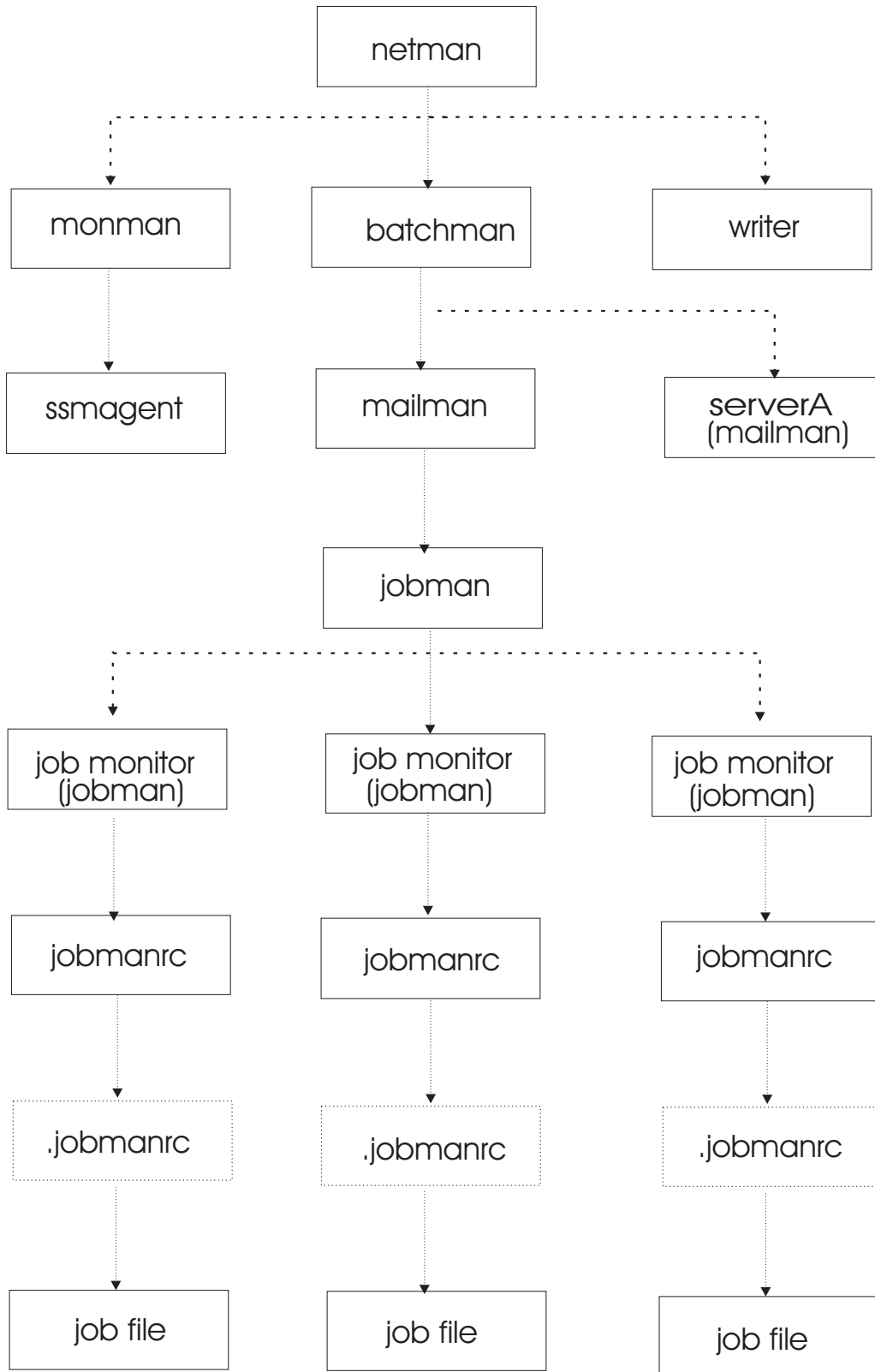


Figure 3. Process tree in UNIX

Figure 4 on page 25 shows the process tree on Tivoli Workload Scheduler workstations, other than standard agents, installed on Windows:

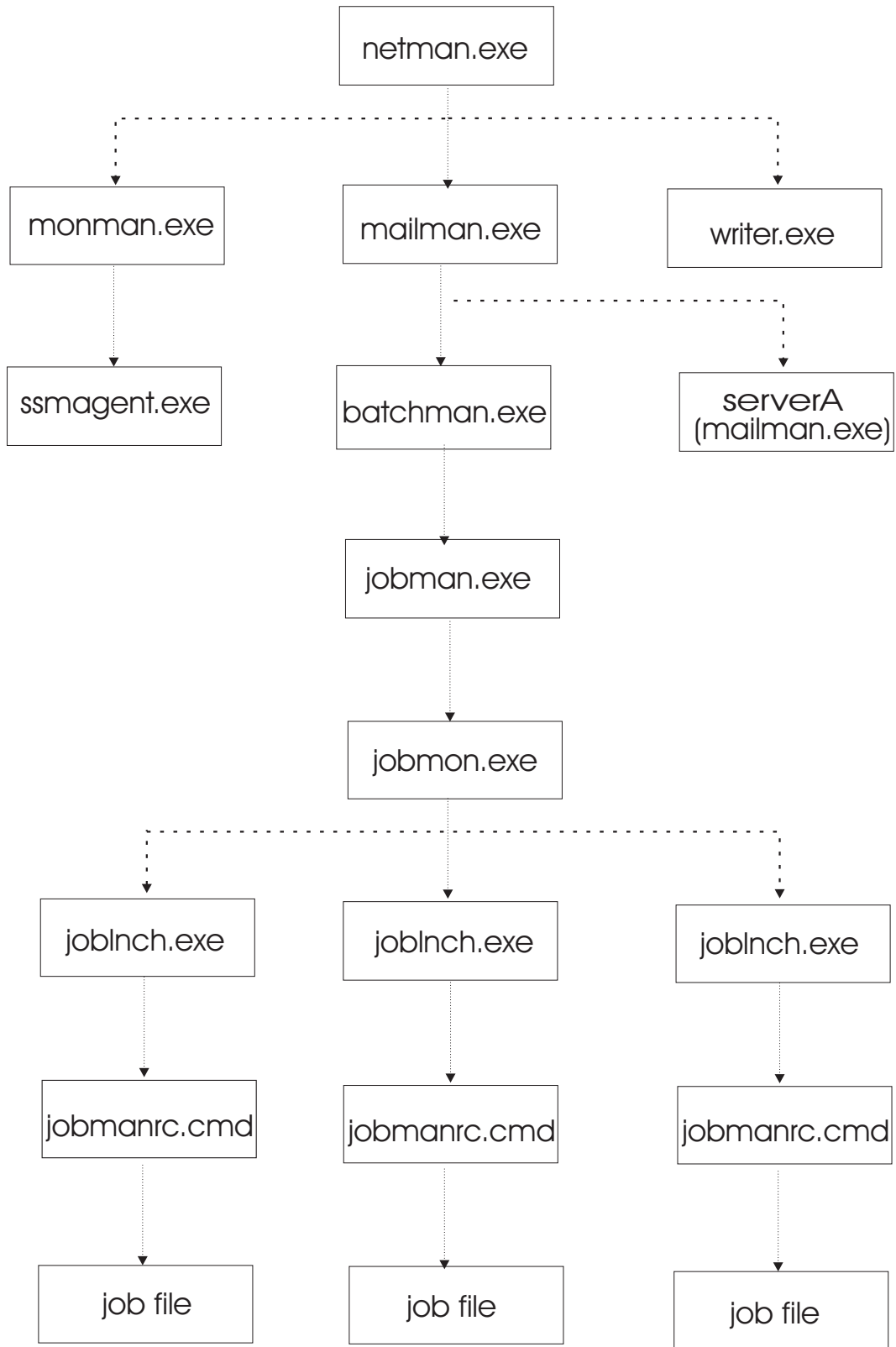


Figure 4. Process tree in Windows

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 .	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 Tivoli Workload Scheduler agent locally	./StartUpLwa.sh Note: can be run by <i>TWS_user</i> or root user only.	startuplwa Note: On Windows 2008 must be run as Administrator.
Stop the Tivoli Workload Scheduler agent locally	./ShutDownLwa.sh Note: can be run by <i>TWS_user</i> 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 405 for more information on the **StartUp** utility command.

Refer to “shutdown” on page 404 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 345 for more information on the **conman start** command.

Refer to “stop” on page 350 for more information on the **conman stop** command.

Refer to “shutdown” on page 345 for more information on the **conman shutdown** command.

Refer to “startappserver” on page 348 for more information on the **conman startappserver** command.

Refer to “stopappserver” on page 353 for more information on the **conman stopappserver** command.

Refer to “startmon” on page 349 for more information on the **conman startmon** command.

Refer to “stopmon” on page 355 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.

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 interfaces (Tivoli Dynamic Workload Console or Job Scheduling 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.

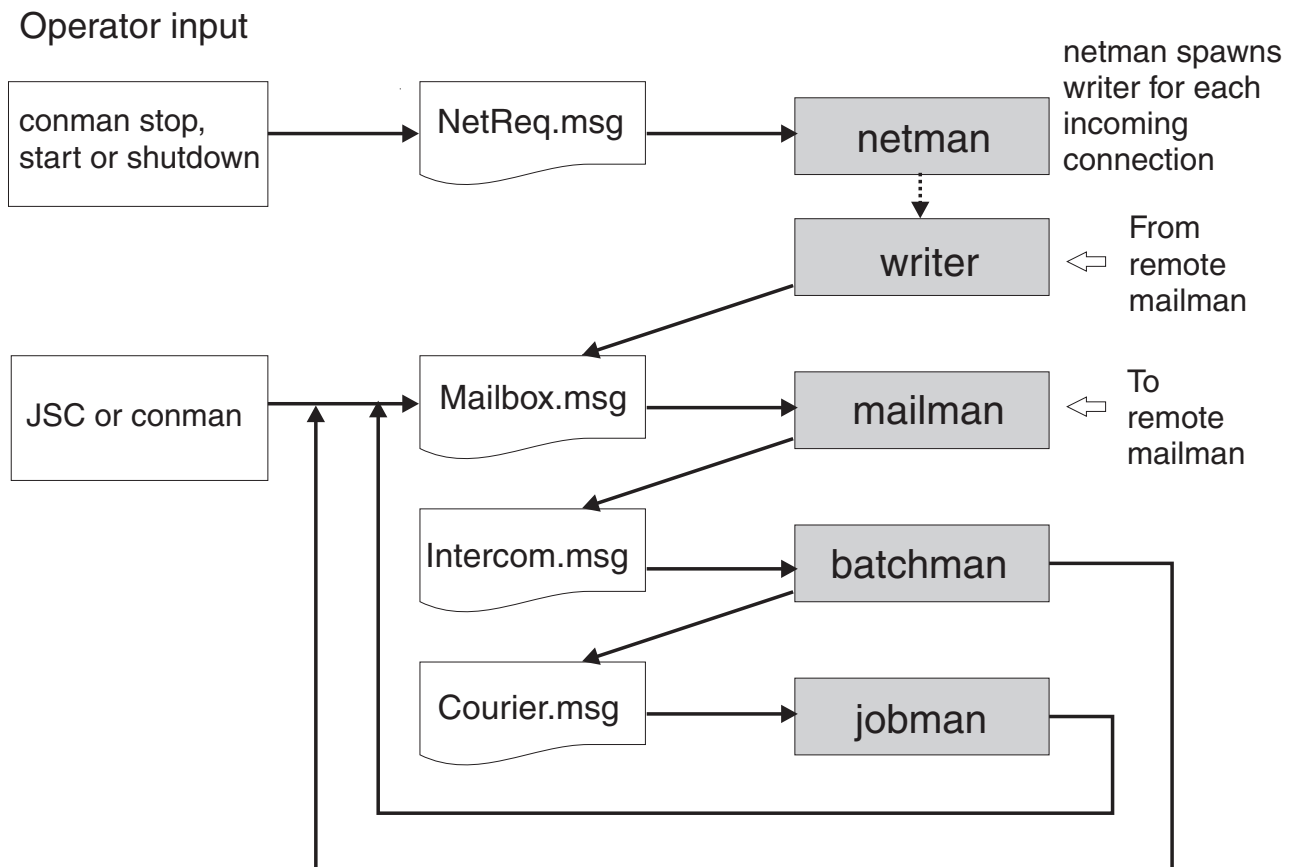


Figure 5. Inter-process communication

These files have a default maximum size of 10MB. The maximum size can be changed using the **evtsize** utility (see “evtsize” on page 388).

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 121 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 121, and by the “link” on page 299 and “unlink” on page 371 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”
- “Environment variables exported by jobman” on page 34
- “Customizing job processing on a UNIX workstation - jobmanrc” on page 37
- “Customizing job processing for a user on UNIX workstations - .jobmanrc” on page 39
- “Customizing job processing on a Windows workstation - jobmanrc.cmd” on page 41

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*.

Environment variables exported by jobman

The variables listed in Table 3 are set locally on the workstation and exported by **jobman** on Windows operating systems:

Table 3. Job environment variables for Windows

Variable Name	Value
COMPUTERNAME	The value of the <i>COMPUTERNAME</i> set in the user environment.
HOME	The path where the Tivoli Workload Scheduler instance was installed.
HOMEDRIVE	The value of the <i>HOMEDRIVE</i> set in the user environment.
HOMEPATH	The value of the <i>HOMEPATH</i> set in the user environment.
LANG	The value of the <i>LANG</i> 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.
SystemDrive	The value of the <i>SYSTEMDRIVE</i> set in the user environment.
SystemRoot	The value of the <i>SYSTEMROOT</i> set in the user environment.
TEMP	The value of the <i>TEMP</i> 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 <i>TMP</i> set in the user environment. If not specified, its value is set to c:\temp.
TMPDIR	The value of the <i>TMPDIR</i> 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 <i>UNISON_DIR</i> 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: workstation#sched_id.job.
UNISON_JOBNUM	The job number.
UNISON_MASTER	The name of the master workstation.

Table 3. Job environment variables for Windows (continued)

Variable Name	Value
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 <i>jobstreamID</i> of the job stream containing the job in process.
UNISON_SCHED_IA	The <i>StartTime</i> 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 <i>USERDOMAIN</i> set in the user environment.
USERNAME	The value of the <i>USERNAME</i> set in the user environment.
USERPROFILE	The value of the <i>USERPROFILE</i> set in the user environment.

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

Table 4. Job environment variables for UNIX

Variable Name	Value
HOME	The home directory of the user.
LANG	The value of the <i>LANG</i> set in the user environment.
LD_LIBRARY_PATH	The value of the <i>LD_LIBRARY_PATH</i> set in the user environment.
LD_RUN_PATH	The value of the <i>LD_RUN_PATH</i> set in the user environment.
LOGNAME	The login user name.
MAESTRO_OUTPUT_STYLE	The setting for output style for long object names.
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.
TWS_TISDIR	The value of the <i>TWS_TISDIR</i> set in the user environment.
TZ	The time zone set.

Table 4. Job environment variables for UNIX (continued)

Variable Name	Value
UNISON_CPU	The name of this workstation.
UNISON_DIR	The value of the <i>UNISON_DIR</i> set in the user environment.
UNISON_EXEC_PATH	The <i>.jobmanrc</i> 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: workstation#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 <i>jobstreamID</i> of the job stream containing the job in process.
UNISON_SCHED_IA	The <i>StartTime</i> 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.

1
1

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 5 describes the `jobmanrc` variables.

Table 5. 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 <code>\$UNISON_JCL</code> as the first argument. The user might be allowed or denied this option. See "Customizing job processing for a user on UNIX workstations - <code>.jobmanrc</code> " on page 39 for more information. If set to no , the presence of a local configuration script is ignored, and <code>\$UNISON_JCL</code> is run. Any other setting is interpreted as no .

Table 5. Variables defined by default in the jobmanrc file (continued)

Variable Name	Value
MAIL_ON_ABEND	<p>yes no</p> <p>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:</p> <pre>cpu#sched.job jcl-file failed with exit-code Please review standard-list-filename</pre> <p>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."</p>
SHELL_TYPE	<p>standard user script</p> <p>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.</p>
USE_EXEC	<p>yes no</p> <p>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 <i>MAIL_ON_ABEND</i> 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.</p>

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 5 on page 37).
- 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 5 on page 37).
- *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 5 on page 37), 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 6 describes the *jobmanrc.cmd* variables.

Table 6. Variables defined by default in the *jobmanrc.cmd* file

Variable Name	Value
HOME	The path to the <i>TWS_home</i> directory
POSIXHOME	The path to the <i>TWS_home</i> directory in a POSIX complaint format

Setting up options for using the user interfaces

To use the Tivoli Workload Scheduler user interfaces (the command-line programs, the Job Scheduling Console, and the Tivoli Dynamic Workload Console), and run the **logman** command and the **JnextPlan** script, you need to provide the following setup information to connect to the master domain manager via HTTP/HTTPS using the WebSphere Application Server infrastructure:

- The *hostname* of the master domain manager.
- The *port number* used when establishing the connection with the master domain manager.
- The credentials, *username* and *password*, of the *TWS_user*.
- The *proxy hostname* used in the connection with the HTTP protocol.
- The *proxy port number* used in the connection with the HTTP protocol.
- The *protocol* used during the communication. This can be HTTP with basic authentication, or HTTPS with certificate authentication.
- 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.

All this information, except *username* and *password*, is stored in the *TWS_home/localopts* properties file. For information about this configuration file, refer to *IBM Tivoli Workload Scheduler Administration Guide*.

The file containing the user credential is the **useropts** file. This file is first read when the user connects to a Tivoli Workload Scheduler command-line program. In addition to the *username* and *password* this file can also contain other settings which, if defined, overwrite those set in the **localopts** file. If the *username* and *password* are not specified in this file then they must be specified when invoking the command-line program.

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"` in **useropts**.

This is the sample content of a useropts file:

```
#
# Tivoli Workload Scheduler useropts file defines attributes of this Workstation.
#
#-----
# Attributes for CLI connections
USERNAME = MDMDBE4 # Username used in the connection
PASSWORD = "ENCRYPT:YEE7cEZs+HE+mEHCsdNOfg==" # Password used in the connection
#HOST = # Master hostname used when attempting a connection.
PROTOCOL = https # Protocol used to establish a connection with the Master.
#PROTOCOL = http # Protocol used to establish a connection with the Master.
PORT = 3111 # Protocol port
#PROXY =
#PROXYPORT =
TIMEOUT = 120 # Timeout in seconds to wait a server response
#DEFAULTWS =
```

The # symbol is used to comment a line. The user specified in the username field is defined in the security file on the master domain manager. The ENCRYPT label in the password field indicates that the specified setting has been encrypted; if this label does not appear, you must exit and access the interface program again to allow the encryption of that field.

Because Tivoli Workload Scheduler supports multiple product instances installed on the same machine, there can be more than one useropts file instances. 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 in the *user_home/.TWS* directory 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 *operator_home/.TWS/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 *operator_home/.TWS/useropts2* file containing the connection parameters settings that user operator needs to use to connect to that Tivoli Workload Scheduler instance.

The set of values described in this section is named *connection_parameters*. The *connection_parameters* are used when invoking either a command-line program, such as **composer**, **optman**, or **planman**, or when using the **logman** command or the **JnextPlan** scripts. If any of these parameters is omitted when invoking a command, Tivoli Workload Scheduler searches for the value first in the useropts file and then in the localopts file. If a parameter is not specified either in the connection parameters when invoking the command or inside the useropts and localopts files, an error is displayed.

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 60
- “Generating a production plan - JnextPlan” on page 63
- “Planman command line” on page 66
- “Starting production plan processing” on page 81
- “Automating production plan processing” on page 82

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 “Generating a production plan - JnextPlan” on page 63.

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 21.

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 windows 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 173, “at” on page 157 and “schedtime” on page 182.

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 322.

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 267.

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*:

Closest preceding

Using the closest earlier job or job stream instance. 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 6 on page 47 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 6. Closest preceding matching criteria

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

```

schedule Js2
on Th
at 0700
;job2
end

```

```

schedule Js1
on Fr
at 1000
follows wk1#Js2 previous
;job1
end

```

The job stream Js1 is 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 7 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.

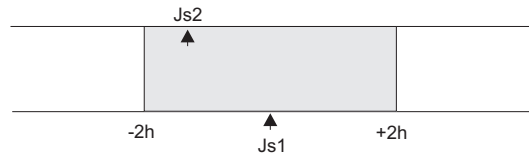


Figure 7. Within a relative interval matching criteria

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

```

schedule Js2
on everyday
at 0900
;job2

end

```

```

schedule Js1
on everyday
at 1000
follows wk1#Js2 relative from 0200 to
0200
;job1
end

```

The job stream Js1 is 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 8 on page 48 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.

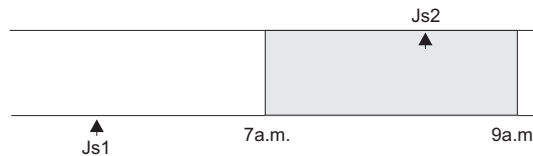


Figure 8. Within an absolute interval matching criteria

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

```

schedule Js1
on everyday
at 0800
;job2
end

```

```

schedule Js2
on everyday
at 1000
follows wk1#Js1 from 0700 to 0900
;job1
end

```

The job stream Js1 is 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.

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 9 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 9. Sameday matching criteria

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

```

schedule Js2
on everyday
at 0700
;job2
end

```

```

schedule Js1
on everyday
at 1000
follows wk1#Js2 sameday
;job1
end

```

The job stream Js1 is not launched until the job stream instance of Js2 on the workstation wk1 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 by 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.

- 1 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
- 1 they are specified in the *jobStreamName.workstationName.jobName* format.
- 1

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 on page 50 shows how a

pending predecessor and its successor are positioned in the preproduction plan.

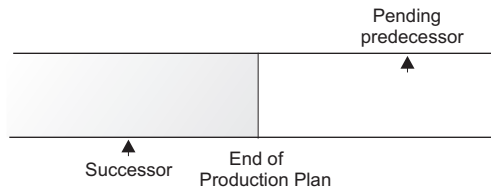


Figure 11. Pending predecessor instance

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 [0] 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.

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 12 shows the status of the two job streams in the plan on Thursdays and Fridays.

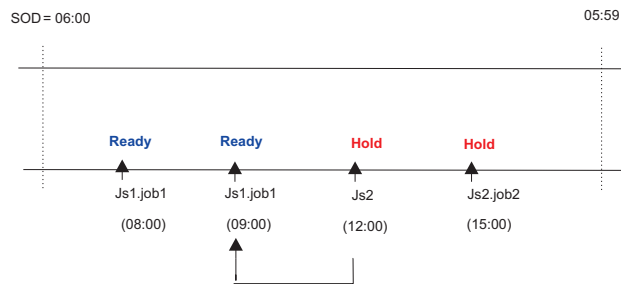


Figure 12. Closest preceding matching criteria - Step 1: before 08:00

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 13 shows the status of the two job streams in the plan on any other weekday except Thursdays and Fridays.

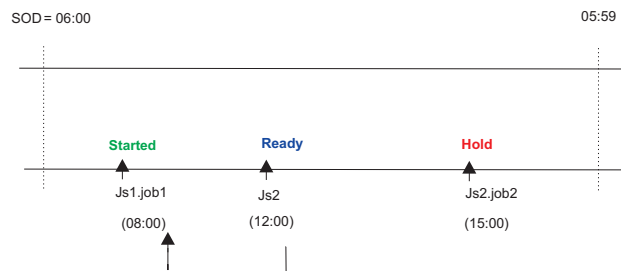


Figure 13. Closest preceding matching criteria - Step 2: at 08:00 on weekdays except Thursdays and Fridays

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 14 on page 52 shows the status of the two job streams in the plan.

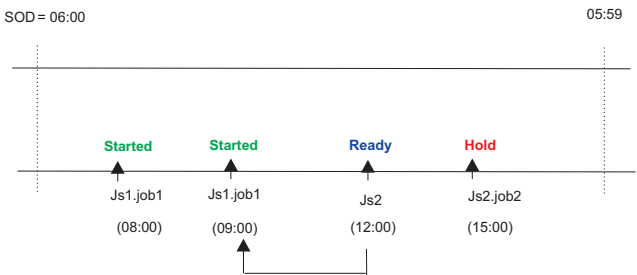


Figure 14. Closest preceding matching criteria - Step 3: at 09:00 on Thursdays and Fridays

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

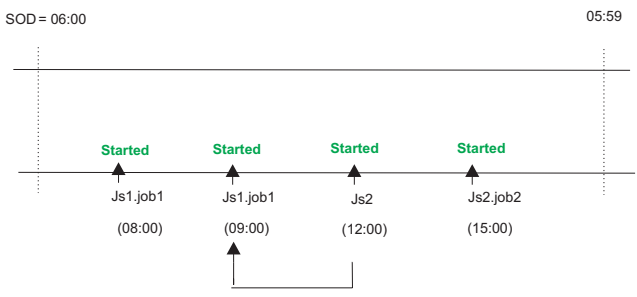


Figure 15. Closest preceding matching criteria - Step 4: at 15:00 on every day

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

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 16 shows the status of the two job streams in the plan at 06:00 (SOD) on Thursday:

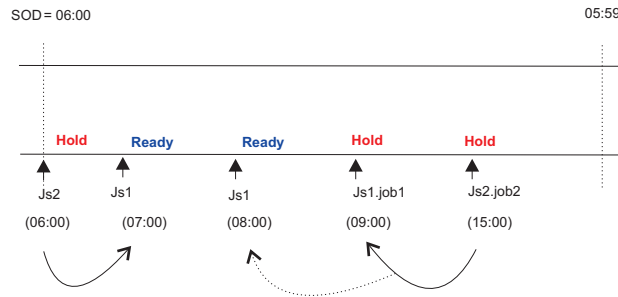


Figure 16. Sameday matching criteria - Step 1: at Start of Day (SOD) on a Thursday

2. At 09:00, Js1.job1 starts and Js1 changes status. Js2.job2 is held until its scheduled time. Figure 17 shows the status of the job streams in the plan at 09:00.

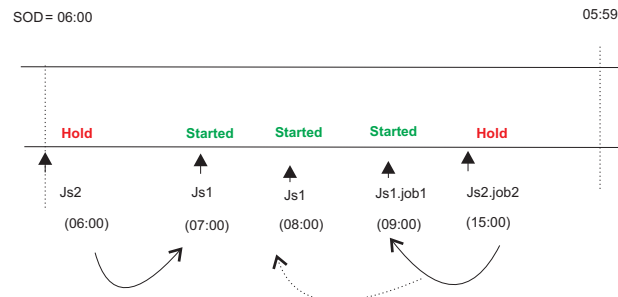


Figure 17. Sameday matching criteria - Step 2: at 9:00

3. On Thursdays at 15:00, Js2 changes to ready status and Js2.job2 starts. Figure 18 shows the status of the two job streams in the plan at 15:00.

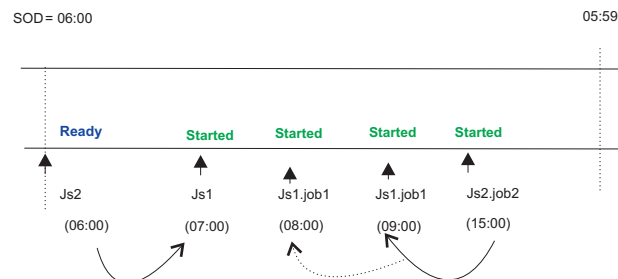


Figure 18. Sameday matching criteria - Step 3: at 15:00

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)
              JOB2      HOLD  10(13:00)
              JS1(1500 11/13/09).@
```

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

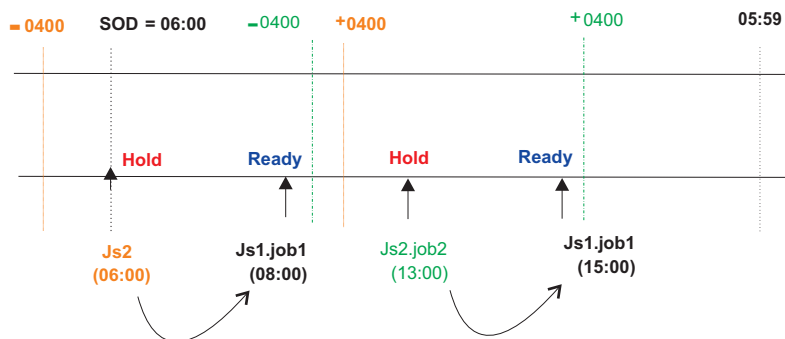


Figure 19. Relative Interval matching criteria - at start of day on Thursday

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 on page 56 shows the status of the job streams in the plan at start of day on Thursday.

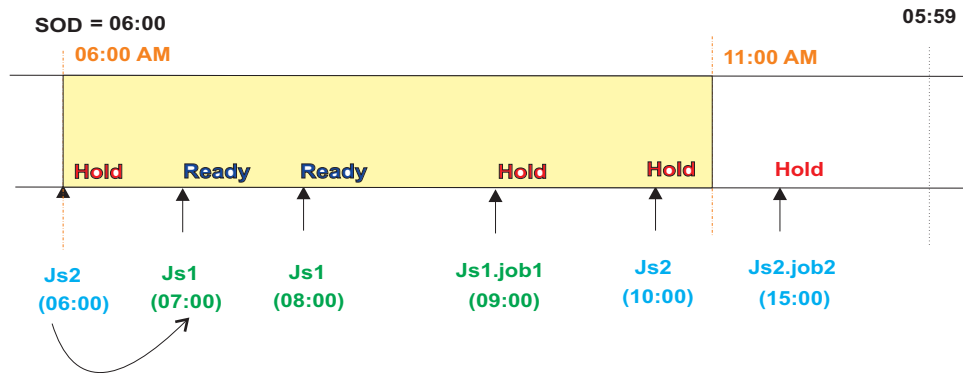


Figure 20. Absolute interval matching criteria - at start of day on Thursday

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 “Generating a production plan - JnextPlan” on page 63.

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 158.
- The **enCarryForward** global option. See *IBM Tivoli Workload Scheduler Administration Guide*.
- The **stageman -carryforward** command-line keyword. See “The stageman command” on page 76.
- The **carryStates** global option. See *IBM Tivoli Workload Scheduler Administration Guide*.

Table 7 shows how the carry forward global options work together.

Table 7. 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.
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 8 shows the result of the carry forward setting based on how the **enCarryForward** global option and the **stageman -carryforward** keywords are set.

Table 8. 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
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 60.

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 66.

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.

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 66.

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 0600.

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. 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 322. 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 445 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 446 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*.

Generating a production plan - JnextPlan

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 21.
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
```

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 *hhmm*, where *hh* 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 6:00 a.m. and that the date format set in the *localopts* file is *mm/dd/yyyy*, if the values set are **-from** 07/05/2005 and **-to** 07/07/2005 then the plan is created to span the time frame from 07/05/2005 at 6:00 to 07/07/2005 at 5:59 and not to 07/08/2005 at 5:59.

Comments

The **JnextPlan** script is run using the default connection parameters defined in either *localopts* or *useropts* files. 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 it is described in “Planman command line” on page 66.

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, Windows 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.
2. Prints preproduction reports.

SwitchPlan

This script invokes internally the **stageman** command. For more information refer to “The stageman command” on page 76. **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 being 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 76. **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

Represents the set of parameters that rule the interaction between the product interface, **planman** 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 with HTTP protocol.

proxy_port_number

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

user_password

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

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 **planman**. Make sure this username is specified in the security file on the master domain manager with the following authorization:

file name=prodsked,Symphony access=build

To perform commands against the production plan.

file name=trialsked access=build

To perform commands against the trial and forecast plans.

file name=@ access=build

To perform commands against all types of plan.

If any of these parameters is omitted when invoking **planman**, Tivoli Workload Scheduler searches for that 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 41 for information on `useropts` and `localopts` files.

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 68
- "Creating an intermediate plan for a plan extension" on page 69
- "Retrieving the production plan information" on page 70
- "Creating a trial plan" on page 71
- "Creating a trial plan of a production plan extension" on page 72

- “Creating a forecast plan” on page 73
- “Unlocking the production plan” on page 75
- “Resetting the production plan” on page 75

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

“Deploying rules” on page 74.

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 66.

-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 *hhmm*, where *hh* 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 03/21/2005 at 23:07 to 03/22/2005 at 23:06 in the local time zone:

```
planman crt -from 03/21/05 2307
```
2. This command creates the production plan from 03/21/2005 at 09:00 to 03/21/2005 at 15:00:

```
planman crt -from 03/21/2005 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 03/21/2005 at 6:00 to 03/25/2005 at 5:59:

```
planman crt -to 03/25/2005
```
4. This command creates a production plan from 03/21/2005 at 18:05 to 03/24/2005 at 23:00 in the time zone of Europe\Paris:

```
planman crt -from 03/21/2005 1805 tz Europe\Rome  
-to 03/24/2005 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 66.

- 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 *hhmm*, where *hh* 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 66.

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:

```
C:\TWS\maestro830>bin\planman showinfo
TWS for WINDOWS/PLANMAN 8.3 (1.0)
Licensed Materials Property of IBM
5698-WKB
(C) Copyright IBM Corp 1998, 2006
US Government User Restricted Rights
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.
Installed for user 'STEFAN003\maestro830'.
Locale LANG set to the following: "en"
Plan Creation Start Time: 01/03/2006 06:00 TZ CST
Production Plan End time: 01/05/2006 05:59 TZ CST
Production Plan Time Extention: 048:00
Plan Last Update: 01/03/2006 12:24 TZ CST
Pre-Production Plan End Time: 01/19/2006 06:00 TZ CST
Start Time of first incomplete JobStream Instance: 01/03/2006 23:00 TZ CST
Run Number: 5
Confirm Run Number: 5

C:\TWS\maestro830>
```

Creating a trial plan

The following syntax is used to create a trial plan:

```
planman [connection_parameters] crtrial 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 66.

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 **Tfilename**. 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 *hhmm*, where *hh* 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 66.

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 *Tfilename*. 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 *hhmm*, where *hh* 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 66.

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 *Efilename*. 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 *hhmm*, where *hh* 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 66.

-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.

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 is 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 66.

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 66.

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 **dbunstats** before the **JnextPlan** script. See the *Administration Guide* for details on **dbunstats**.

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 56 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 302 and “setsym” on page 313. If neither **-log** nor **-nolog** keywords are specified, Tivoli Workload Scheduler archives the old production plan using the following naming convention:

Myyyymmddhhtt

where *yyyyymmddhhtt* 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 267.

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

[*connection_parameters*]
[-**prod** *symphony-file*]
[-**minmax** *setting*]
[-**smooth** *weighting*]

Arguments

-U Displays command usage information and exits.

-V Displays the command version and exits.

connection_parameters

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*] [-**proxypport** *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 41 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 *setting*

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 successfully, 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 “Generating a production plan - *JnextPlan*” on page 63 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 **tw_s_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 86
- “Planning critical jobs” on page 89
- “Processing and monitoring critical jobs” on page 91
- “Workload service assurance scenario” on page 93

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 9 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 9. Workload service assurance global options

Option	Description
<code>enWorkloadServiceAssurance</code> <code>wa</code>	Enables or disables privileged processing of mission-critical jobs and their predecessors. The default value is YES. Specify NO to disable.
<code>promotionOffset</code> <code>po</code>	<p>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.</p> <p>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 <code>promotionoffset</code> 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.</p>

Table 9. Workload service assurance global options (continued)

Option	Description
<p>longDurationThreshold 1d</p>	<p>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.</p> <p>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.</p> <p>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.</p>
<p>approachingLateOffset a1</p>	<p>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.</p> <p>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 Tivoli Dynamic Workload Console. The default is 120 seconds.</p> <p>Note: The value of this parameter is checked regularly. JnextPlan does not need to be run for changes to take effect.</p>

Table 9. Workload service assurance global options (continued)

Option	Description
deadlineOffset do	<p>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.</p> <p>The <code>deadlineOffset</code> 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.</p> <p>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.</p>

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 10 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 10. Workload service assurance local options

Option	Description
jm promoted nice	<p>Sets the <code>nice</code> 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.</p> <p>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.</p> <p>The <code>jm nice</code> 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 <code>jm promoted nice</code> is set to -4 and <code>jm nice</code> to -2, the critical job submitted by user root would have a priority of -6.</p>

Table 10. Workload service assurance local options (continued)

Option	Description
jm promoted priority	<p>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.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> • High • AboveNormal • Normal • BelowNormal • Low or Idle <p>The default is AboveNormal.</p> <p>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 <code>jm promoted nice</code> sets it back to the default.</p>

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 Tivoli 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** key word to 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.

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 Tivoli 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 92 shows the critical path through a critical network at a specific moment during the processing of the plan.

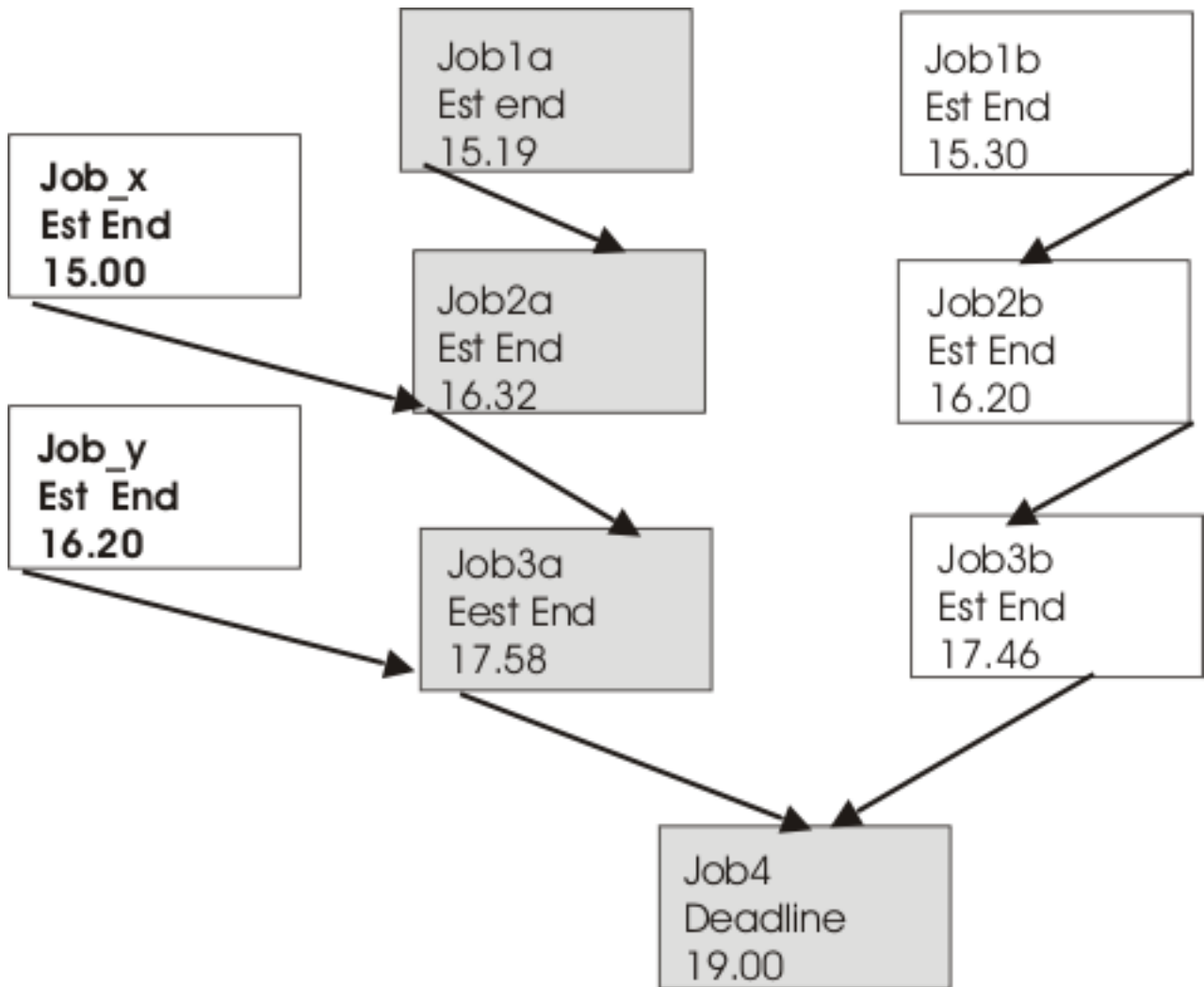


Figure 21. Critical path

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 Tivoli 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

When you upgrade from versions prior to 8.5, the global parameter definitions, from now on 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 as before the migration since any Tivoli 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
```

See the *Tivoli Workload Scheduler Planning and Installation* manual for details about the upgrade process.

When upgrading from versions 8.3 or later, you 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 on both the version 8.3 or 8.4 and the version 8.5 masters.

Local parameters that were created and managed with the **parms** utility command in the local parameter database on the workstations, work as before.

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 11. The relationship between variable tables and their enclosed variables in the Tivoli Workload Scheduler security file

Defined access to variable	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.”
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\twaprofile\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.”
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 prompts 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 e-mail, 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 e-mail, 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.

Table 12 lists some simple scenarios involving the use of event rules. The corresponding XML coding is shown in Table 16 on page 109.

Table 12. Simple event rule scenarios

<p>Scenario 1: Send e-mail notification</p>	<ol style="list-style-type: none"> 1. The administrator defines the following event rule: <ul style="list-style-type: none"> • When any of the job123@ jobs terminates in error, send an e-mail to operator john.smith@mycorp.com. The subject of the e-mail includes the names of the job instance and of the associated workstation. <p>The event rule is valid from December 1st to December 31st in the 12:00-16:00 EST time window.</p> 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 e-mail so that he can check the job and take corrective action.
<p>Scenario 2: Monitor that workstation links back</p>	<ol style="list-style-type: none"> 1. The administrator defines the following event rule: <ul style="list-style-type: none"> • If workstation CPU1 becomes unlinked and does not link back within 10 minutes, send a notification e-mail 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. <p>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 e-mail to Chuck Derry.</p> 4. Chuck Derry receives the e-mail, 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.

Table 12. Simple event rule scenarios (continued)

<p>Scenario 3: Submit job stream when FTP has completed</p>	<ol style="list-style-type: none"> 1. The administrator defines the following event rule: <ul style="list-style-type: none"> • When file daytransac* is created in the SFoperation directory in workstation system1, and modifications to the file have terminated, submit the calmonthlyrev job stream. <p>The event rule is valid year-round in the 18:00-22:00 EST time window.</p> 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.
<p>Scenario 4: Start long duration jobs based on timeout</p>	<ol style="list-style-type: none"> 1. The administrator defines the following event rule: <ul style="list-style-type: none"> • 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 e-mail 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 e-mail 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. <p>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 e-mail, otherwise it replies Yes to the prompt and submits jobstream-z.</p>
<p>Scenario 5: Integration with SAP R/3 (in combination with Tivoli Workload Scheduler for Applications)</p>	<ol style="list-style-type: none"> 1. The administrator defines the following event rule: <ul style="list-style-type: none"> • When an event called ID3965 is generated on SAP R/3 server Billing, Tivoli Workload Scheduler must: <ol style="list-style-type: none"> a. Run the command: <pre>"/usr/apps/helpDesk -openTicket -text 'Processing error \$parameter on SAP system \$wsname'"</pre> <p>to open a service desk ticket</p> 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. <p>When the ID3965 event is detected, Tivoli Workload Scheduler runs the specified help desk command and sends an event to TEC.</p> 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.

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.

Table 13. *conman* commands for managing monitoring engines (continued)

Command	Purpose
stopmon	Stops monman on an agent. Can be issued from a different agent.

1

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.

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 summarizes the ones you need:

Table 15. Interfaces and commands for managing event-driven workload automation

Interface or command	Use to...
optman	<p>Change the default values of global options associated with event management. Global options are used to configure:</p> <ul style="list-style-type: none"> • 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). • 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 e-mails 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. <p>See the <i>Administration Guide</i> for a list of global options.</p>
composer	<p>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.</p> <p>Query the Tivoli Workload Scheduler relational database for:</p> <ul style="list-style-type: none"> • event rule definitions filtered by: <ul style="list-style-type: none"> – rule, event, and action properties – jobs and job streams involved with the rule action • event rule instances, actions run, and message log records <p>See “Event rule definition” on page 188 to learn how to define event rules. See Chapter 9, “Managing objects in the database - composer,” on page 199 for command reference.</p>

Table 15. Interfaces and commands for managing event-driven workload automation (continued)

Interface or command	Use to...
Tivoli Dynamic Workload Console	<p>Have a graphical user interface to:</p> <ul style="list-style-type: none"> • Model and manage event rule definitions (browse, create, delete, modify, query, unlock) • Query the Tivoli Workload Scheduler relational database for: <ul style="list-style-type: none"> - event rule definitions filtered by: <ul style="list-style-type: none"> - 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 104 and Table 14 on page 105 <p>See the Tivoli Dynamic Workload Console online documentation.</p>
conman	<p>Manage the monitoring devices, namely the event processing server and monitoring engines, as described in tables Table 13 on page 104 and Table 14 on page 105.</p> <p>See Chapter 10, "Managing objects in the plan - conman," on page 253 for command reference.</p>
utility commands	<p>Create custom event definitions and manually send custom events to the event processing server. See "evtdef" on page 385 and "sendevent" on page 402 for details on these commands.</p>
planman	<p>Manually deploy new and changed rules.</p> <p>See "Deploying rules" on page 74 for details.</p>
Security file	<p>Set security authorizations to manage event rules, events, actions, and their instances.</p> <p>See the <i>Tivoli Workload Scheduler Administration Guide</i> for reference about configuring the Tivoli Workload Scheduler security file.</p>

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 Tivoli 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 188, while the explanation of how you use the Tivoli 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.

You can deploy and undeploy rules as you need by setting the `isDraft` attribute to no or to yes with `composer` or with the Tivoli 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 e-mail.

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 e-mail subject and body matters.

Event rule examples

The following tables describe the event rule definitions that apply to the scenarios described in table Table 12 on page 102.

Table 16. Event rule definition for scenario1

When any of the <code>job123@</code> jobs terminates in error, send an e-mail to operator <code>john.smith@mycorp.com</code> . The subject of the e-mail 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.
--

Table 16. Event rule definition for scenario1 (continued)

```

<?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="JobName" operator="eq">
  <value>job123@</value>
  </attributeFilter>
  <attributeFilter name="Status" operator="eq">
  <value>Error</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>

```

Table 17. Event rule definition for scenario2

If workstation CPU1 becomes unlinked and does not link back within 10 minutes, send a notification e-mail to chuck.derry@mycorp.com.

Table 17. Event rule definition for scenario2 (continued)

```

<?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="10" unit="minutes" />
  <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="onTimeOut">
    <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>

```

Table 18. Event rule definition for scenario3

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.

Table 18. Event rule definition for scenario3 (continued)

```
<?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>
```

Table 19. Event rule definition for scenario4

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 e-mail to twsoper@mycompany.com alerting that the job is late.

Save as draft.

Table 19. Event rule definition for scenario4 (continued)

```

<?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 scenario2</description>
  <timeZone>America/Buenos_Aires</timeZone>
  <timeInterval amount="5" unit="minutes" />
    <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="ReplyPrompt" 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>

```

Rule operation notes

The following table contains critical information on event rule behavior that might help you understand the reason for unexpected results:

Table 20. Rule operation notes

Topic	Note
Notes on rule status	<ul style="list-style-type: none"> • Depending on its from and to validity dates, the status of any rule changes as follows upon deployment: <ul style="list-style-type: none"> – 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	<p>If the event processor is stopped or crashes, the status of rule instances that are only partially matched is lost.</p>

Table 20. Rule operation notes (continued)

Topic	Note
Repeating set and sequence rules	<p>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:</p> <ol style="list-style-type: none"> 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. <p>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.</p> <p>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.</p>

Table 20. Rule operation notes (continued)

Topic	Note
Set and sequence rule types with shorter than 24 hours activity time windows	<p>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.</p> <p>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.</p> <p>If on January 1 the first event is received at 8 AM, the rule waits for the second event to take place, and stays <i>alive</i> beyond 10 AM if the event is not detected.</p> <p>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 <i>alive</i>. If the second event is received again at 7 AM of January 2, the rule is triggered and the actions are implemented.</p> <p>If you don not want the rule to <i>carry over</i> to the next day, you should purge it.</p>

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 495, 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 Tivoli 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

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 233 and “unlock” on page 248.

You can use short and long keywords when issuing commands from **composer**, as Table 21 on page 120 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 21. List of supported scheduling object keywords

Long keywords	Short keywords	backward 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 22 when defining jobs and job streams:

Table 22. List of reserved words when defining jobs and job streams

abendprompt	after	as	at	autodocoff
autocon	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	rcondsucc	recovery	relative	request
rerun	runcycle	sa	sameday	schedtime
schedule	scriptname	stop	streamlogon	su
tasktype	timezone	to	token_in	until
validfrom	validto	variable	vt	weekday(s)
workday(s)				

Avoid using the keywords listed in Table 23 on page 121 when defining workstations, workstation classes, and domains:

Table 23. 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	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 24 when defining Windows users:

Table 24. 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]]
    [type fta | s-agent | x-agent | manager | broker ]
    [ignore]
    [autolink on | off]
    [behindfirewall on | off]
    [securitylevel enabled | on | force]

```

```
[fullstatus on | off]
[server serverid]
end
```

```
[cpuname ...]
```

```
[cpuclass ...]
```

```
[domain ...]
```

Arguments

Table 25 describes the values you set for each attribute for management workstation types. Following the tables you find additional details about each attribute.

Table 25. 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.		
variable	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		
node	The system host name or IP address.		
tcpaddr	The value assigned to <i>nm port number</i> in the <i>localopts</i> file. For multiple workstations on a system, enter an unused port number. The default value is 31111.		
secureaddr	The value assigned to <i>nm SSL port</i> in the <i>localopts</i> file. Specify it if securitylevel is set to <i>on</i> , <i>force</i> or <i>enabled</i> .		
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.		
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.		

Table 25. Attribute settings for management workstation types (continued)

Attributes	Master Domain Manager	Domain Manager	Backup Domain Manager
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.

Table 26 describes the values you set for each attribute for target workstation types. Following the table you find additional details about each attribute.

Table 26. Attribute settings for target workstation types

Attribute	Fault-tolerant Agent and Standard Agent	Workload Broker Workstation	Extended Agent
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 Specify OTHER for i5/OS systems running as limited fault-tolerant agents.	OTHER	The operating system installed on the machine. Specify one of the following values: UNIX, WNT, OTHER
node	The system host name or IP address.		The system host name or IP address. Specify NULL when: <ul style="list-style-type: none"> defining an extended agent for PeopleSoft, SAP or Oracle. host is set to \$MASTER
tcpaddr	The value assigned to <i>nm port number</i> in the <i>localopts</i> file. When defining multiple workstations on a system, enter an unused port number. The default value is 31111.		See the selected access method specifications.
secureaddr	The value assigned to <i>nm SSL port</i> in the <i>localopts</i> file. Specify it if securitylevel is set to on, force or enabled.	This setting is ignored.	Not Applicable

Table 26. Attribute settings for target workstation types (continued)

Attribute	Fault-tolerant Agent and Standard Agent	Workload Broker Workstation	Extended Agent
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.
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 ignored.
host	Not Applicable	Not Applicable	The host workstation. It can be set to \$MASTER.
access	Not Applicable		Select the appropriate access method file name.
type	fta s-agent The default value is fta.	broker	x-agent
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
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
securitylevel	The type of SSL authentication to use: enabled on force Not applicable for i5/OS systems running as limited fault-tolerant agents.	Not Applicable	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	OFF
server	0-9, A-Z. When specified, it requires the creation of a dedicated mailman processes on the parent workstation.		Not Applicable

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 22 on page 120

description *"description"*

Provides a description of the workstation. The text must be enclosed within double quotes.

variable *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.

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 i5/OS systems running as limited fault-tolerant agents. Optional value for other types of workstations.

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

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 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 other types of workstations

Specify the value assigned in the `localopts` file for variable `nm SSL port`. The value must be different 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.

See Chapter 13, “Managing time zones,” on page 445 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 specifies:

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 you defined a backup master domain manager and you want to ensure that the extended agent is not isolated if the hosting workstation becomes unavailable, you can specifying **\$MASTER** in this field to indicate that the host workstation for the extended agent is the master domain manager. In this case make sure that the extended agent methods are installed also on the backup master domain manager. You can enable and disable the automatic resolution of the **\$MASTER** key using the `nm 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.

type Specifies the type of the workstation. 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.

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* manual.

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 managed domain, if different from **MASTERDM** domain.

You specify that a workstation is a manager also in the **manager** field of the "Domain definition" on page 133. 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 manages the lifecycle of Tivoli Workload Scheduler workload broker type jobs and routes the information related to the dynamic workload life cycle to the master domain manager.

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 workstation 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 for fault-tolerant agent workstation. For domain managers this keyword is automatically set to **on**. Specify:

- on** If you want that the fault-tolerant agent workstation operates in **Full Status** mode, this means that the workstation is updated with the status of jobs and job streams running on all other workstations in its domain and in subordinate domains, but not on peer or parent domains. In this case, 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 that the fault-tolerant agent workstation is informed only about the status of jobs and job streams on other workstations that affect its own jobs and job streams. This can improve performance by reducing network activity.

This setting automatically sets accordingly the resolve dependencies option.

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 specified, 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 27. 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
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 a fault-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.

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 shut down 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
end

```

The following example creates a broker workstation.

```

cpuname ITDWBAGENT
  os OTHER
  node itdwbtst11.ibm.com TCPADDR 31111
  timezone Europe/Rome
  domain MASTERDM
  for MAESTRO
    type BROKER
    autolink OFF
    behindfirewall OFF
    fullstatus OFF
  end
end

```

See also

To create a workstation definition in the Tivoli 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. 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 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 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**→**Workstation Class**
4. Specify your choices in the Properties - Workstation Class panel.

For the equivalent Job Scheduling Console task, see "Creating a Workstation Class" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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
```

[**cpu***name* ...]

[**cpu***class* ...]

[**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 121.

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 Tivoli 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.

For the equivalent Job Scheduling Console task, see "Creating a Domain" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 | docommand "command"}
    streamlogon username
    [description "description"]
    [tasktype tasktype]
    [interactive]
    [rcondsucc "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 152 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 *defaulttws* when starting the **composer** session.

For more information on how to start a composer session refer to “Running the composer program” on page 201. 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 140 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=tws]JobName]
```

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 (").

1 The job fails if the script specified in the **scriptname** option is not found
 1 or does not have execute permission. It abends if the script that is not
 1 found or does not have execute permission includes parameters.

docommand *command*

1 Specifies a command that the job runs. Enter a valid command and any
 1 options and arguments enclosed in double quotes ("). The length of
 1 *command* plus the length of *Success Condition* (of the **rccondsucc** keyword)
 1 must not exceed 4095 characters. You can also enter Tivoli Workload
 1 Scheduler parameters.

1 The job abends if the file specified with the **docommand** option is not
 1 found or does not have execute permission.

1 See "Using variables and parameters in job definitions" on page 140 for
 1 more information.

streamlogon *username*

The user name under which the job runs. 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 140 for more information.

For Windows jobs, the user must also have a user definition.

See "Windows user definition" on page 141 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.

1 The job fails if the user specified in the **streamologon** option does not
 1 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. Refer to *IBM Tivoli Workload Scheduler for Applications: User's Guide* for information about customized task types for supported vendor acquired applications.

OTHER

For jobs that run on extended agents.

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 contain a combination of comparison and Boolean expressions:

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 28. 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 29. 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.

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 140 for more information.

Table 30 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 30. 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 <i>yes</i> , 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 <i>yes</i> , run job2.	Issue recovery prompt. If reply is <i>yes</i> , 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 <i>yes</i> , repeat above. If job1 is successful, run job2.
Recovery prompt: Yes Recovery job: Yes	Issue recovery prompt. If reply is <i>yes</i> , run jobr. If it ends abnormally, intervention is required. If it is successful, run job2.	Issue recovery prompt. If reply is <i>yes</i> , run jobr. Run job2.	Issue recovery prompt. If reply is <i>yes</i> , 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#g11
    scriptname "/usr/acct/scripts/g11"
    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
```

See also

To create a job 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**→**Job Definition**
4. Click on a type of job definition and specify your choices in the ensuing properties panel.

For the equivalent Job Scheduling Console task, see "Creating a Job Definition" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 144 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 144.

Windows 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 system. 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 a Windows user.

workstation

Specifies the workstation on which the user is allowed to launch jobs. The pound sign is required. The default is blank, meaning all workstations.

[domain\]username

Specifies the Windows domain of the user and the name of the user.

Note: Windows user names are case-sensitive. Also, the user must be able to log on to the workstation on which Tivoli Workload Scheduler launches jobs, and have the permission to **Log on as batch**.

The domain name can contain up to 16 characters (including the backslash), and the user name can contain up to 31 characters.

If the name is not unique in Windows, 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 29 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.

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 Windows 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**→**Windows User**
4. Specify your choices in the Properties - Windows User panel.

For the equivalent Job Scheduling Console task, see "Creating a Windows User" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

Using the Tivoli Workload Scheduler user and streamlogon definitions

In Windows, 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` 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.

When you define or submit 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**.

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. 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.

For the equivalent Job Scheduling Console task, see "Creating a Calendar" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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. Do not include the names of other variables.

However, the recommended way to define variables is to use a “Variable table definition” on page 147. 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 98.

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 31. How to handle a backslash in variable substitution

Wrong way	Right way
<pre>1. Define the MYDIR variable as: \$PARM MYDIR "scripts" 2. Use it in this way: job01 scriptname "c:\operid\^MYDIR\test.cmd"</pre>	<pre>1. Define the MYDIR variable as: \$PARM MYDIR "\scripts" 2. Use it in this way: job01 scriptname "c:\operid^MYDIR^test.cmd"</pre>

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 398 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 32 summarizes in which submit command keyword you can use parameters.

Table 32. 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 253.

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^"
```

See also

For the equivalent Job Scheduling Console task, see "Creating a Parameter" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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

```
variable tablename
[description "description"]
[isdefault]
members
[variablename "variablevalue"]
...
[variablename "variablevalue"]
end
```

Arguments

variable *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. Values must be enclosed within double quotes.

Example

The following example shows a variable table and its contents.

```
VARIABLE TEST1
MEMBERS
DEVBATC "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 with 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 33. Required access keyword on variable table in the security file (variable object) and allowed actions on its variable entries.

Required security file access keyword on enclosing variable table	Allowed action on listed variable entries
Modify	Add
	Delete
	Modify
	Rename
Display	Display
Unlock	Unlock

See also

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 prompts 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 535 and “Job stream definition” on page 152.

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 eight alphanumeric characters, including dashes (-) and underscores (_), and must start with a letter.

text

Provides the text of the prompt. 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 144 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.

For the equivalent Job Scheduling Console task, see "Creating a Prompt" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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. 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.

For the equivalent Job Scheduling Console task, see "Creating a Distributed Resource" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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]
  [variable table_name]
  [freedays calendarname [-sa] [-su]]
  [on [runcycle name]
    [validfrom date] [validto date]
    [description "text"]
    [variable 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

```

Arguments

Table 34 contains a brief description of the job stream definition keywords. A detailed description of each scheduling keyword is provided in the next subsections.

Table 34. 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 157
carryforward	Carries the job stream forward if it is not completed.	"carryforward" on page 158

Table 34. List of scheduling keywords (continued)

Keyword	Description	Page
<i>comment</i>	Includes comments in the definition of a job stream or in a job contained in the job stream.	"comment" on page 158
confirmed	Specifies that the completion of this job requires confirmation.	"confirmed" on page 159
critical	Specifies that the job is mission critical and must therefore be managed preferentially.	"critical" on page 159
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 160
description	Contains a description of the job stream. The maximum length of this field is 120 characters.	"description" on page 161
draft	Specifies that the plan generation process must ignore this job stream.	"draft" on page 161
end	Marks the end of a job stream.	"end" on page 161
every	Launches the job repeatedly at a specified rate.	"every" on page 162
except	Specifies dates that are exceptions to the on dates the job stream is selected to run.	"except" on page 164
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 164
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 165
freedays	Specifies a freeday calendar for calculating <i>workdays</i> for the job stream. It can also set Saturdays and Sundays as <i>workdays</i> .	"freedays" on page 167
job statement	Defines a job and its dependencies.	"job statement" on page 169
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 170
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 170
limit	Sets a limit on the number of jobs that can be launched concurrently from the job stream.	"limit" on page 171
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 171
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 172

Table 34. List of scheduling keywords (continued)

Keyword	Description	Page
on	Defines the dates on which the job stream is selected to run.	"on" on page 173
opens	Defines files that must be accessible before the job or job stream is launched.	"opens" on page 178
onuntil	Specifies the action to take on a job or job stream whose until time has been reached.	"until" on page 185
priority	Defines the priority for a job or job stream.	"priority" on page 180
prompt	Defines prompts that must be replied to before the job or job stream is launched.	"prompt" on page 181
runcycle	Specifies a label with a friendly name for the run cycle	<ul style="list-style-type: none"> • "except" on page 164 • "on" on page 173
schedule	Assigns a name to the job stream.	"schedule" on page 183
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 182
timezone tz	Specifies the time zone to be used when computing the start time.	"timezone" on page 184
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 185
validfrom	Defines the date from which the job stream instance starts.	"validfrom" on page 187
validto	Indicates the date on which the job stream instance ends.	"validfrom" on page 187
vartable	Defines the variable table to be used by the job stream and the run cycle.	"vartable" on page 188

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.

On the Job Scheduling Console you can define the time zone at job stream level only. The time zone field displayed for run cycles and time restrictions is a read-only field.

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_PRESIMPLE 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#JOBLOTS
  PROMPT PRMT3

B236153_00#JOB_FTA
  FOLLOWS JOBMDM
END
```

See also

To create a job stream 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**→**Job Stream**
4. Specify your choices in the Properties - Job Stream panel.

For the equivalent Job Scheduling Console task, see the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

Job stream definition keyword details

This section describes the job stream definition keywords listed in table Table 34 on page 153.

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 13, "Managing time zones," on page 445 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 182.

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 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 286 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** and Job Scheduling Console.
- 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/yyyymdd_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 target workstations for the job or job stream. Set the **bm check deadline** option on each workstation on which you want to be notified of the deadline expiration. To disable the option and not check deadlines, enter a value of zero. You do not need to do this if you are setting deadlines for critical jobs.

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 13, "Managing time zones," on page 445 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.

Example2 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.

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` will start 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.

- The following example does not start the testjob2 job iteration until job testjob1 has completed successfully:
testjob2 every 15 follows testjob1

except

Defines the dates that are exceptions to the **on** dates of a job stream. See “on” on page 173 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 173.

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.

1 The scheduler classifies follows dependencies as *internal* when they are specified
1 only by their job name within the job stream. It classifies them as *external* when
1 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 46.

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.job
```

Do not launch sked6 until jobx in the job stream skedx on network agent cluster4 has completed successfully:

```
sked6 follows cluster4::site4#skedx.job
```

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 535), 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]
  [rcondsucc "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 535.

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 416.

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 165 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. 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 164 for more information.

Syntax

on [**runcycle** *name*]

[**validfrom** *date*] [**validto** *date*]

[**description** “*text*”]

[**variable** *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.

validfrom *date* ... **validto** *date*

1
1
1
1
1

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 **validto** 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.

variable

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 **validfrom** 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 **validfrom** 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:

```
1 JobstreamName Workstation Draft ValidFrom ValidTo UpdatedBy UpdatedOn LockedBy
1 -----
1 TESTCLI SITE2 Y 08/25/2008 - mdmDBE4 08/25/2008 mdmDBE4
1
1 SCHEDULE W5#TESTCLI VALIDFROM 08/25/2008 TIMEZONE ACT
1 DESCRIPTION "Job stream with several run cycle settings."
1 DRAFT
1 ON RUNCYCLE M5 VALIDFROM 08/25/2008
1 DESCRIPTION "monthly"
1 "FREQ=MONTHLY;INTERVAL=5;BYMONTHDAY=-3,1"
1 ( AT 0000 )
1 ON RUNCYCLE W4 VALIDFROM 08/25/2008
1 DESCRIPTION "weekly"
1 "FREQ=WEEKLY;INTERVAL=5;BYDAY=MO,WE"
1 FDNEXT ( AT 0000 )
1 ON RUNCYCLE D3 VALIDFROM 08/25/2008
1 DESCRIPTION "daily"
1 "FREQ=DAILY;INTERVAL=2"
1 FDPREV ( AT 0000 )
1 ON RUNCYCLE C2 VALIDFROM 08/25/2008
1 DESCRIPTION "calendar"
1 ITALY +2 DAYS
1 ( AT 0000 )
1 ON RUNCYCLE M6 VALIDFROM 08/25/2008
1 DESCRIPTION "monthly"
1 "FREQ=MONTHLY;INTERVAL=2;BYDAY=1MO,1TH,2WE"
1 ( AT 0000 +2 DAYS )
1 ON RUNCYCLE Y7 VALIDFROM 08/25/2008
1 DESCRIPTION "yearly"
1 "FREQ=YEARLY;INTERVAL=7"
```

```

1          ( AT 0100 )
1  ON RUNCYCLE S1 VALIDFROM 08/25/2008
1      08/10/2008,08/18/2008,08/20/2008,08/25/2008
1      ( AT 0000 UNTIL 0000 +1 DAYS ONUNTIL SUPPR DEADLINE 0000 +2 DAYS )
1  EXCEPT RUNCYCLE S1 VALIDFROM 08/25/2008
1      DESCRIPTION "simple"
1      08/26/2008,08/28/2008,08/30/2008,09/13/2008
1      ( AT 0000 )
1
1  CARRYFORWARD
1  MATCHING SAMEDAY
1  FOLLOWS LAB235004#SROBY2.@
1  FOLLOWS X8#COPYOFJS2.RR
1  FOLLOWS XA15::TPA
1  KEYSCHED
1  LIMIT 22
1  PRIORITY 15
1  :
1  X8#PIPP0 AS JOBTC
1  CONFIRMED
1  PRIORITY 13
1  KEYJOB
1  FOLLOWS W5#POPO.@
1  FOLLOWS X8#JS2.F3
1  END
1
1  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 144 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 backslash **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 144 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. Possible values are from **0000** to **2359**.

tzname Specifies the time zone to be used when calculating the start time. See Chapter 13, "Managing time zones," on page 445 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 46.

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 Job Scheduling Console and 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 13, “Managing time zones,” on page 445.

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 a 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 on 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 13, “Managing time zones,” on page 445 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 13, "Managing time zones," on page 445 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 new job stream date. If the until and the new job stream 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

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 **valid from** and **valid to**. For more information refer to Chapter 9, “Managing objects in the database - composer,” on page 199.

variable

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

variable *tablename*

Arguments

variable *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 101 for an overview of scheduling event rules.

Syntax

Unlike all other scheduling object definitions, 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 200 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 35 explains the meaning of the notation that follows each language element. *n* represents an unbounded number.

Table 35. 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:

- **&**; for **&**
- **>**; for **>**
- **<**; for **<**
- **"**; 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.

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. Click on the event types to see their properties.

Table Table 36 lists the TWSObjectsMonitor events.

Table 36. 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

Table 36. *TWSObjectsMonitor* events. (continued)

Event type	This event is sent when...
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

Table Table 37 lists the *FileMonitor* events.

Table 37. *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 495 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:

TECEventForwarder

Forwards the event to an external TEC (Tivoli Enterprise Console) server (or any other application capable of listening to events in TEC format).

MailSender

Connects to an SMTP server to send an e-mail.

MessageLogger

Logs the occurrence of a situation in an internal auditing database.

TWSAction

Runs one of the following conman commands:

- submit job (sbj)
- submit job stream (sbs)
- submit command (sbd)
- reply to prompt (reply)

GenericAction

Runs non-Tivoli Workload Scheduler commands. The commands are run on the same computer where the event processor runs.

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. Click on the action types to see their properties.

Table 38. Action types by action provider.

Action provider	Action types
TECEventForwarder	TECFWD
MailSender	SendMail
MessageLogger	PostOperatorMessage
TWSAction	sbs (SubmitJobStream)
	sbj (SubmitJob)
	sbd (SubmitAdHocJob)
	reply (ReplyPrompt)
GenericAction	RunCommand

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 113.

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:

- `&` for `&`
- `>` for `>`
- `<` for `<`
- `"` 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 525 for a list of parameters, or property names, available for every action type.

value See "Action providers and definitions" on page 525 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 e-mail.

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 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 e-mail is sent to the evening operator. This is accomplished by defining two sequential event conditions that have to be 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" encoding="UTF-8"?>
<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 e-mail 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="TWSObjectMonitor"
      eventType="JobSubmit">
      <filteringPredicate>
        <attributeFilter name="JobStreamWorkstation" operator="eq">
          <value>ACCREC03</value>
        </attributeFilter>
      </filteringPredicate>
    </eventCondition>
  </EventRule>
</eventRuleSet>
```

```

        <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 109 to find more event rule examples.

See also

To create an event rule definition in the Tivoli 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.

For the equivalent Job Scheduling Console task, see "Launching the Tivoli Dynamic Workload Console Event Rule Editor from the Job Scheduling Console" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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:

- “Using the composer command-line program”
- “Running commands from composer” on page 204

Using 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

Refer to Table 45 on page 225 and Table 46 on page 231 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:

MAESTROL

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 *MAESTROL* 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/locaopts* file. The default command prompt is a dash (-). To select a different prompt, edit the *composer prompt* option in the *locaopts* 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 *locaopts* 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 [connection_parameters] [-defaulttwscpu] ["command  
&[command]][...]"
```

where:

connection_parameters

Represents the set of parameters that rule the interaction between the product interface, **composer** running on the master domain manager in this case, and the WebSphere Application Server infrastructure using HTTP or HTTPS. The parameters can be supplied in any of these ways:

- Using the default values in the **useropts** and **locaopts** files (do not supply connection parameters as part of the command)
- Specifically supplying all of the parameters which are different from those defined in the **useropts** and **locaopts** files (see the syntax below)
- Identifying a file which contains the same parameters (see the syntax below)

Use the following syntax to specify the parameters:

```
[-host hostname]  
[-port port_number]  
[-protocol protocol_name]  
[-proxy proxy_name]  
[-proxyport proxy_port_number]
```

[-password *user_password*]
[-timeout *timeout*]
[-username *username*]
| **[-file***filename*]

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 with HTTP protocol.

proxy_port_number

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

user_password

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

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 communication request failed.

username

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

Note: When accessing the **composer** command line the user with the *username* and *password* specified in the connection parameters must be defined as system user on the operating system where the master domain manager is installed and must have the authorization defined in the security file on the master domain manager.

-file *filename*

Indicates an alternate custom properties file containing the settings for the connection parameters, used in place of the **useropts** and **localopts** files.

If any of these parameters is omitted when invoking **composer**, Tivoli Workload Scheduler searches for a value first in the **useropts** file and then in the **localopts** file. If no value is found then an error is displayed. Refer to "Setting up options for using the user interfaces" on page 41 for information on **useropts** and **localopts** files.

-defaultws *twscpu*

Indicates the default workstation **composer** uses for identifying the scheduling object when a specific workstation is not specified

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

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 39 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 39. Scheduling objects filtering criteria

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
workstation	<i>workstationname</i>	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
	variable	Applies the command to the workstations which refer the specified variable table.	mod ws=@; filter variable =table2
domain	<i>domainname</i>	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	<i>promptname</i>	Applies the command to the global prompts whose name satisfies the criteria.	lock prompt=p@
Windows user	<i>workstationname# username</i>	Applies the command to the users whose identifier satisfies the criteria.	list users=cpu1#operator?
resource	<i>workstationname# resourcename</i>	Applies the command to the resources whose identifier satisfies the criteria.	print res=cpu?#operator?
variable	<i>variablename</i>	Applies the command to the parameters whose name satisfies the criteria.	delete vb=mytable.myparm@
job definition	<i>jobname</i>	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 39. Scheduling objects filtering criteria (continued)

Scheduling object	Filter keywords or fields that can be filtered	Description	Example
job stream	<i>workstationname# jobstreamname</i>	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
	Rcvaritable	Applies the command to the run cycles in the job streams that refer to the variable table specified in the filter.	list js=@#@; filter Rcvaritable=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	<i>eventrulename</i>	Applies the command to the event rules that include an action on a specific job or job stream.	list er=@; filter js=accrcjs5
vartable	<i>vartablename</i>	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 call, non-working days calendar VACATIONS, and containing a job with job definition CPUA#job01.

Delimiters and special characters

Table 40 lists characters have special meanings in composer commands.

Table 40. Delimiters and special characters for composer

Character	Description
&	Command delimiter. See “Running the composer program” on page 201.
;	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

Command descriptions

Table 41 on page 208 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 41. List of composer commands

Command	Short Name	Description	See page
add	a	Adds scheduling objects.	"add" on page 213
authenticate	au	Changes the credentials of the user running composer .	"authenticate" on page 215
continue		Ignores the next error.	"continue" on page 215
create	cr	Creates a text file for editing containing an object definition from the database.	"create - extract" on page 216
delete	de	Deletes scheduling objects.	"delete" on page 219
display	di	Displays the details of the specified scheduling object.	"display" on page 223
edit	ed	Edits a file.	"edit" on page 227
exit	e	Exits composer .	"exit" on page 228
extract	ex	Extracts an object definition from the database and writes it in a text file.	"create - extract" on page 216
help	h	Invoke the help on line for a command.	"help" on page 228
list	l	Lists scheduling objects.	"list, print" on page 229
lock		Locks the access to database objects.	"lock" on page 233
modify	m	Modifies scheduling objects.	"modify" on page 237
new	n	Creates scheduling objects using a text file where the object definition is inserted on line.	"new" on page 241
print		Prints scheduling objects.	"display" on page 223
redo		Edits and reruns the previous command.	"redo" on page 243
rename	rn	Changes the object name.	"rename" on page 244
replace	rep	Replaces scheduling objects.	"replace" on page 247
unlock		Releases lock on the scheduling object defined in the database.	"unlock" on page 248
validate	val	Validates the syntax, semantic, and data integrity of an object definition.	"validate" on page 251
version	v	Displays the composer command-line program banner.	"version" on page 252
<i>system command</i>		Invokes an operating system command.	"system command" on page 248

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 42 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 42. Object identifiers for each type of object defined in the database

Object type	Object identifiers
domain	<i>domainname</i>
workstation	<i>workstationname</i> (checked across workstations and workstation classes)
workstation class	<i>workstationclassname</i> (checked across workstations and workstation classes)
calendar	<i>calendarname</i>
job definition	<i>workstationname</i> and <i>jobname</i>
Windows user	<i>workstationname</i> and <i>username</i>
job stream	<i>workstationname</i> and <i>jobstreamname</i> and, if defined, <i>validfrom</i>
job within a job stream	<i>workstationname</i> and <i>jobstreamname</i> , <i>jobname</i> , and, if defined, <i>validfrom</i>
resource	<i>workstationname</i> and <i>resourcename</i>
prompt	<i>promptname</i>
variable table	<i>variabletablename</i>
variable	<i>variabletablename.variablename</i>
event rule	<i>eventrulename</i>

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 43 shows all cases when a referenced object can be deleted even if other objects reference it:

Table 43. 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 44 describes how the product behaves when it is requested to delete an object referenced by another object with using a specific relationship:

Table 44. 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.

Table 44. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
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.
	windows user B	windows 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.
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.

Table 44. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
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.
variable A	job stream B	variable A is used in job stream B in: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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

Table 44. Referential integrity check when deleting an object from the database (continued)

Object to be deleted	Referenced by object	Relationship	Delete rule
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 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 188 for XML reference and see “The composer editor” on page 200 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

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*. Event rule

management is not supported by the Job Scheduling Console.

authenticate

Switches to another user credentials while running **composer**.

Authorization

Any user authorized to run **composer** is authorized to issue this command.

Syntax

```
authenticate [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 **tw_s_user1** with password **mypasswd1** from within the **composer** command-line program, run the following command:

```
authenticate username=tw_s_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
```

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 "continue&delete cpu=site4&print cpu=@"
```

create - 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 | extract filename from
{[calendars | calendar | cal=calname] |
[eventrule | erule | er=eventrulename] |
[parms | parm | vb=[tablename.]variablename] |
[vartable | vt=tablename] |
[prompts | prom=promptname] |
[resources | resource | res=[workstation#]resourcename] |
[cpu={workstationname | workstationclassname | domainname}] |
[workstation | ws=workstationname] |
[workstationclass | wsc=workstationclassname] |
[domain | dom=domainname] |
[jobs | jobdefinition | jd=[workstation#]jobname] |
[sched | jobstream | js= [workstation#]jstreamname
  [validfrom date | validto date | validin date date]
  [;full]} |
[users | user=[workstation#]username]}
[;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.

variable | 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 *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* 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 *workstationname#jobname* follows, copies the *jobname* job of the *workstationname* workstation on which the job runs 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 *jobname*.

sched | jobstream | js

If no argument follows, copies all job stream definitions into the file.

If argument *workstationname#jstreamname* follows, copies the *jstreamname* job stream of the *workstationname* workstation on which the job stream 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 *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*. The password field is not copied for security reasons.

;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 398.

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:

```
create allrules.xml from erule=@
```

See also

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*. Event rule management and variable tables are not supported by the Job Scheduling Console.

delete

Deletes object definitions in the database.

Authorization

You must have *delete* access to the objects being deleted.

Syntax

```
delete  
[calendars | calendar | cal=calname] |  
[eventrule | erule | er=eventrulename] |  
[parms | parm | vb=[tablename.]variablename] |  
[variable | vt=tablename] |  
[prompts | prom=promptname] |  
[resources | resource | res=[workstationname#]resourcename] |  
[cpu={workstationname | workstationclassname | domainname}]  
[workstation | ws=workstationname] |  
[workstationclass | wsc=workstationclassname] |  
[domain | dom=domainname] |  
[jobs | jobdefinition | jd=[workstationname#]jobname] |  
[sched | jobstream | js= [workstationname#]jstreamname  
  [validfrom date|validto date |validin 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.

variable | 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 *workstationname#resourcename* follows, deletes the *resourcename* resource of the *workstationname* workstation on which the resource is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *resourcename*.

cpu Deletes 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, deletes all workstation definitions.

If argument *workstationname* follows, deletes the *workstationname* workstation. Wildcard characters are permitted.

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.

jobs | jobdefinition | jd

If no argument follows, deletes all job definitions.

If argument *workstationname#jobname* follows, deletes the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *jobname*.

sched | jobstream | js

If no argument follows, deletes all job stream definitions.

If argument *workstationname#jstreamname* follows, deletes 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, deletes all user definitions.

If argument *workstationname#username* follows, deletes 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*. 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**.

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide* Event rule management is not supported by the Job Scheduling Console.

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
{[calendars | calendar | cal=calname] |
[eventrule | erule | er=eventrulename] |
[parms | parm | vb=variablename.]variablename] |
[table | vt=tablename] |
[prompts | prom=promptname] |
[resources | resource | res=[workstationname#]resourcename] |
[cpu={workstationname | workstationclassname | domainname}]
[workstation | ws=workstationname] |
[workstationclass | wsc=workstationclassname] |
[domain | dom=domainname] |
[jobs | jobdefinition | jd=[workstationname#]jobname] |
[sched | jobstream | js= [workstationname#]jstreamname
  [validfrom date | validto date | validin 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.

variable | 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 *workstationname#resourcename* follows, displays the *resourcename* resource of the *workstationname* workstation on which the resource is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* 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 *workstationname#jobname* follows, displays the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* 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 200.

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 45 shows an example of the output produced based on the value set for the **MAESTROCOLUMNS** variable.

Table 45. 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 45. 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"
Windows 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 199 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 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 display using the provided filters and click **Edit** to display their details.

For the equivalent Job Scheduling Console task, see "Working with Object Lists" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*. Event rule management is not available from the Job scheduling Console.

edit

Edits a file.

Authorization

Any user authorized to run composer is authorized to issue this command.

Syntax

```
edit 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 200 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
```

See also

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

exit

Exits the **composer** command line program.

Authorization

Any user authorized to run **composer** is authorized to issue this command.

Syntax

```
exit
```

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
```

help

Displays the on-line help for a command or displays the list of commands that can be issued from **composer**.

Authorization

Any user authorized to run **composer** is authorized to issue this command.

Syntax

```
help [commandname]
```

Arguments

commandname

The name of the **composer** command which you want to display on-line help about.

list, print

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 *MAESTROL*P local variable. The print command can be used to send the output to a local printer, if the *MAESTROL*P 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
{[calendars | calendar | cal=calname] |
[eventrule | erule | er=eventrulename] |
[parms | parm | vb=[tablename.]variablename] |
[table | vt=tablename] |
[prompts | prom=promptname] |
[resources | resource | res=[workstationname#]resourcename] |
[cpu={workstationname | workstationclassname | domainname}]
[workstation | ws=workstationname] |
[workstationclass | wsc=workstationclassname] |
[domain | dom=domainname] |
[jobs | jobdefinition | jd=[workstationname#]jobname] |
[sched | jobstream | js= [workstationname#]jstreamname
    [validfrom date |
    validto date | validin date date] |
[users | user=[workstationname#]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.

variable | 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 *workstationname#resourcename* follows, lists or prints the *resourcename* resource of the *workstationname* workstation on which the resource is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* 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 *workstationname#jobname* follows, lists or prints the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname*

is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* 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 200. 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 *MAESTROL*P local variable. The print command can be used to send the output to a local printer, if you set the *MAESTROL*P variable accordingly. Make sure the *MAESTROL*P 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 46 shows an example of the output produced according to the value set for the *MAESTROCOLUMNS* variable.

Table 46. 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"

Table 46. Output formats for displaying scheduling objects (continued)

Object Type	Output format if MAESTROCOLUMNS<120	Output format if MAESTROCOLUMNS ≥ 120
Domain	"DomainName : ParentDomain : Master : UpdatedOn : LockedBy"	"DomainName : ParentDomain : Master : UpdatedBy : UpdatedOn : LockedBy : LockedOn"
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"
Windows 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 199 for more information on how to set *MAESTROL*.

Examples

- To list all calendars, run the following command:

```
list calendars=@
```

this is a sample output:

```
Calendar Name      Updated On  Locked By
-----
HOLIDAYS           03/02/2006
PAYDAYS            03/02/2006
HOLIDAYS           03/02/2006
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 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	06/06/2007	-
EVENT-MULTIPLE2	filter		active	06/06/2007	-
EVENT-MULTIPLE3	filter		active	06/06/2007	-
M_SUCC_12_S	sequence	Y	inactive	06/07/2007	-
M_SUCC_12_S_A	filter		active	06/07/2007	-
M_SUCC_12_S_B	filter	Y	inactive	06/07/2007	-
NEWEVENTRULE	filter		active	06/01/2007	administrator

If MAESTROCOLUMNS≥120, the output looks something like this:

Event Rule Name	Type	Draft	Status	Updated On	Locked By
EVENT-MULTIPLE1	filter		active	06/06/2007	-
EVENT-MULTIPLE2	filter		active	06/06/2007	-
EVENT-MULTIPLE3	filter		active	06/06/2007	-
M_SUCC_12_S	sequence	Y	inactive	06/07/2007	-
M_SUCC_12_S_A	filter		active	06/07/2007	-
M_SUCC_12_S_B	filter	Y	inactive	06/07/2007	-
NEWEVENTRULE	filter		active	06/01/2007	administrator

See also

In the Tivoli 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.

For the equivalent Job Scheduling Console task, see "Working with Object Lists" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*. Event rule management is not available from the Job Scheduling Console.

lock

Locks the access to scheduling objects definitions in the database.

Authorization

You must have *modify* access to the object.

Syntax

lock

```
{[calendars | calendar | cal=calname] |  
[eventrule | erule | er=eventrulename]  
[parms | parm | vb=[tablename.]variablename] |  
[variable | 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  
  [validfrom date | validto date | validin date date]] |  
[users | user=[workstationname#]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.

variable | 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 *workstationname#resourcename* follows, locks the *resourcename* resource of the *workstationname* workstation on which the resource is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* 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 *workstationname#jobname* follows, locks the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *jobname*.

sched | jobstream | js

If no argument follows, locks all job stream definitions.

If argument *workstationname#jstreamname* follows, locks 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, locks all user definitions.

If argument *workstationname#username* follows, locks 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*.

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**.

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*. Event rule management is not available from the Job Scheduling Console.

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  
{[calendars | calendar | cal=calname] |  
[eventrule | erule | er=eventrulename] |  
[parms | parm | vb=[tablename.]variablename] |  
[variable | 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  
  [validfrom date | validto date | validin date date]  
  [;full]} |  
[users | user=[workstationname#]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 *workstationname#resourcename* follows, modifies the *resourcename* resource of the *workstationname* workstation on which the resource is defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* 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 *workstationname#jobname* follows, modifies the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *jobname*.

sched | jobstream | js

If no argument follows, modifies all job stream definitions.

If argument *workstationname#jstreamname* follows, modifies 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 Modifies all job definitions contained in the job stream.

users | user

If no argument follows, modifies all user definitions.

If argument *workstationname#username* follows, modifies 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*.

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 188 for XML reference and see "The composer editor" on page 200 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**.

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*. Event rule management is not available from the Job Scheduling Console.

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 |
variable |
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 188 for XML reference and see “The composer editor” on page 200 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 variable
```

To create a new variable definition, run:

```
new parameter
```

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. 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

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

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 **ds**.

- 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
display 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@
    erv
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
{calendars | calendar | cal |
parms | parm | vb |
  vartable | vt |
prompts | prom |
resources | 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 ca11 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 ca12 as new identifier to be assigned to calendar object previously named ca11.

For what concerns jobs, job streams, resources and Windows 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 152 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 Windows 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 *workstationname* 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 *workstationname*:

- 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 Tivoli 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.

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 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 213.

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.

- You run:

```
rep 2Q07rules.xml
```

See also

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

system command

Runs a system command.

Syntax

```
[ : | ! ] sys-command
```

Arguments

sys-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

```
{[calendars | calendar | cal=calname] |
[eventrule | erule | er=eventrulename] |
[parms | parm | vb=[tablename.]variablename] |
[table | vt=tablename] |
[prompts | prom=promptname] |
[resources | resource | res=[workstationname#]resourcename] |
[cpu={workstationname | workstationclassname | domainname}]
[workstation | ws=workstationname] |
[workstationclass | wsc=workstationclassname] |
[domain | dom=domainname] |
[jobs | jobdefinition | jd=[workstationname#]jobname] |
[sched | jobstream | js= [workstationname#]jstreamname
```

[*validfrom date* | *validto date* | *validin date date*] |
[**users** | **user**=[*workstationname#*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.

variable | **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 *workstationname#resourcename* follows, unlocks the *resourcename* resource of the *workstationname* workstation on which the resource is

defined. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* 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 *workstationname#jobname* follows, unlocks the *jobname* job of the *workstationname* workstation on which the job runs. If *workstationname* is omitted, the default is the workstation on which **composer** is running. Wildcard characters are permitted for both *workstationname* and *jobname*.

sched | jobstream | js

If no argument follows, unlocks all job stream definitions.

If argument *workstationname#jstreamname* follows, unlocks 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, unlocks all user definitions.

If argument *workstationname#username* follows, unlocks 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*.

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**.

For the equivalent Job Scheduling Console task, see "Managing database objects" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 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 188 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

Examples

To display the **composer** command line program banner, run the following command:

```
version
```

or:

```
v
```

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:

- “Using the conman command line program”
- “Running commands from conman” on page 257
- “Selecting jobs in commands” on page 259
- “Selecting job streams in commands” on page 267
- “Managing jobs and job streams from back-level agents” on page 274
- “Command descriptions” on page 274

Using 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.

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 not set, or set to any value other than **LONG**, long names are truncated to eight characters followed by a plus sign (+).

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

Represents the set of parameters that rule the interaction between the product interface, **conman** running on the master domain manager in this case, and the WebSphere Application Server infrastructure using HTTP or HTTPS. The parameters can be supplied in any of these ways:

- Using the default values in the **useropts** and **localopts** files (do not supply connection parameters as part of the command)
- Specifically supplying all of the parameters which are different from those defined in the **useropts** and **localopts** files (see the syntax below)
- Identifying a file which contains the same parameters (see the syntax below)

Use the following syntax to specify the parameters:

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

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 with HTTP.

proxy_port_number

Is the proxy port number used in the connection with HTTP.

user_password

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

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 user ID of the user running **conman**. This user name must be authorized to run the specific **conman** commands in the security file both on the connecting workstation and on the master domain manager, where the command actually runs.

-file filename

Indicates an alternate custom properties file containing the settings for the connection parameters, used in place of the **useropts** and **localopts** files.

If any of these parameters is omitted when invoking **conman**, Tivoli Workload Scheduler searches for the value first in the **useropts** file and then in the **localopts** file. If a setting for the parameter is not specified either in the connection parameters when invoking the command or inside the **useropts** and **localopts** files then an error is displayed. Refer to "Setting up options for using the user interfaces" on page 41 for information on **useropts** and **localopts** files.

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
```

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 47 lists characters having special meanings in **conman** commands:

Table 47. Delimiters and special characters for conman

Char.	Description
&	Command delimiter. See “Using the conman command line program” on page 253.
+	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: <i>*comment</i> or sj& <i>*comment</i>
>	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
>>	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

Table 47. Delimiters and special characters for conman (continued)

Char.	Description
	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.

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 page 268 for more information on job stream identifier.

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, "Job qualifiers" on page 261.

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 16, "Managing internetwork dependencies," on page 485.

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 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 13, "Managing time zones," on page 445 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 13, "Managing time zones," on page 445 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 13, "Managing time zones," on page 445 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 16, "Managing internetwork dependencies," on page 485.

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 page 268 for more information on job stream identifier.

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_ids* and not the *jobstreamnames*. 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 13, "Managing time zones," on page 445 for valid names.

lowtime

Specifies the lower limit of a time range, expressed in the same format as *time*. Jobs 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*. Jobs are selected that started at or before this time.

If **started** is used alone and it is preceded by + then the jobs selected are the jobs that have started running.

If **started** is used alone and it is preceded by ~ then the jobs selected are the jobs that have not started running.

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.

EXEC+

A transitory state when the job is launched 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.

INTRO+

A transitory state when the job is introduced 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 13, "Managing time zones," on page 445 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 alphanumeric 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 13, "Managing time zones," on page 445 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 13, "Managing time zones," on page 445 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 16, "Managing internetwork dependencies," on page 485.

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 page 268 for more information on job stream identifier.

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_ids* and not the *jobstreamnames*. 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 13, "Managing time zones," on page 445 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 13, "Managing time zones," on page 445 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 0AAAAAAAAAAAY3 must be identified in the **conman** command line as follows:

```
CPU1#@AAAAAAAAAAAY3
```

- 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 0AAAAAAAAAAAT1 running on CPU1 workstation, run the following command on Tivoli Workload Scheduler version 8.1, 8.2, or 8.2.1 agents:

```
sj CPU1#@AAAAAAAAAAAT1.job2
```

These changes will also be seen in reports and logs, and any other places where job stream names are printed or displayed.

Command descriptions

Table 48 lists the **conman** 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.

Table 48. List of conman commands

Command	Short Form	Description	Type ¹	Page
adddep	adj ads	Adds job or job stream dependencies.	F	"adddep job" on page 277, "adddep sched" on page 278
altpass		Alters a user object definition password.	F	"altpass" on page 280
altpri	ap	Alters job or job stream priorities.	F	"altpri" on page 281
bulk_discovery	bulk	Performs a bulk discovery. For use with IBM Tivoli Monitoring 6.1 (Tivoli Enterprise Portal).	F	"bulk_discovery" on page 282

Table 48. List of conman commands (continued)

Command	Short Form	Description	Type ¹	Page
cancel	cj cs	Cancels a job or a job stream.	F	"deldep job" on page 288, "cancel sched" on page 284
1 checkhealthstatus	chs	Invokes chkhltst service to check if mailbox can be successfully read by mailman or if there are errors in the mailbox header.	M, F, S	"checkhealthstatus" on page 285
confirm		Confirms job completion.	F	"confirm" on page 286
console		Assigns the Tivoli Workload Scheduler console.	F,S	"console" on page 287
continue		Ignores the next error.	F,S	"continue" on page 288
deldep	ddj dds	Deletes job or job stream dependencies.	F	"release job" on page 305, "deldep sched" on page 290
deployconf	deploy	Gets the latest monitoring configuration for the event monitoring engine on the workstation.	F,S	"deployconf" on page 291
display	df dj ds	Displays files, jobs, and job streams.	F,S ²	"display" on page 292
exit		Exits conman .	F,S	"exit" on page 294
fence		Sets Tivoli Workload Scheduler job fence.	F	"fence" on page 295
help ⁴		Displays command information.	F,S	"help" on page 296
kill		Stops an executing job.	F	"kill" on page 297
limit	lc ls	Changes a workstation or job stream job limit.	F	"limit cpu" on page 297, "limit sched" on page 299
link	lk	Opens workstation links.	F,S	"link" on page 299
listsym		Displays a list of Symphony log files.	F	"listsym" on page 302
recall	rc	Displays prompt messages.	F	"recall" on page 303
redo		Edits the previous command.	F,S	"redo" on page 304
release	rj rs	Releases job or job stream dependencies.	F	"cancel job" on page 282, "release sched" on page 307
reply		Replies to prompt message.	F	"reply" on page 308
rerun	rr	Reruns a job.	F	"rerun" on page 309
resource		Changes the number of resource units.	F	"resource" on page 312
setsym		Selects a Symphony log file.	F	"setsym" on page 313
showcpus	sc	Displays workstation and link information.	F,S	"showcpus" on page 313
showdomain		Displays domain information.	F,S	"showdomain" on page 319
showfiles	sf	Displays information about files.	F	"showfiles" on page 320
showjobs	sj	Displays information about jobs.	F	"showjobs" on page 322
showprompts	sp	Displays information about prompts.	F	"showprompts" on page 335

Table 48. List of conman commands (continued)

Command	Short Form	Description	Type ¹	Page
showresources	sr	Displays information about resources.	F	"showresources" on page 337
showschedules	ss	Displays information about job streams.	F	"showschedules" on page 340
shutdown	shut	Stops Tivoli Workload Scheduler production processes.	F,S	"shutdown" on page 345
start		Starts Tivoli Workload Scheduler production processes.	F,S	"start" on page 345
startappserver		Starts the embedded WebSphere Application Server process	F,S	"startappserver" on page 348
starteventprocessor	startevtp	Starts the event processing server.	M ⁵	"starteventprocessor" on page 348
startmon	startm	Starts the monman process that turns on the event monitoring engine on the agent.	F,S	"startmon" on page 349
status		Displays Tivoli Workload Scheduler production status.	F,S	"status" on page 349
stop		Stops Tivoli Workload Scheduler production processes.	F,S	"stop" on page 350
stop ;progressive		Stops Tivoli Workload Scheduler production processes hierarchically.		"stop ;progressive" on page 352
stopappserver		Stops the embedded WebSphere Application Server process	F,S	"stopappserver" on page 353
stopeventprocessor	stopevtp	Stops the event processing server.	M ⁵	"stopeventprocessor" on page 354
stopmon	stopm	Stops the event monitoring engine on the agent.	F,S	"stopmon" on page 355
submit	sbd sbf sbj sbs	Submits a command, file, job, or job stream.	F,S ³	"submit docommand" on page 355, "submit file" on page 359, "SubmitJob" on page 528, "submit sched" on page 365
switcheventprocessor	switchevtp	Switches the event processing server from master domain managers to backup masters or vice versa.	M	"switcheventprocessor" on page 368
switchmgr		Switches the domain manager.	F	"switchmgr" on page 369
<i>sys-command</i>		Sends a command to the system.	F,S	"system" on page 370
tellop	to	Sends a message to the console.	F,S	"tellop" on page 370
unlink		Closes workstation links.	F,S	"unlink" on page 371
version	v	Displays conman 's command line program banner.	F,S	"version" on page 374

1. Workstation types: master domain managers and backup masters (M), domain managers and fault-tolerant agents (F), standard agents (S).
2. You can only display files on a standard agent.
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.

4. Not available on supported Windows operating system.
5. Includes workstations installed as backup masters but used as ordinary fault-tolerant agents.

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

```
adj jobselect
    [;dependency[:...]]
    [;noask]
```

Arguments

jobselect

See “Selecting jobs in commands” on page 259.

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 target workstations for the job or job stream. This does not apply to critical jobs. For information, refer to *IBM Tivoli Workload Scheduler Administration Guide*.

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.

For the equivalent Job Scheduling Console task, see "Creating Dependencies between Jobs" and "Adding External Dependencies to a Job Stream" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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

```
ads jstreamselect  
    [dependency[:...]]  
    [noask]
```

Arguments

jstreamselect

See "Selecting job streams in commands" on page 267.

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** variable is set to a value higher than 0 in the `localopts` configuration file on the target workstations for the job or job stream. This does not apply to critical jobs. For information, refer to *IBM Tivoli Workload Scheduler Administration Guide*.

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 337 and “showprompts” on page 335.

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 CUA#TEST(0900 02/19/06) ; follows CUA#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.

For the equivalent Job Scheduling Console task, see "Creating Dependencies within a Job Stream" and "Creating Dependencies between Job Streams" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 Windows user definition. For more information refer to "Windows user definition" on page 141. 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 upper case for this field even though you used the mixed case when specifying the *[domain\]username* in the Windows user definition. For more information, refer to "Windows user definition" on page 141.

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 "Windows user definition" on page 141.

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"
```

See also

For the equivalent Job Scheduling Console task, see "Changing Windows User Passwords" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

altpri

Alters the priority of a job or job stream.

You must have *altpri* access to the job or job stream.

Syntax

```
ap jobselect | jstreamselect  
    [pri]  
    [noask]
```

Arguments

jobselect

See "Selecting jobs in commands" on page 259.

jstreamselect

See "Selecting job streams in commands" on page 267.

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

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.logfile**.

cancel job

Cancels a job.

You must have *cancel* access to the job.

Syntax

```
cj jobselect  
      [;pend]  
      [;noask]
```

Arguments

jobselect

See “Selecting jobs in commands” on page 259.

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” on page 490.

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**.

For the equivalent Job Scheduling Console task, see "Deleting a Job from a Job Stream" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

cancel sched

Cancels a job stream.

You must have *cancel* access to the job stream.

Syntax

```
cs jstreamselect
    [;pend]
    [;noask]
```

Arguments

jstreamselect

See "Selecting job streams in commands" on page 267.

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**.

For the equivalent Job Scheduling Console task, see "Deleting Job Streams" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

checkhealthstatus

Invokes **chkhltst** service to check if mailbox can be successfully read by **mailman** or if there are errors in the mailbox header.

Syntax

```
checkhealthstatus [workstation]
```

Comments

Use this command to launch the **chkhltst** service to verify that the workstation mailbox can be successfully read, and check if there are errors in the mailbox header. Resulting information is logged in the **TWSMERGE.log** file of the domain manager as follows:

- If a file system error occurs while opening the mailbox, the following message is reported: **AWSBDY126E** An error occurred opening the Mailbox.msg file in CPU_NAME.
- If an error occurs while opening the mailbox because mailman is reading the mailbox, the following message is reported: **AWSBDY123I** The Mailbox.msg file in CPU_NAME is correctly read by Mailman.

- 1 • If the mailbox is correctly opened, but an error occurs while reading the header,
1 the following message is reported: AWSBDY125E An error occurred reading the
1 header of the Mailbox.msg file in CPU_NAME.
- 1 • If the mailbox is correctly opened and no error occurs while reading the header,
1 the following message is reported: AWSBDY124W The Mailbox.msg file in
1 CPU_NAME is not read by Mailman.

1 If *workstation* is not specified, the service is launched locally.

1 Examples

1 To check the health status of the *site1* workstation, launch the following command:
1 `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 jobselect
      ;{succ | abend}
      [;noask]
```

Arguments

jobselect

See “Selecting jobs in commands” on page 259.

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 159. For more information about EXTERNAL jobs, see “Managing internetwork dependencies” on page 490.

Table 49 shows the effect of the **confirm** command on the various states of jobs:

Table 49. 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

Table 49. State change after confirm command (continued)

Initial Job State	State after confirm ;succ	State after confirm ;abend
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
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
  [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:

- 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
```

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 "continue&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

```
ddj jobselect  
;dependency[;...]  
[;noask]
```

Arguments

jobselect

See "Selecting jobs in commands" on page 259.

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.

For the equivalent Job Scheduling Console task, see "Managing Jobs" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

deldep sched

Deletes dependencies from a job stream.

You must have *deldep* access to the job stream.

Syntax

```
dds jstreamselect
    ;dependency[;...]
    [;noask]
```

Arguments

jstreamselect

See "Selecting jobs in commands" on page 259.

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.

For the equivalent Job Scheduling Console task, see "Managing Job Streams" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

deployconf

Gets the latest monitoring configuration for the event monitoring engine on the workstation.

Syntax

```
deployconf [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

df *filename* [**;offline**]

dj *jobselect* [**;offline**]

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 if 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 259. 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 267.

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 254.

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_PRESIMPLE 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**.

For the equivalent Job Scheduling Console task, see "Managing Jobs" and "Managing Job Streams" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

exit

Exits the **conman** command line program.

Syntax

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 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 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**→**Fence...**

For the equivalent Job Scheduling Console task, see "Viewing and Modifying Workstation Properties" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

help

Displays help information about commands. Not available in Windows.

Syntax

help *command*

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. You can also enter the following keywords:

commands

Lists all conman commands.

jobselect

Lists information about selecting jobs for commands.

jobstates

Lists information about job states.

schedselect

Lists information about selecting job streams for commands.

schedstates

Lists information about job stream states.

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
```

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 jobselect  
    [;noask]
```

Arguments

jobselect

See “Selecting jobs in commands” on page 259.

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 456, run the following command:

```
k 456
```

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

```
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 job limit. You can enter from 0 to 1024. Entering **system** sets the job limit to zero. For a broker workstation entering **system**, sets no limit.

Note: When you set **lc** to zero:

- 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.

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 pre-production 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 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**→**Limit...**

For the equivalent Job Scheduling Console task, see "Viewing and Modifying Workstation Properties" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 171. You must have *limit* access to the job stream.

Syntax

```
ls jstreamselect
    ;limit
    [;noask]
```

Arguments

jstreamselect

See "Selecting job streams in commands" on page 267.

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#apwk1y;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...**

For the equivalent Job Scheduling Console task, see "Managing Job Streams" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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

```
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.

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 121.

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 on page 301 and Table 50 on page 301 show the links opened by **link** commands run by users in various locations in the network.

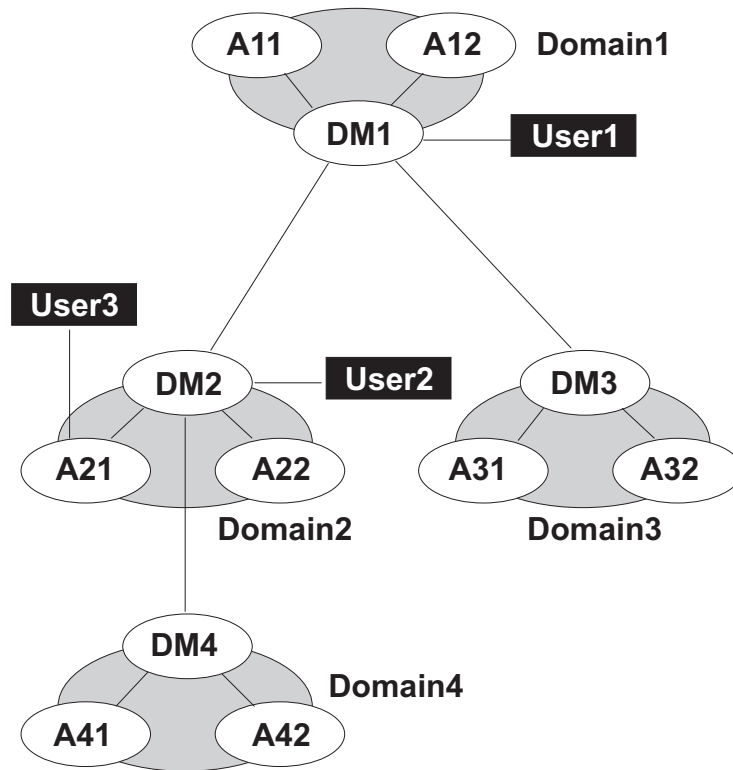


Figure 22. Network links

DM n are domain managers and A mn are agents.

Table 50. 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 [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 254.

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	Exp
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	Exp
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	Exp
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

```
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 254.

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
```

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.

- itext* Inserts text before the character above the *i*.
- rtext* 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.
- >rtext* Replaces characters at the end of the line with *text*.

Directive Examples

- ddd** Deletes the three characters above the **ds**.
- 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

```
rj jobselect
    [;dependency[;...]]
    [;noask]
```

Arguments

jobsselect

Specifies the job or jobs to be released. See “Selecting jobs in commands” on page 259.

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**.

For the equivalent Job Scheduling Console task, see "Releasing a Job Instance from Dependencies" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

release sched

Releases job streams from dependencies.

You must have *release* access to the job stream.

Syntax

```
rs jstreamselect
    [dependency[:...]]
    [noask]
```

Arguments

jstreamselect

See "Selecting job streams in commands" on page 267.

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 259.

Examples

To release job stream instance with *jobstream_id* 0AAAAAAAAAAAABSE from all of its dependencies, run the following command:

```
rs 0AAAAAAAAAAAABSE; 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#0;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**.

For the equivalent Job Scheduling Console task, see "Releasing a Job Stream Instance from Dependencies" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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  
  { 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 `arprmt`, 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

```
rr jobselect
    [;from=wkstat#]job[
    ;at=time]
    [;pri=pri]
    [;noask]
```

```
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 13, "Managing time zones," on page 445 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 389.

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 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 **Rerun...**

For the equivalent Job Scheduling Console task, see "Rerunning a Job Instance" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

resource

Changes the number of total units of a resource.

You must have *resource* access to the resource.

Syntax

```
resource [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:

```
res 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...**

For the equivalent Job Scheduling Console task, see "Managing Resources in the Database" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 [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

```
sc [[domain!]workstation]
    [;info |;link]
    [;offline]
```

```
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 254.

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.

Results

1 When the getmon parameter is not used, the output of the command is produced in
1 three formats, **standard**, **info**, and **link**. The default value is **standard**. The
1 meaning of the characters displayed depends on the type of format you select.

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	8.5	US/Pacific	Linux 2.6.5-7.191-s390 #1 SM
FTA1	8.5		Linux 2.4.9-e.24 #1 Tue May
FTA2	8.5		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
1 MASTER	360	*WNT MASTER	10	0	06/04/2010	1348	I J E		MASTERDM
1 FTA1	360	WNT FTA	10	0	06/04/2010	1348	FTI JW M		MASTERDM
1 FTA2	360	WNT FTA	10	0	06/04/2010	1348	FTI JW M		MASTERDM
1 FTA3	360	WNT MANAGER	10	0	06/04/2010	1348	LTI JW M		DOMAIN1
1 FTA4	360	WNT FTA	10	0	06/04/2010	1348	F I J M		DOMAIN1
1 FTA5	360	WNT FTA	10	0	06/04/2010	1348	I J M		DOMAIN1
1 SA1	360	WNT S-AGENT	10	0	06/04/2010	1348	F I J M		DOMAIN1
1 XA_FTA4	360	OTHR X-AGENT	10	0	06/04/2010	1348	L I J M		DOMAIN1
1 FTA6	360	WNT MANAGER	10	0	06/04/2010	1348	F I J M		DOMAIN2
1 FTA7	360	WNT FTA	10	0	06/04/2010	1349	F I J M		DOMAIN2
1 FTA7	360	WNT FTA	10	0	06/04/2010	1349	F I J M		DOMAIN2
1 BROKER	360	OTHR BROKER	10	0	06/04/2010	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
1 MASTER	360	*WNT MASTER	10	0	06/04/2010	1348	I J E		MASTERDM
1 FTA1	360	WNT FTA	10	0	06/04/2010	1348	FTI JW M		MASTERDM
1 FTA2	360	WNT FTA	10	0	06/04/2010	1348	FTI JW M		MASTERDM
1 FTA3	360	WNT MANAGER	10	0	06/04/2010	1348	FTI JW M		DOMAIN1
1 FTA4	360	WNT FTA	10	0	06/04/2010	1348	F I J M		DOMAIN1
1 FTA5	360	WNT FTA	10	0	06/04/2010	1348	L I M		DOMAIN1
1 SA1	360	WNT S-AGENT	10	0	06/04/2010	1348	F I J M		DOMAIN1

```

1      XA_FTA4 360 OTHR X-AGENT 10 0 06/04/2010 1348 L I J M          DOMAIN1
1      FTA6   360 WNT  MANAGER 10 0 06/04/2010 1348 F I J M          DOMAIN2
1      FTA7   360 WNT  FTA      10 0 06/04/2010 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:

```

1      CPUID  RUN    NODE    LIMIT FENCE  DATE    TIME    STATE    METHOD  DOMAIN
1      MASTER 360 *WNT  MASTER 10 0 06/04/2010 1348 I J E    MASTERDM
1      FTA1   360 WNT  FTA     10 0 06/04/2010 1348 FTI JW M  MASTERDM
1      FTA2   360 WNT  FTA     10 0 06/04/2010 1348 FTI JW M  MASTERDM
1      FTA3   360 WNT  MANAGER 10 0 06/04/2010 1348 FTI JW M  DOMAIN1
1      FTA4   360 WNT  FTA     10 0 06/04/2010 1348 F I J M  DOMAIN1
1      FTA5   360 WNT  FTA     10 0 06/04/2010 1348 F I J M  DOMAIN1
1      SA1    360 WNT  S-AGENT 10 0 06/04/2010 1348 F I J M  DOMAIN1
1      XA_FTA4 360 OTHR X-AGENT 10 0 06/04/2010 1348 L I J M  DOMAIN1
1      FTA6   360 WNT  MANAGER 10 0 06/04/2010 1348 F I J M  DOMAIN2
1      FTA7   360 WNT  FTA     10 0 06/04/2010 1349 F I J M  DOMAIN2

```

- 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 : 04/22/2010 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 Tivoli Dynamic Workload Console:

- Click **Tivoli Workload Scheduler**→**Scheduling Environment**→**Monitor**→**Monitor Workstations**
- Select **All Workstations in plan** or another predefined task name
- Choose an engine name, or specify connection properties, and click **OK**

For the equivalent Job Scheduling Console task, see "Working with Object Lists" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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

Workstation types are as follows:

- MASTER
- MANAGER
- FTA
- S-AGENT

- X-AGENT

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] [I] [J] [W|H|X] [M] [E|e] [D] [A|R]

where:

- L** The primary link is open (linked) to its domain/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).
- I** The **jobman** program has completed startup initialization.
- J** The **jobman** program is running.
- W** The **writer** process is active on the workstation.

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.
- 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.

- 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.

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, no information is listed.

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's links. 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 [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 254.

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-CPU	CPU-Type
MASTERDM	*MASTER	MASTER
DOM1	FTA1	MANAGER
DOM2	FTA2	MANAGER

See also

In the Tivoli 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**

For the equivalent Job Scheduling Console task, see "Displaying and Modifying Domain Properties" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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

```
sf [[workstation#]file]
    [;state[;...]]
    [;keys]
    [;offline]

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 254.

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_JOB1 HOLD 10 (11/26) parms FILE_JOB1`

```

See also

In the Tivoli 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

```
sj [jobsselect]  
    [;keys | info | step | logon | crit | keys retcod]  
    [;short | single]  
    [;offline]  
    [;showid]
```

```
sj [jobsselect]  
    [;deps[;keys | info | logon]]  
    [;short | single]  
    [;offline]  
    [;showid]
```

```
sj [jobsselect | [workstation#]jobnumber.hhmm]  
    [;stdlist[;keys]]  
    [;short | single]  
    [;offline]  
    [;showid]
```

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 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.

jobsselect
See “Selecting jobs in commands” on page 259.

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 254.

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

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.

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      {0AAAAAAAAAAAAACRZ}
                JDOC SUCC 10 11/26 00:01      0 #J25565
```

The job stream identifier 0AAAAAAAAAAAAACRZ 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)
                jobaa HOLD 10(02/24)(00:01)      {02/20/06}
                TEST1; JOBC          TEST2; JOB1
                JS18(0600 02/24/06).@
```

CPUA#JS25.JOB1 Dependencies are:

```
CPUA        #JS25      0600 02/24 ***** HOLD 10(02/24)
                JOBC HOLD 10(02/24)(00:01)      {02/20/06}
                jobaa HOLD 10(02/24)(00:01)      JOB1
                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),(0AAAAAAAAAAAAABQM)].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.

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 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**

For the equivalent Job Scheduling Console task, see "Working with Object Lists" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 [0] is displayed. For more information on pending predecessors and orphaned dependencies refer to “Managing external follows dependencies for jobs and job streams” on page 46.
- 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 #Jnnnnn.
- 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.
- [**Confirm**] 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	***** (M235062+#)JOBMDM	HOLD		55(03/04) 30(03/04)			[03/04/06]; #33 #1(PRMT3);-16 JOBSLOTS-
MYCPU+#SCHED_F+	1010	03/04	***** (M235062+#)JOBMDM	HOLD		55(03/04) 30(03/04)			[03/04/06]; #34 #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/g11
                (B236153+#)JOB_FTA echo job12
M235062+#SCHED_22 0600 03/07
                JOBMDM /usr/acct/scripts/g11
                (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/g11
                (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 #Jnnnnnn.

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 #Jnnnnn.

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 @#0;crit
```

is:

CPU	Schedule	SchedTime	Job	State	Pr	(Est) Start	(Est) Elapse	CP	Crit 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.

Depts 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.

Depts;keys format

Jobs and job streams that have follows dependencies are listed, one on each line.

Depts;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.

Depts;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

```
sp [promptselect]
    [;keys]
    [;offline]
```

```
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 254.

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),
(0AAAAAAAAAAAABST)]) 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),(0AAAAAAAAAAAAABTR)])  
Are you ready to process job3?
```

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**

For the equivalent Job Scheduling Console task, see "Displaying and Modifying Prompt Properties" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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

```
sr [[workstation#]resourcename]
    [:keys]
    [:offline]
```

```
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 254.

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:

```
Workstation Job Stream SchedTime Job State Pr Start Elapse RetCode Dependencies
              (Est) (Est)
```

```
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 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**

For the equivalent Job Scheduling Console task, see "Displaying Resource Properties" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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.

Depsjinfo 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.

Depsjlogon 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

```
ss [jstreamselect]
    [;keys]
    [;offline]
    [;showid]
```

```
ss [jstreamselect]
    [;deps[;keys | info | logon]]
    [;offline]
    [;showid]
```

Arguments

jstreamselect

See “Selecting job streams in commands” on page 267.

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 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 254.

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:

Workstation	Job Stream	SchedTime	State	Pr	(Est) Start	(Est) Elapse	Jobs #	Sch OK	Lim	
site3	#JS_DOCOM	0600 11/26	SUCC	10	11/26 00:01	1	1			{0AAAAAAAAAAAAACRZ}

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:

Workstation	Job Stream	SchedTime	State	Pr	(Est) Start	(Est) Elapse	Jobs #	Sch 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),(0AAAAAAAAAAAAABVY)] Dependencies are:
CPUA#MOD 0212 03/10                JOBMDM /usr/scripts/g11(B236153+#)JOB_FTA1  echo Start g11?
CPUA#MOD 0251 03/10                JOBMDM /usr/scripts/g12(B236153+#)JOB_FTA2  echo Start g12?
```

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:

Workstation	Job Stream	SchedTime	State	Pr	(Est) Start	(Est) Elapse	Jobs #	Sch 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 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**

For the equivalent Job Scheduling Console task, see "Displaying a Job Stream" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 @#@
CPU      Schedule SchedTime State Pr Start (Est) (Est) Jobs Sch
MASTERB1#JS_B 1800 08/18 HOLD 10(18:00) Elapse # OK Lim
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 @#@
CPU      Schedule SchedTime State Pr Start (Est) (Est) Jobs Sch
MASTERB1#JS_A 0500 08/19 HOLD 10(08/19) Elapse # OK Lim
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**, all **mailman** servers, and all **writer** processes.

It does not stop the WebSphere Application Server services.

You must have *shutdown* access to the workstation.

Syntax

```
shut[down] [;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 405. 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 348 and “startmon” on page 349 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.

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 369 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 and Table 51 below show the workstations started by **start** commands run by users in various locations in the network.

DM_n are domain managers and **A_{mn}** are agents.

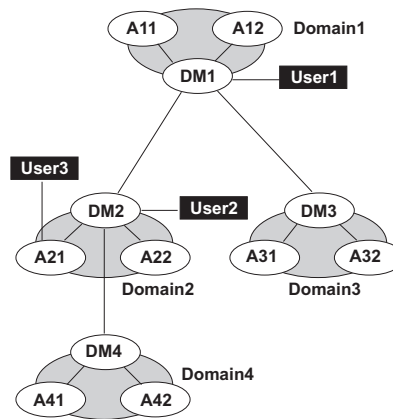


Figure 23. Started workstations in network

Table 51. 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 [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 [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
```

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.

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

“stop” on page 350 and Table 52 on page 352 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_{mn}** are agents.

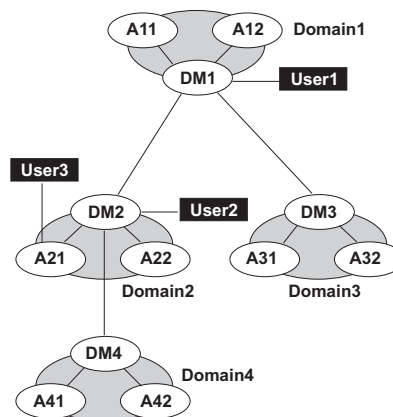


Figure 24. Stopped workstations in network

Table 52. 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
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.

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

“stop” on page 350 and Table 53 on page 353 show the workstations stopped by issuing the **stop ;progressive** command on DM2.

Table 53. 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 [*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.

1 When you run the command, the *appserverman* process first checks if WebSphere
1 Application Server can retrieve the user's credentials (username and password)
1 from the soap.client.props file located in the WebSphere Application Server
1 profile. If the check is negative, *appserverman* reads them from the useropts file of
1 the user and runs the stopServer.sh (bat) script to pass them to WebSphere
1 Application Server.

1 To be able to run the command, you must therefore complete one of the following
1 two customization procedures to provide the user credentials to WebSphere
1 Application Server:

- 1 • Customize the user name (com.ibm.SOAP.loginUserId) and password
1 (com.ibm.SOAP.loginPassword) properties in the soap.client.props file located
1 in:

1 *TWS_home/appserver/profiles/twsprofile/properties/* (Version 8.4 and earlier master)
1 *TWS_home/appserver/profiles/twsconnprofile/properties/* (Version 8.4 and earlier agents)
1 *TWA_home/eWAS/profiles/twaprofile/properties/* (Version 8.5 and up master and agents)

1 You must also:

- 1 1. Set property `com.ibm.SOAP.securityEnabled` to true in the same file to enable
1 the SOAP client security
 - 1 2. Run the `encryptProfileProperties.sh` script to encrypt the password. See the
1 Tivoli Workload Scheduler *Administration Guide* for more information on this
1 application server tool.
- 1 • Customize the Attributes for conman (CLI in version 8.4) connections
1 section in the `localopts` file by specifying the details of the connector or of the
1 master domain manager.

1 You must also:

- 1 1. Create (or customize if already present) the `useropts` file manually, adding
1 the `USERNAME` and `PASSWORD` attributes for the user who will run
1 `stopappserver`. Make sure the `useropts` file name is entered in the `USEROPTS`
1 key in the Attributes for conman (CLI) connections section. See the
1 *Administration Guide* for further details.
- 1 2. Encrypt the password in the `useropts` file simply by running `conman`.

stopeventprocessor

Stops the event processing server.

Syntax

stopeventprocessor [*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 [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 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 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

```
sbid [workstation#]"cmd"  
      [;alias[=name]]  
      [;into=[workstation#]  
      {jobstream_id;schedid | jobstreamname (hmmm[ 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:

```
%sbid "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*[,...]

rcondsucc"*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.

1 The scheduler classifies follows dependencies as *internal* when they are specified
 1 only by their job name within the job stream. It classifies them as *external* when
 1 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 afile";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

```
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 jcl1ttx5, 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.

1 The scheduler classifies `follows` dependencies as *internal* when they are specified
1 only by their job name within the job stream. It classifies them as *external* when
1 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 `myjc1`), run the following command:

```
submit file=d:\jobs\lib\daily\myjc1
```

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/jc14;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 `twc_env.cmd`, whose path contains a blank, on a Windows workstation run:

- In interactive mode:

```
sbf "\"C:\Program Files\IBM\TWS\lucaMDM\twc_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\twc_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.

1 To run this command, in the security file you must have **submit** (**submitdb**) access
1 for the job with the name specified in its database definition and, if you use the
1 **alias** keyword, also with the name specified with this keyword. In addition, if you
1 use the **recoveryjob** keyword, you must have **submit** access for the job specified
1 with that keyword.

1 Note that if you have security **submitdb** rights only, you are limited to submit jobs
1 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

Syntax

```
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.

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.

1 The scheduler classifies follows dependencies as *internal* when they are specified
1 only by their job name within the job stream. It classifies them as *external* when
1 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**.

For the equivalent Job Scheduling Console task, see "Submitting Jobs" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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

```
sbs [workstation#] jstreamname  
    [;alias[=name]]  
    [;jstreamoption[;...]]  
    [;variable=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 157 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. 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.

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.

1 The scheduler classifies follows dependencies as *internal* when they are specified
1 only by their job name within the job stream. It classifies them as *external* when
1 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

For the equivalent Job Scheduling Console task, see "Managing distributed plans" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

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 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 *IBM Tivoli Workload Scheduler: Administration Guide*.

Syntax

switchmgr *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.

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

Runs a system command.

Syntax

```
[[: | !] sys-command
```

Arguments

sys-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

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 287.

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
TELOP>*****
TELOP>* TWS will be stopped at *
TELOP>* 4:30 for 15 minutes. *
TELOP>*****
TELOP>//
```

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.

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 299.

Examples

Figure 25 on page 373 and Table 54 on page 373 show the links closed by **unlink** commands run by users in various locations in the network.

DM_n are domain managers and **Ann** are agents.

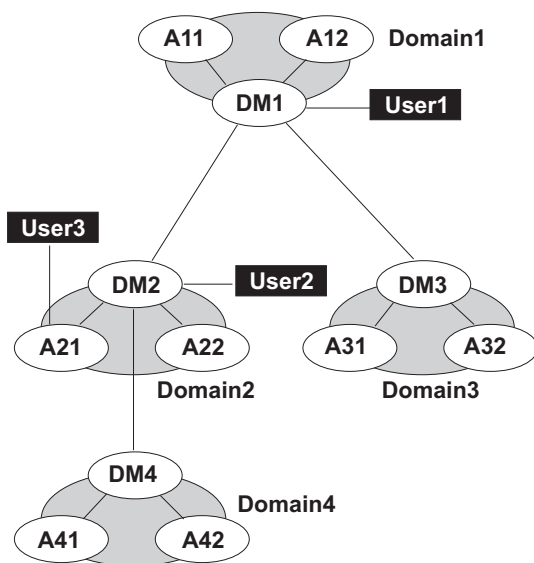


Figure 25. Unlinked network workstations

Table 54. Unlinked workstations

Command	Closed by User1	Closed by User2	Closed by User3
<code>unlink@!@</code>	All links are closed	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4 DM4-A41 DM4-A42	DM2-A21
<code>unlink @</code>	DM1-A11 DM1-A12 DM1-DM2 DM1-DM3	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4	DM2-A21
<code>unlink DOMAIN3!@</code>	DM3-A31 DM3-A32	Not allowed	Not allowed
<code>unlink DOMAIN4!@</code>	DM4-A41 DM4-A42	DM4-A41 DM4-A42	Not allowed
<code>unlink DM2</code>	DM1-DM2	Not applicable	DM2-A21
<code>unlink A42</code>	DM4-A42	DM4-A42	Not allowed
<code>unlink A31</code>	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

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. Using utility commands

This chapter describes Tivoli Workload Scheduler utility commands. These commands, with the exception of **StartUp** and **version**, are installed in the *TWS_home/bin* directory. **StartUp** is installed in the *TWS_home* directory and **version** is installed in the *TWS_home/version* directory. You run utility commands from the operating system command prompt.

Command descriptions

Table 55 contains the list of the utility commands, and for each command, its description and the operating systems it supports.

Table 55. 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
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
metronome.pl	Is replaced by tws_inst_pull_info .	UNIX, Windows
morestdl	Displays the contents of standard list files.	UNIX, Windows
parms	Displays, changes, and adds parameters.	UNIX, Windows
release	Releases units of a resource.	UNIX, Windows

1
1

Table 55. List of utility commands (continued)

Command	Description	Operating system
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
StartUp	Starts the netman process and, optionally, WebSphere Application Server.	UNIX, Windows
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 <i>Tivoli Workload Scheduler: Troubleshooting Guide</i> .	UNIX, Windows
version	Displays version information.	UNIX

1
1
1
1
1

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 378 for more information about job streams.

The following keywords apply only to **at** jobs:

-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 378 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.”

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 37.

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 files are created by Tivoli Workload Scheduler using the following naming convention:

```
TWS_home/at.jobs/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 `at.jobs` 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**, and **OTHER**.

node Returns the value of the **node** field.

port Returns the value of the **port** field.

autolink

Returns the value of the **autolink** field: ON or OFF.

fullstatus

Returns the value of the **fullstatus** field: ON or OFF.

host Returns the value of the **host** field.

method

Returns the value of the **access** field.

server Returns the value of the **server** field.

type Returns the type of workstation: **MASTER**, **MANAGER**, **FTA**, **S-AGENT**, and **X-AGENT**.

time_zone

Returns the time zone of the workstation. For an extended agent, the field is blank.

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 an extended agent, the field is blank.

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
oak_wks	manager	MASTERDM	03/04/2006	-

```
CPUNAME oak_wks
DESCRIPTION "MASTER CPU"
OS UNIX
NODE oak.tivoli.com TCPADDR 51099
DOMAIN MASTERDM
FOR MAESTRO
TYPE MANAGER
AUTOLINK ON
BEHINDFIREWALL OFF
FULLSTATUS ON
END
```

To print the **node** and **port** for workstation oak, run the following command:

```
>cpuinfo oak_wks node port
oak.tivoli.com
51099
```

To print all information for workstation oak, run the following command:

```
>cpuinfo oak_wks
OS TYPE:UNIX
NODE:oak.tivoli.com
PORT:51099
SSLPORT: 0
SEC_LEVEL: NONE
AUTOLINK:ON
FULLSTATUS:ON
RESOLVEDEP: ON
BEHINDFIREWALL: OFF
HOST:oak_wks
DOMAIN: MASTERDM
METHOD:
SERVER:
TYPE: MASTER
TIME_ZONE:Europe/Rome
VERSION:8.3
INFO: Linux 2.4.21-4.ELsmp #1 SMP Fri 0 i686
```

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]*.

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.

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.

yyyymmddhhmm

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 37.

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

Represents the set of parameters that rule the interaction between the

command line 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 with HTTP.

proxy_port_number

Is the proxy port number used in the connection with HTTP.

user_password

Is the password of the user running the command.

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 user ID of the user running the command.

If any of these parameters is omitted when invoking the command, Tivoli Workload Scheduler searches for the value first in the `useropts` file and then in the `localopts` file. If a setting for the parameter is not specified either in the connection parameters when invoking the command or inside the `useropts` and `localopts` files then an error is displayed. Refer to "Setting up options for using the user interfaces" on page 41 for information on `useropts` and `localopts` files.

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 402 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.xml`
`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]

1 **evtsize -info filename**

evtsize -show filename

1 **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.

1 **-info filename**

1 Displays the percentage use of the queue of messages contained in the
1 message file.

-show filename

Displays the size of the queue of messages contained in the message file.

filename

The name of the message file. Specify one of the following:

Courier.msg
Intercom.msg
Mailbox.msg
pobox/workstation.msg

1 *size* The maximum size of the message file in bytes. It must be no less than
1 1048576 bytes (1 MB).

1 When first built by Tivoli Workload Scheduler, the maximum size is set to
1 10 MB.

1 **Note:** The size of the message file is equal to or bigger than the real size of
1 the queue of messages it contains and it progressively increases until
1 the queue of messages becomes empty; as this occurs the message
1 file is emptied.

1 **-info | -show pobox**
1 Displays the name of the message file, within the pobox directory, with the
1 largest queue size calculated as a percentage of the total file size. Both the
1 name of the file and the percentage used are returned. Either **-info** and
1 **-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

1 If the following command:

```
1           evtsize -info Mailbox.msg
```

1 returns:

```
1           25
```

1 it means that 25 percent of the file has been used.

jobinfo

Used in a job script to return information about the job.

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_recode

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
...

```

- 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
```

```

1 Returns the standard list file name for job1 for the instance of job_stream1 with ID
1 0AAAAAAAAAAAAAB5.
1 jobstdl -name "job_stream1(0600 04/05/10).job1"

1 Returns the standard list file names for job1 for all possible instances of
1 job_stream1 scheduled to run at 0600 of 04/05/10.
1 jobstdl -name job_stream1(0600).job1

1 Returns the standard list file names for job1 for all possible instances of
1 job_stream1 scheduled to run at 0600 of the current day.
1 jobstdl -name 310

1 Returns the standard list file names for job1 for all the instances it had job number
1 310.

```

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 *Schedule*) 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: *Chhmm*, 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 381 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

1 Metronome is replaced by **twins_inst_pull_info**. See *IBM Tivoli Workload Scheduler*
1 *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.

Syntax

```
morestdl -V | -U
```

```
morestdl
```

```
[-day num]  
[-first | -last | -num n | -all]  
[-twslog]  
[{-name ["jobstreamname [(hmm 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),(0AAAAAAAAAAAAAEE)].DIR"
```

To display 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:

```
morestdl -day 3 -name "MY_CPU#ELI[(1824 03/09/06),(0AAAAAAAAAAAAAEE)].DIR"
```

To display the standard list file for the last run of job MY_CPU#ELI[(1824 03/09/06), (0AAAAAAAAAAAAAEE)].DIR from four days ago, run the following command:

```
morestdl -day 4 -last -name "MY_CPU#ELI[(1824 03/09/06), (0AAAAAAAAAAAAAEE)].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 213 or the “replace” on page 247 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 “create - extract” on page 216 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 remote command line.

-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 remote command line.

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 Administrators 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
```

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 345 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
```

1 tws_inst_pull_info

1 This is a script that produces information about your Tivoli Workload Scheduler
1 environment and your local workstation, and can take a snapshot of DB2 and
1 WebSphere Application Server data on the master domain manager, saving them as
1 a dated package.

1 It can also generate a report containing not only the results of the snapshot, but
1 also many configuration and environment parameters. The tool is useful when
1 describing a problem to IBM Software Support. For best results, it must be run as
1 soon as the problem is discovered.

1 Comments

1 Refer to *IBM Tivoli Workload Scheduler Troubleshooting Guide* for more information
1 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 command version 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 should not be 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 located 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 12. 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 410
- “Sample report outputs” on page 417
- “Report extract programs” on page 427
- “Running reports on the Tivoli Dynamic Workload Console” on page 437

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:

MAESTROL

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. Set the variable to **LONG** to use full length (long) fields for object names.

If it is not set or is set to anything other than **LONG**, long names are truncated to eight characters and a plus sign. For example: **A1234567+**.

You are suggested to use a fixed font size for correctly managing reports output formatting.

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 56. Date formats

<i>date format</i> 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 57:

Table 57. 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.

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.

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
```

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 report does not include jobs that were submitted using an alias name.

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

The report does not include jobs that were submitted using an alias name.

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 *MAESTROL* variable. If *MAESTROL* 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        : ^ACCLLOGIN^
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)

```

* * * * End of Report * * * *

In the output you see the values set in the “Job” on page 535 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).

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)
Report 02

ibm
Prompt Message Listing

Page 1
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

* * * * End of Report * * * *

GOODEXIT 0
SCRPATH /usr/local/Tivoli/maestro_admin/scripts

Number of Parameters on file: 5

***** End of Report *****

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)
Report 4B

ibm
TWS Resources Listing

Page 1
03/06/06

CPU	Resource Name	Number 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

***** End of Report *****

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)
Report 07

ibm
Job History Listing

Page 1
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

***** End of Report *****

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	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 .*

*** End of Report ***

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				
NEWTST	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					

*** End of Report ***

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

	Job Name	Estimated Run Time	Priority	Start Time	Actual Run Time	CPU Seconds	Job Number	Status
Schedule	NETAG #EXTERNAL E0000000		0					EXTRN ERROR
	Total	00:00			00:00	0		
Schedule	MYMST #MONTHSKE GETLOGS	00:02 00:02	10 10	06:01(03/06/06) 06:01(03/06/06)	00:03 00:03		#J11612	SUCC SUCC
	Total	00:02			00:03	0		
Schedule	MYFTA #IWSKE JOBIWD		10 10					HOLD HOLD
	Total	00:00			00:00	0		
Schedule	MYMST #TESTSKE TESTCRO+ NEWTST	00:29 00:01 00:29	10 10 10	06:01(03/06/06) 06:01(03/06/06)	00:02 00:02		#J11613	STUCK ABEND HOLD
	Total	00:30			00:02	0		
Schedule	MYMST #FINAL JnextPlan	00:01 00:01	10 10	05:59(03/07/06)				HOLD HOLD
	Total	00:01			00:00	0		
	Total	01:38			00:09	0		

* * * * End of Report * * * *

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)

Report 11

Planned Production Schedule for FEB 2006

Page 1

03/08/065

CPU: FTAWIN8+

Schedule	Num Jobs	Est Cpu Time	1 Tu	2 We	3 Th	4 Fr	5 Sa	6 Su	7 Mo	8 Tu	9 We	10 Th	11 Fr	12 Sa	13 Su	14 Mo	15 Tu	16 We	17 Th	18 Fr	19 Sa	20 Su	21 Mo	22 Tu	23 We	24 Th	25 Fr	26 Sa	27 Su	28 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 0	2 0	3 0	4 0	5 0	6 0
7 0	8 0	9 0	10 0	11 1	12 0	13 0
14 0	15 0	16 0	17 0	18 0	19 0	20 0
21 0	22 0	23 0	24 0	25 0	26 0	27 0
28 0						

TWS for UNIX (AIX)/REPORT11 8.3 (1.7)

Report 11

Planned Production Schedule for FEB 2006

Page 2

03/08/06

CPU: MASTER8+

Schedule	Num Jobs	Est Cpu Time	1 Tu	2 We	3 Th	4 Fr	5 Sa	6 Su	7 Mo	8 Tu	9 We	10 Th	11 Fr	12 Sa	13 Su	14 Mo	15 Tu	16 We	17 Th	18 Fr	19 Sa	20 Su	21 Mo	22 Tu	23 We	24 Th	25 Fr	26 Sa	27 Su	28 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 4	2 4	3 4	4 4	5 4	6 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						

* * * * End of Report * * * *

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

* * * * End of Report * * * *

User defined text TMP

* * * * End 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

* * * * End 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 58:

Table 58. Report extract programs.

Report extract program	Description
jbextract	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 410.

jbextract

Extracts information about jobs from the database.

Syntax

```
jbextract [-V | -U]
           [-j job]
           [-c wkstat]
           [-f date -t date]
           [-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.
- 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.
- o *output*
Specifies the output file. The default is **stdout**.

Results

The *MAESTRO_OUTPUT_STYLE* variable specifies the output style for long object names. Set the variable to **LONG** to use full length (long) fields for object names. If the variable is not set, or 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 59.

Table 59. *Jbextract* 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
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

Table 59. Jbextract output fields (continued)

Field	Description	Max Length (bytes)
25	minimum run date (yymmdd)	8

Examples

To extract information about job myjob on workstation main and direct the output to the file jinfo, run the following command:

```
jextract -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 60.

Table 60. 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 prininfo, run the following command:

```
prxtract -o prininfo
```

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 61.

Table 61. *Cextract 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:

```
cextract -o cainfo
```

pextract

Extracts information about global parameters (variables) from the database.

Syntax

```
pextract [-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 62.

Table 62. *Pextract 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:

```
pextract -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 63.

Table 63. 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. Set the variable to **LONG** to use full length (long) fields for object names. If the variable is not set, or 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 64.

Table 64. 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
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

`xrxtct [-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. Set the variable to `LONG` to use full length (long) fields for object names. If the variable is not set, or 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:

```
xrxtct
```

`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 65.

Table 65. *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 66.

Table 66. *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 67.

Table 67. *Xdep_sched* output fields

Field	Description	Length (bytes)
1	02	2
2	workstation name	16
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 68 on page 435.

Table 68. 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 69.

Table 69. 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
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 70.

Table 70. Xprompts output fields

Field	Description	Length (bytes)
1	05	2
2	workstation name	16

Table 70. Xprompts output fields (continued)

Field	Description	Length (bytes)
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 71.

Table 71. 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
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 72.

Table 72. Xsched output fields

Field	Description	Length (bytes)
1	00	2
2	workstation name	16
3	job stream name	16
4	not used	248

Table 72. Xsched output fields (continued)

Field	Description	Length (bytes)
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 73.

Table 73. 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
13	end-of-record (null)	1

1

Running reports on the Tivoli Dynamic Workload Console

These reports can be run only from the Tivoli Dynamic Workload Console. This section provides an overview. Refer to the Tivoli Dynamic Workload Console online help for details.

See also the *Administration Guide* to learn how to configure the Tivoli Dynamic Workload Console to view reports.

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 the *Administration Guide* for security file information.

Historical reports

The following table summarizes the historical reports in terms of their:

- Functionality
- Selection criteria
- Output content options

Table 74. Summary of historical reports

Report name	Description	Selection criteria	Output content options
Job run history	<p>Corresponds to Report 07.</p> <p>Collects historical job execution data during a time interval. Helps you find:</p> <ul style="list-style-type: none"> • Jobs ended in error • Late jobs • Missed deadlines • Long duration • Rerun indicators for reruns • Other historical information. <p>Note: The report does not include jobs that were submitted using an alias name.</p>	<ul style="list-style-type: none"> • Job name, job stream name, workstation name, and workstation name (job stream). Each field can be specified using a wildcard and commas to add more values. • Status (Success, Error, Canceled, Unknown) and external status. • Delay indicators • Job execution interval • Include/Exclude rerun iterations 	<p>You can select from the following:</p> <ul style="list-style-type: none"> • Job name • Workstation (job) • Job stream name • Workstation (job stream) • Scheduled time • Actual start time • Estimated duration • Actual duration • Job number • Started late (delay) • Ended late (delay) • Status • Return code • CPU consumption (not available on Windows workstations) • Logon user • Rerun type • Iteration number • Return code <p>The output is in table view.</p>

Table 74. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
Job run statistics	<p>Corresponds to Report 01.</p> <p>Collects job execution statistics. Helps you find:</p> <ul style="list-style-type: none"> • Success/error rates • Minimum and maximum elapsed and CPU times • Average duration • Late and long duration statistics <p>Note: The report does not include jobs that were submitted using an alias name.</p>	<ul style="list-style-type: none"> • Job name, workstation name, and user login. Each field can be specified using a wildcard and commas to add more values. • Percentage of jobs in Success, Error, Started late, Ended late, and Long duration • Total runs and total reruns 	<p>You can select from the following:</p> <ul style="list-style-type: none"> • Job details: <ul style="list-style-type: none"> – Job name – Workstation name – Logon user – Job creator – Description – Script • Job statistics: <ul style="list-style-type: none"> – Total runs (divided in Successful and Error) – Total of runtime exceptions (Started late, Ended late, Long duration) – Minimum and maximum elapsed and CPU times (for successful runs only) – Average duration (for successful runs only) – CPU consumption (not available on Windows workstations) • Report format: <ul style="list-style-type: none"> – Charts view (pie, bar) – Table view – Include table of contents by job or by workstation

Table 74. 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).	<ul style="list-style-type: none"> • Workstation names. Each field can be specified using a wildcard and commas to add more values. • 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) 	<p>You can select from the following:</p> <ul style="list-style-type: none"> • Workstation information granularity arranged by: <ul style="list-style-type: none"> – Hour – Day – Production day • Information aggregation options: <ul style="list-style-type: none"> – Provide cumulative and aggregated workstation summary information for all or a desired subset of workstations • Report format: <ul style="list-style-type: none"> – Charts view (bar, line) – Table view – Include table of contents by date or by workstation

Table 74. Summary of historical reports (continued)

Report name	Description	Selection criteria	Output content options
Workstation workload runtimes	<p>Corresponds to Report 08.</p> <p>Provides data on the job runs (time and duration) on the workstations. Helps making the necessary capacity planning adjustments (workload modeling, and workstation tuning).</p> <p>Note: The report does not include jobs that were submitted using an alias name.</p>	<ul style="list-style-type: none"> • Job and workstation names. Each field can be specified using a wildcard and commas to add more values. • Job execution interval • Daily time intervals 	<p>You can select from the following:</p> <ul style="list-style-type: none"> • Information grouped by: <ul style="list-style-type: none"> – Workstation – Run date – Production day Can be ordered by rerun iteration • Job information: <ul style="list-style-type: none"> – Job name – Workstation name – Workstation name (Job stream) – Job stream name – Scheduled time – Actual start time – Actual duration – Status – Rerun iteration • Report format: <ul style="list-style-type: none"> – Charts view (GANNT) – Table view – Include table of contents – Limit the number of jobs in each single chart
Custom SQL	<p>A wizard helps you define your custom SQL query (only on the database views which you are authorized to access). The resulting report has a table with the column name as specified in the SELECT part of the SQL statement.</p>	-	-

All reports are created in HTML (PDF) or CSV format. The heading of the output contains the SQL statement. The body contains the report charts in the selected form.

Production reports

The following table summarizes the production reports in terms of their:

- Functionality

- Selection criteria
- Output content options

Table 75. Summary of production reports

Report name	Description	Selection criteria	Output content options
Actual production details	<p>Corresponds to Report 10B.</p> <p>Provides data on current and archived plans.</p>	<ul style="list-style-type: none"> • Job stream name and workstation name (job stream). • Job name, workstation name (job), job stream name (job), job stream workstation (job) • Priority 	<p>You can select from the following:</p> <ul style="list-style-type: none"> • Job stream information: <ul style="list-style-type: none"> - Job stream name - Workstation name - Priority - Limit - Total jobs - Status - Internal status - Scheduled start time - Latest start time - Actual start time - Estimated deadline time - Estimated duration - Actual duration - CPU time • Job information: <ul style="list-style-type: none"> - Job name - Workstation name - Priority - Status - Internal status - Is every job - Is rerun job - CPU time - Job log - Estimated start time - Latest start time - Actual start time - Estimated deadline time - Estimated duration - Actual duration - Rerun code - Job number - Dependencies • Report format: <ul style="list-style-type: none"> - Flat format - CSV separator

Table 75. Summary of production reports (continued)

Report name	Description	Selection criteria	Output content options
Planned production details	<p>Corresponds to Report 9B.</p> <p>Provides data on trial and forecast plans.</p>	<ul style="list-style-type: none"> • Job stream name and workstation name (job stream). • Job name, workstation name (job), job stream name (job), job stream workstation (job) • Priority 	<p>You can select from the following:</p> <ul style="list-style-type: none"> • Job Stream information: <ul style="list-style-type: none"> - Job stream name - Workstation name - Priority - Limit - Total jobs - Scheduled start time - Latest start time - Estimated deadline time • Job information: <ul style="list-style-type: none"> - Job name - Workstation name - Priority - Is every job - Scheduled start time - Latest start time - Estimated deadline time - Estimated duration - Dependencies • Report format: <ul style="list-style-type: none"> - Flat format - CSV separator

Chapter 13. 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).

At the end of this chapter are two sections each containing a time zone list table. The first shows for each 3-character format time zone name that was already in use and the corresponding default long name notation assigned. The second contains the complete set of time zones with long name notation.

The chapter is divided in the following sections:

- “Enabling time zone management”
- “How Tivoli Workload Scheduler manages time zones” on page 446
- “Moving to daylight saving time on” on page 448
- “Moving to daylight saving time off” on page 448
- “General rules” on page 448
- “Backward compatibility time zone table” on page 449
- “Complete table of time zones with variable length notation” on page 450

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 26 on page 447 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 26 on page 447.

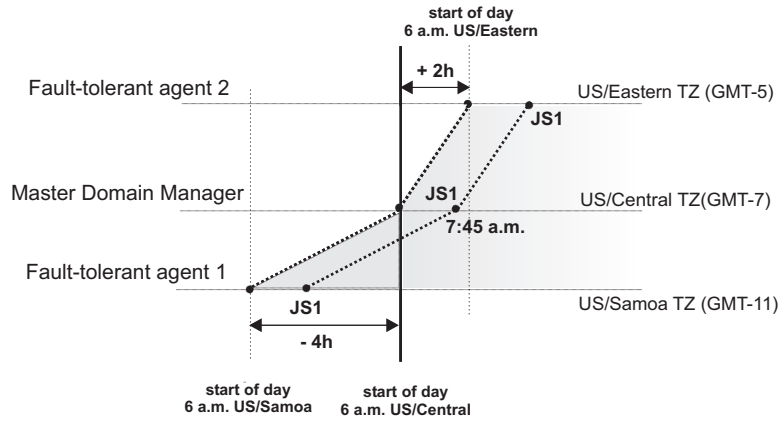


Figure 26. Example when start of day conversion is not applied

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 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. 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 27.

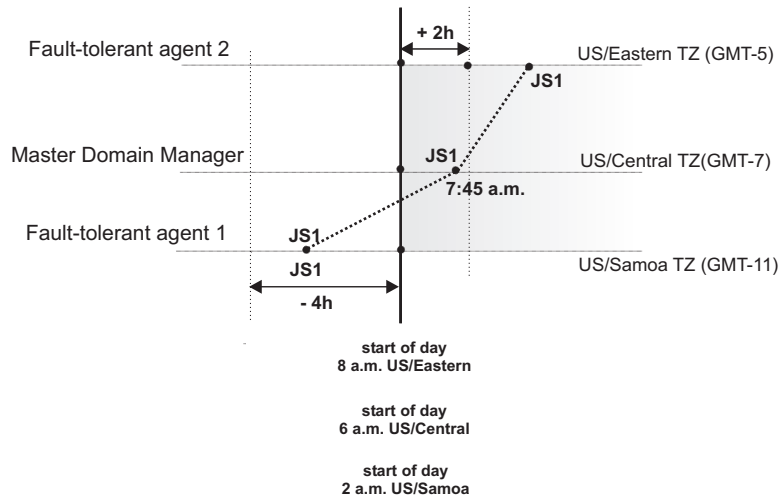


Figure 27. Example when start of day conversion is applied

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 at 3:00. The reason for doing this is that at 2:00 the clock time is moved one hour ahead because DST is switched on, and so all job streams planned to start between 2:00 and 2:59, if free from dependencies, start immediately because 3:00 is later than their scheduled start time.

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.

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.

Backward compatibility time zone table

Table 76 shows for each 3-character format time zone that was in use with the previous versions of the product, the default long name notation assigned.

The time zone name is case sensitive.

Table 76. Backward compatibility time zone table

Long Name	Short Name	Description	Relative to GMT
GMT	GMT	Greenwich Mean Time	GMT
UTC	UTC	Universal Coordinated Time	GMT
Europe/Paris	ECT	European Central Time	GMT+1:00
Europe/Istanbul	EET	Eastern European Time	GMT+2:00
Africa/Cairo	ART	(Arabic) Egypt Standard Time	GMT+2:00
Asia/Riyadh	EAT	Eastern African Time	GMT+3:00
Europe/Paris	MET	Middle Europe Time	GMT+1:00
Asia/Yerevan	NET	Near East Time	GMT+4:00
Asia/Karachi	PLT	Pakistan Lahore Time	GMT+5:00
Asia/Calcutta	IST	India Standard Time	GMT+5:30
Asia/Dacca	BST	Bangladesh Standard Time	GMT+6:00
Asia/Bangkok	VST	Vietnam Standard Time	GMT+7:00

Table 76. Backward compatibility time zone table (continued)

Long Name	Short Name	Description	Relative to GMT
Asia/Shanghai	CTT	China Taiwan Time	GMT+8:00
Asia/Tokyo	JST	Japan Standard Time	GMT+9:00
Australia/Darwin	ACT	Australia Central Time	GMT+9:30
Australia/Sydney	AET	Australia Eastern Time	GMT+10:00
Pacific/Guadalcanal	SST	Solomon Standard Time	GMT+11:00
Pacific/Fiji	NST	New Zealand Standard Time	GMT+12:00
Pacific/Apia	MIT	Midway Islands Time	GMT-11:00
Pacific/Honolulu	HST	Hawaii Standard Time	GMT-10:00
America/Anchorage	AST	Alaska Standard Time	GMT-9:00
America/Los_Angeles	PST	Pacific Standard Time	GMT-8:00
America/Phoenix	PNT	Phoenix Standard Time	GMT-7:00
America/Denver	MST	Mountain Standard Time	GMT-7:00
America/Chicago	CST	Central Standard Time	GMT-6:00
America/New_York	EST	Eastern Standard Time	GMT-5:00
America/Indianapolis	IET	Indiana Eastern Standard Time	GMT-5:00
America/Caracas	PRT	Puerto Rico and US Virgin Islands Time	GMT-4:00
America/St_Johns	CNT	Canada Newfoundland Time	GMT-3:30
America/Buenos_Aires	AGT	Argentina Standard Time	GMT-3:00
America/Sao_Paulo	BET	Brazil Eastern Time	GMT-3:00
Atlantic/Cape_Verde	CAT	Central African Time	GMT-1:00

Complete table of time zones with variable length notation

Table 77 shows all the supported time zones expressed with variable length notation, their descriptions, and their offset with respect to GMT.

The new time zone labels that can be used with this version of the product are shown with revision bars. Note that :

- Time zone names are case sensitive.
- The SystemV time zones are no longer supported.
- The new names are not supported by earlier Tivoli Workload Scheduler versions than 8.4.

Table 77. Variable length notation time zones list

Long Name	Description	Relative to GMT
ACT	Central Standard Time (Northern Territory)	GMT+9
AET	Eastern Standard Time (New South Wales)	GMT+11
Africa/Abidjan	Greenwich Mean Time	GMT
Africa/Accra	Greenwich Mean Time	GMT
Africa/Addis_Ababa	Eastern African Time	GMT+3

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Africa/Algiers	Central European Time	GMT+1
Africa/Asmara	Eastern African Time	GMT+3
Africa/Asmera	Eastern African Time	GMT+3
Africa/Bamako	Greenwich Mean Time	GMT
Africa/Bangui	Western African Time	GMT+1
Africa/Banjul	Greenwich Mean Time	GMT
Africa/Bissau	Greenwich Mean Time	GMT
Africa/Blantyre	Central African Time	GMT+2
Africa/Brazzaville	Western African Time	GMT+1
Africa/Bujumbura	Central African Time	GMT+2
Africa/Cairo	Eastern European Time	GMT+2
Africa/Casablanca	Western European Time	GMT
Africa/Ceuta	Central European Time	GMT+1
Africa/Conakry	Greenwich Mean Time	GMT
Africa/Dakar	Greenwich Mean Time	GMT
Africa/Dar_es_Salaam	Eastern African Time	GMT+3
Africa/Djibouti	Eastern African Time	GMT+3
Africa/Douala	Western African Time	GMT+1
Africa/El_Aaiun	Western European Time	GMT
Africa/Freetown	Greenwich Mean Time	GMT
Africa/Gaborone	Central African Time	GMT+2
Africa/Harare	Central African Time	GMT+2
Africa/Johannesburg	South Africa Standard Time	GMT+2
Africa/Kampala	Eastern African Time	GMT+3
Africa/Khartoum	Eastern African Time	GMT+3
Africa/Kigali	Central African Time	GMT+2
Africa/Kinshasa	Western African Time	GMT+1
Africa/Lagos	Western African Time	GMT+1
Africa/Libreville	Western African Time	GMT+1
Africa/Lome	Greenwich Mean Time	GMT
Africa/Luanda	Western African Time	GMT+1
Africa/Lubumbashi	Central African Time	GMT+2
Africa/Lusaka	Central African Time	GMT+2
Africa/Malabo	Western African Time	GMT+1
Africa/Maputo	Central African Time	GMT+2
Africa/Maseru	South Africa Standard Time	GMT+2
Africa/Mbabane	South Africa Standard Time	GMT+2
Africa/Mogadishu	Eastern African Time	GMT+3
Africa/Monrovia	Greenwich Mean Time	GMT

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Africa/Nairobi	Eastern African Time	GMT+3
Africa/Ndjamena	Western African Time	GMT+1
Africa/Niamey	Western African Time	GMT+1
Africa/Nouakchott	Greenwich Mean Time	GMT
Africa/Ouagadougou	Greenwich Mean Time	GMT
Africa/Porto-Novo	Western African Time	GMT+1
Africa/Sao_Tome	Greenwich Mean Time	GMT
Africa/Timbuktu	Greenwich Mean Time	GMT
Africa/Tripoli	Eastern European Time	GMT+2
Africa/Tunis	Central European Time	GMT+1
Africa/Windhoek	Western African Time	GMT+2
AGT	Argentine Time	GMT-3
America/Adak	Hawaii-Aleutian Standard Time	GMT-10
America/Anchorage	Alaska Standard Time	GMT-9
America/Anguilla	Atlantic Standard Time	GMT-4
America/Antigua	Atlantic Standard Time	GMT-4
America/Araguaina	Brazil Time	GMT-2
America/Argentina/ Buenos_Aires	Argentine Time	GMT-3
America/Argentina/ Catamarca	Argentine Time	GMT-3
America/Argentina/ ComodRivadavia	Argentine Time	GMT-3
America/Argentina/ Cordoba	Argentine Time	GMT-3
America/Argentina/ Jujuy	Argentine Time	GMT-3
America/Argentina/ La_Rioja	Argentine Time	GMT-3
America/Argentina/ Mendoza	Argentine Time	GMT-3
America/Argentina/ Rio_Gallegos	Argentine Time	GMT-3
America/Argentina/ San_Juan	Argentine Time	GMT-3
America/Argentina/ San_Luis	Argentine Time	GMT-3
America/Argentina/ Tucuman	Argentine Time	GMT-3
America/Argentina/ Ushuaia	Argentine Time	GMT-3
America/Aruba	Atlantic Standard Time	GMT-4
America/Asuncion	Paraguay Time	GMT-3

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
America/Atikokan	Eastern Standard Time	GMT-5
America/Atka	Hawaii-Aleutian Standard Time	GMT-10
America/Bahia	Brazil Time	GMT-3
America/Barbados	Atlantic Standard Time	GMT-4
America/Belem	Brazil Time	GMT-3
America/Belize	Central Standard Time	GMT-6
America/Blanc-Sablon	Eastern Standard Time	GMT-5
America/Boa_Vista	Amazon Standard Time	GMT-4
America/Bogota	Colombia Time	GMT-5
America/Boise	Mountain Standard Time	GMT-7
America/Buenos_Aires	Argentine Time	GMT-3
America/Cambridge_Bay	Mountain Standard Time	GMT-7
America/Cancun	Central Standard Time	GMT-6
America/Caracas	Venezuela Time	GMT-4
America/Catamarca	Argentine Time	GMT-3
America/Cayenne	French Guiana Time	GMT-3
America/Cayman	Eastern Standard Time	GMT-5
America/Chicago	Central Standard Time	GMT-6
America/Chihuahua	Mountain Standard Time	GMT-7
America/Cordoba	Argentine Time	GMT-3
America/Coral_Harbour	Eastern Standard Time	GMT-5
America/Costa_Rica	Central Standard Time	GMT-6
America/Cuiaba	Amazon Standard Time	GMT-3
America/Curacao	Atlantic Standard Time	GMT-4
America/Danmarkshavn	Greenwich Mean Time	GMT
America/Dawson	Pacific Standard Time	GMT-8
America/Dawson_Creek	Mountain Standard Time	GMT-7
America/Denver	Mountain Standard Time	GMT-7
America/Detroit	Eastern Standard Time	GMT-5
America/Dominica	Atlantic Standard Time	GMT-4
America/Edmonton	Mountain Standard Time	GMT-7
America/Eirunepe	Acre Time	GMT-5
America/El_Salvador	Central Standard Time	GMT-6
America/Ensenada	Pacific Standard Time	GMT-8
America/Fort_Wayne	Eastern Standard Time	GMT-5
America/Fortaleza	Brazil Time	GMT-3
America/Glace_Bay	Atlantic Standard Time	GMT-4
America/Godthab	Western Greenland Time	GMT-3

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
America/Goose_Bay	Atlantic Standard Time	GMT-4
America/Grand_Turk	Eastern Standard Time	GMT-5
America/Grenada	Atlantic Standard Time	GMT-4
America/Guadeloupe	Atlantic Standard Time	GMT-4
America/Guatemala	Central Standard Time	GMT-6
America/Guayaquil	Ecuador Time	GMT-5
America/Guyana	Guyana Time	GMT-4
America/Halifax	Atlantic Standard Time	GMT-4
America/Havana	Central Standard Time	GMT-5
America/Hermosillo	Mountain Standard Time	GMT-7
America/Indiana/ Indianapolis	Eastern Standard Time	GMT-5
America/Indiana/Knox	Eastern Standard Time	GMT-5
America/Indiana/ Marengo	Eastern Standard Time	GMT-5
America/Indiana/ Petersburg	Eastern Standard Time	GMT-5
America/Indiana/ Tell_City	Eastern Standard Time	GMT-5
America/Indiana/ Vevay	Eastern Standard Time	GMT-5
America/Indiana/ Vincennes	Eastern Standard Time	GMT-5
America/Indiana/ Winamac	Eastern Standard Time	GMT-5
America/Indianapolis	Eastern Standard Time	GMT-5
America/Inuvik	Mountain Standard Time	GMT-7
America/Iqaluit	Eastern Standard Time	GMT-5
America/Jamaica	Eastern Standard Time	GMT-5
America/Jujuy	Argentine Time	GMT-3
America/Juneau	Alaska Standard Time	GMT-9
America/Kentucky/ Louisville	Eastern Standard Time	GMT-5
America/Kentucky/ Monticello	Eastern Standard Time	GMT-5
America/Knox_IN	Eastern Standard Time	GMT-5
America/La_Paz	Bolivia Time	GMT-4
America/Lima	Peru Time	GMT-5
America/Los_Angeles	Pacific Standard Time	GMT-8
America/Louisville	Eastern Standard Time	GMT-5
America/Maceio	Brazil Time	GMT-3
America/Managua	Central Standard Time	GMT-6

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
America/Manaus	Amazon Standard Time	GMT-4
America/Marigot	Atlantic Standard Time	GMT-4
America/Martinique	Atlantic Standard Time	GMT-4
America/Mazatlan	Mountain Standard Time	GMT-7
America/Mendoza	Argentine Time	GMT-3
America/Menominee	Central Standard Time	GMT-6
America/Merida	Central Standard Time	GMT-6
America/Mexico_City	Central Standard Time	GMT-6
America/Miquelon	Pierre & Miquelon Standard Time	GMT-3
America/Moncton	Atlantic Standard Time	GMT-4
America/Monterrey	Central Standard Time	GMT-6
America/Montevideo	Uruguay Time	GMT-3
America/Montreal	Eastern Standard Time	GMT-5
America/Montserrat	Atlantic Standard Time	GMT-4
America/Nassau	Eastern Standard Time	GMT-5
America/New_York	Eastern Standard Time	GMT-5
America/Nipigon	Eastern Standard Time	GMT-5
America/Nome	Alaska Standard Time	GMT-9
America/Noronha	Fernando de Noronha Time	GMT-2
America/ North_Dakota/ Center	Central Standard Time	GMT-6
America/ North_Dakota/ New_Salem	Central Standard Time	GMT-6
America/Panama	Eastern Standard Time	GMT-5
America/Pangnirtung	Eastern Standard Time	GMT-5
America/Paramaribo	Suriname Time	GMT-3
America/Phoenix	Mountain Standard Time	GMT-7
America/Port_of_Spain	Atlantic Standard Time	GMT-4
America/Port-au-Prince	Eastern Standard Time	GMT-5
America/Porto_Acre	Acre Time	GMT-5
America/Porto_Velho	Amazon Standard Time	GMT-4
America/Puerto_Rico	Atlantic Standard Time	GMT-4
America/Rainy_River	Central Standard Time	GMT-6
America/Rankin_Inlet	Eastern Standard Time	GMT-6
America/Recife	Brazil Time	GMT-3
America/Regina	Central Standard Time	GMT-6
America/Resolute	Central Standard Time	GMT-6
America/Rio_Branco	Acre Time	GMT-5
America/Rosario	Argentine Time	GMT-3

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
America/Santarem	Amazon Standard Time	GMT-4
America/Santiago	Chile Time	GMT-3
America/Santo_Domingo	Atlantic Standard Time	GMT-4
America/Sao_Paulo	Brazil Time	GMT-2
America/Scoresbysund	Eastern Greenland Time	GMT-1
America/Shiprock	Mountain Standard Time	GMT-7
America/St_Barthelemy	Atlantic Standard Time	GMT-4
America/St_Johns	Newfoundland Standard Time	GMT-3
America/St_Kitts	Atlantic Standard Time	GMT-4
America/St_Lucia	Atlantic Standard Time	GMT-4
America/St_Thomas	Atlantic Standard Time	GMT-4
America/St_Vincent	Atlantic Standard Time	GMT-4
America/Swift_Current	Central Standard Time	GMT-6
America/Tegucigalpa	Central Standard Time	GMT-6
America/Thule	Atlantic Standard Time	GMT-4
America/Thunder_Bay	Eastern Standard Time	GMT-5
America/Tijuana	Pacific Standard Time	GMT-8
America/Tortola	Atlantic Standard Time	GMT-4
America/Toronto	Central Standard Time	GMT-6
America/Vancouver	Pacific Standard Time	GMT-8
America/Virgin	Atlantic Standard Time	GMT-4
America/Whitehorse	Pacific Standard Time	GMT-8
America/Winnipeg	Central Standard Time	GMT-6
America/Yakutat	Alaska Standard Time	GMT-9
America/Yellowknife	Mountain Standard Time	GMT-7
Antarctica/Casey	Western Standard Time (Australia)	GMT+8
Antarctica/Davis	Davis Time	GMT+7
Antarctica/DumontDUrville	Dumont-d'Urville Time	GMT+10
Antarctica/Mawson	Mawson Time	GMT+6
Antarctica/McMurdo	New Zealand Standard Time	GMT+13
Antarctica/Palmer	Chile Time	GMT-3
Antarctica/Rothera	Rothera Time	GMT-3
Antarctica/South_Pole	New Zealand Standard Time	GMT+13
Antarctica/Syowa	Syowa Time	GMT+3
Antarctica/Vostok	Vostok Time	GMT+6
Arctic/Longyearbyen	Central European Time	GMT+1
ART	Eastern European Time	GMT+2
Asia/Aden	Arabia Standard Time	GMT+3

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Asia/Almaty	Alma-Ata Time	GMT+6
Asia/Amman	Eastern European Time	GMT+2
Asia/Anadyr	Anadyr Time	GMT+12
Asia/Aqtau	Aqtau Time	GMT+4
Asia/Aqtobe	Aqtobe Time	GMT+5
Asia/Ashgabat	Turkmenistan Time	GMT+5
Asia/Ashkhabad	Turkmenistan Time	GMT+5
Asia/Baghdad	Arabia Standard Time	GMT+3
Asia/Bahrain	Arabia Standard Time	GMT+3
Asia/Baku	Azerbaijan Time	GMT+4
Asia/Bangkok	Indochina Time	GMT+7
Asia/Beirut	Eastern European Time	GMT+2
Asia/Bishkek	Kirgizstan Time	GMT+5
Asia/Brunei	Brunei Time	GMT+8
Asia/Calcutta	India Standard Time	GMT+5
Asia/Choibalsan	Choibalsan Time	GMT+9
Asia/Chongqing	China Standard Time	GMT+8
Asia/Chungking	China Standard Time	GMT+8
Asia/Colombo	Sri Lanka Time	GMT+6
Asia/Dacca	Bangladesh Time	GMT+6
Asia/Damascus	Eastern European Time	GMT+2
Asia/Dhaka	Bangladesh Time	GMT+6
Asia/Dili	East Timor Time	GMT+9
Asia/Dubai	Gulf Standard Time	GMT+4
Asia/Dushanbe	Tajikistan Time	GMT+5
Asia/Gaza	Eastern European Time	GMT+2
Asia/Harbin	China Standard Time	GMT+8
Asia/Ho_Chi_Min	Indochina Time	GMT+7
Asia/Hong_Kong	Hong Kong Time	GMT+8
Asia/Hovd	Hovd Time	GMT+7
Asia/Irkutsk	Irkutsk Time	GMT+8
Asia/Istanbul	Eastern European Time	GMT+2
Asia/Jakarta	West Indonesia Time	GMT+7
Asia/Jayapura	East Indonesia Time	GMT+9
Asia/Jerusalem	Israel Standard Time	GMT+2
Asia/Kabul	Afghanistan Time	GMT+4
Asia/Kamchatka	Petropavlovsk-Kamchatski Time	GMT+12
Asia/Karachi	Pakistan Time	GMT+5
Asia/Kashgar	China Standard Time	GMT+8

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Asia/Katmandu	Nepal Time	GMT+5
Asia/Kolkata	Indian Standard Time	GMT+5:30
Asia/Krasnoyarsk	Krasnoyarsk Time	GMT+7
Asia/Kuala_Lumpur	Malaysia Time	GMT+8
Asia/Kuching	Malaysia Time	GMT+8
Asia/Kuwait	Arabia Standard Time	GMT+3
Asia/Macao	China Standard Time	GMT+8
Asia/Macau	China Standard Time	GMT+8
Asia/Magadan	Magadan Time	GMT+11
Asia/Makassar	Central Indonesia Time	GMT+8
Asia/Manila	Philippines Time	GMT+8
Asia/Muscat	Gulf Standard Time	GMT+4
Asia/Nicosia	Eastern European Time	GMT+2
Asia/Novosibirsk	Novosibirsk Time	GMT+6
Asia/Omsk	Omsk Time	GMT+6
Asia/Oral	Oral Time	GMT+4
Asia/Phnom_Penh	Indochina Time	GMT+7
Asia/Pontianak	West Indonesia Time	GMT+7
Asia/Pyongyang	Korea Standard Time	GMT+9
Asia/Qatar	Arabia Standard Time	GMT+3
Asia/Qyzylorda	Qyzylorda Time	GMT+6
Asia/Rangoon	Myanmar Time	GMT+6
Asia/Riyadh	Arabia Standard Time	GMT+3
Asia/Riyadh87	Arabia Standard Time	GMT+3
Asia/Riyadh88	Arabia Standard Time	GMT+3
Asia/Riyadh89	Arabia Standard Time	GMT+3
Asia/Saigon	Indochina Time	GMT+7
Asia/Sakhalin	Sakhalin Time	GMT+10
Asia/Samarkand	Turkmenistan Time	GMT+5
Asia/Seoul	Korea Standard Time	GMT+9
Asia/Shanghai	China Standard Time	GMT+8
Asia/Singapore	Singapore Time	GMT+8
Asia/Taipei	China Standard Time	GMT+8
Asia/Tashkent	Uzbekistan Time	GMT+5
Asia/Tbilisi	Georgia Time	GMT+4
Asia/Tehran	Iran Standard Time	GMT+3
Asia/Tel_Aviv	Israel Standard Time	GMT+2
Asia/Thimbu	Bhutan Time	GMT+6
Asia/Thimphu	Bhutan Time	GMT+6

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Asia/Tokyo	Japan Standard Time	GMT+9
Asia/Ujung_Pandang	Central Indonesia Time	GMT+8
Asia/Ulaanbaatar	Ulaanbaatar Time	GMT+8
Asia/Ulan_Bator	Ulaanbaatar Time	GMT+8
Asia/Urumqi	China Standard Time	GMT+8
Asia/Vientiane	Indochina Time	GMT+7
Asia/Vladivostok	Vladivostok Time	GMT+10
Asia/Yakutsk	Yakutsk Time	GMT+9
Asia/Yekaterinburg	Yekaterinburg Time	GMT+5
Asia/Yerevan	Armenia Time	GMT+4
AST	Alaska Standard Time	GMT-9
Atlantic/Azores	Azores Time	GMT-1
Atlantic/Bermuda	Atlantic Standard Time	GMT-4
Atlantic/Canary	Western European Time	GMT
Atlantic/Cape_Verde	Cape Verde Time	GMT-1
Atlantic/Faeroe	Western European Time	GMT
Atlantic/Faroe	Western European Time	GMT
Atlantic/Jan_Mayen	Eastern Greenland Time	GMT+1
Atlantic/Madeira	Western European Time	GMT
Atlantic/Reykjavik	Greenwich Mean Time	GMT
Atlantic/South_Georgia	South Georgia Standard Time	GMT-2
Atlantic/St_Helena	Greenwich Mean Time	GMT
Atlantic/Stanley	Falkland Is. Time	GMT-3
Australia/ACT	Eastern Standard Time (New South Wales)	GMT+11
Australia/Adelaide	Central Standard Time (South Australia)	GMT+10
Australia/Brisbane	Eastern Standard Time (Queensland)	GMT+10
Australia/Broken_Hill	Central Standard Time (South Australia/New South Wales)	GMT+10
Australia/Canberra	Eastern Standard Time (New South Wales)	GMT+11
Australia/Currie	Eastern Standard Time (Tasmania)	GMT+11
Australia/Darwin	Central Standard Time (Northern Territory)	GMT+8:.45
Australia/Eucla	Western Standard Time (Australia)	GMT+11
Australia/Hobart	Eastern Standard Time (Tasmania)	GMT+11
Australia/LHI	Load Howe Standard Time	GMT+11
Australia/Lindeman	Eastern Standard Time (Queensland)	GMT+10
Australia/Lord_Howe	Load Howe Standard Time	GMT+11
Australia/Melbourne	Eastern Standard Time (Victoria)	GMT+11
Australia/North	Central Standard Time (Northern Territory)	GMT+9
Australia/NSW	Eastern Standard Time (New South Wales)	GMT+11

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Australia/Perth	Western Standard Time (Australia)	GMT+8
Australia/Queensland	Eastern Standard Time (Queensland)	GMT+10
Australia/South	Central Standard Time (South Australia)	GMT+10
Australia/Sydney	Eastern Standard Time (New South Wales)	GMT+11
Australia/Tasmania	Eastern Standard Time (Tasmania)	GMT+11
Australia/Victoria	Eastern Standard Time (Victoria)	GMT+11
Australia/West	Western Standard Time (Australia)	GMT+8
Australia/Yancowinna	Central Standard Time (South Australia/New South Wales)	GMT+10
BET	Brazil Time	GMT-2
Brazil/Acre	Acre Time	GMT-5
Brazil/DeNoronha	Fernando de Noronha Time	GMT-2
Brazil/East	Brazil Time	GMT-2
Brazil/West	Amazon Standard Time	GMT-4
BST	Bangladesh Time	GMT+6
Canada/Atlantic	Atlantic Standard Time	GMT-4
Canada/Central	Central Standard Time	GMT-6
Canada/Eastern	Eastern Standard Time	GMT-5
Canada/East-Saskatchewan	Central Standard Time	GMT-6
Canada/Mountain	Mountain Standard Time	GMT-7
Canada/Newfoundland	Newfoundland Standard Time	GMT-3
Canada/Pacific	Pacific Standard Time	GMT-8
Canada/Saskatchewan	Central Standard Time	GMT-6
Canada/Yukon	Pacific Standard Time	GMT-8
CAT	Central African Time	GMT+2
CET	Central European Time	GMT+1
Chile/Continental	Chile Time	GMT-3
Chile/EasterIsland	Easter Is. Time	GMT-5
CNT	Newfoundland Standard Time	GMT-3
CST	Central Standard Time	GMT-6
CST6CDT	Central Standard Time	GMT-6
CTT	China Standard Time	GMT+8
Cuba	Central Standard Time	GMT-5
EAT	Eastern African Time	GMT+3
ECT	Central European Time	GMT+1
EET	Eastern European Time	GMT+2
Egypt	Eastern European Time	GMT+2
Eire	Greenwich Mean Time	GMT
EST	Eastern Standard Time	GMT-5

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
EST5EDT	Eastern Standard Time	GMT-5
Etc/GMT	Greenwich Standard Time	GMT
Etc/GMT-0	Greenwich Standard Time	GMT-0
Etc/GMT-1	Greenwich Standard Time	GMT-1
Etc/GMT-10	Greenwich Standard Time	GMT-10
Etc/GMT-11	Greenwich Standard Time	GMT-11
Etc/GMT-12	Greenwich Standard Time	GMT-12
Etc/GMT-13	Greenwich Standard Time	GMT-13
Etc/GMT-14	Greenwich Standard Time	GMT-14
Etc/GMT-2	Greenwich Standard Time	GMT-2
Etc/GMT-3	Greenwich Standard Time	GMT-3
Etc/GMT-4	Greenwich Standard Time	GMT-4
Etc/GMT-5	Greenwich Standard Time	GMT-5
Etc/GMT-6	Greenwich Standard Time	GMT-6
Etc/GMT-7	Greenwich Standard Time	GMT-7
Etc/GMT-8	Greenwich Standard Time	GMT-8
Etc/GMT-9	Greenwich Standard Time	GMT-9
Etc/GMT+0	Greenwich Standard Time	GMT+0
Etc/GMT+1	Greenwich Standard Time	GMT+1
Etc/GMT+11	Greenwich Standard Time	GMT+11
Etc/GMT+12	Greenwich Standard Time	GMT+12
Etc/GMT+2	Greenwich Standard Time	GMT+2
Etc/GMT+3	Greenwich Standard Time	GMT+3
Etc/GMT+4	Greenwich Standard Time	GMT+4
Etc/GMT+5	Greenwich Standard Time	GMT+5
Etc/GMT+6	Greenwich Standard Time	GMT+6
Etc/GMT+7	Greenwich Standard Time	GMT+7
Etc/GMT+8	Greenwich Standard Time	GMT+8
Etc/GMT+9	Greenwich Standard Time	GMT+9
Etc/GMT0	Greenwich Standard Time	GMT
Etc/Greenwich	Greenwich Standard Time	GMT
Etc/UTC	Greenwich Standard Time	GMT
Etc/Universal	Greenwich Standard Time	GMT
Etc/Zulu	Greenwich Standard Time	GMT+2
Europe/Amsterdam	Central European Time	GMT+1
Europe/Andorra	Central European Time	GMT+1
Europe/Athens	Eastern European Time	GMT+2
Europe/Belfast	Greenwich Mean Time	GMT
Europe/Belgrade	Central European Time	GMT+1

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Europe/Berlin	Central European Time	GMT+1
Europe/Bratislava	Central European Time	GMT+1
Europe/Brussels	Central European Time	GMT+1
Europe/Bucharest	Eastern European Time	GMT+2
Europe/Budapest	Central European Time	GMT+1
Europe/Chisinau	Eastern European Time	GMT+2
Europe/Copenhagen	Central European Time	GMT+1
Europe/Dublin	Greenwich Mean Time	GMT
Europe/Gibraltar	Central European Time	GMT+1
Europe/Guernsey	Central European Time	GMT+1
Europe/Helsinki	Eastern European Time	GMT+2
Europe/Isle_of_Man	Greenwich Mean Time	GMT
Europe/Istanbul	Eastern European Time	GMT+2
Europe/Jersey	Greenwich Mean Time	GMT
Europe/Kaliningrad	Eastern European Time	GMT+2
Europe/Kiev	Eastern European Time	GMT+2
Europe/Lisbon	Western European Time	GMT
Europe/Ljubljana	Central European Time	GMT+1
Europe/London	Greenwich Mean Time	GMT
Europe/Luxembourg	Central European Time	GMT+1
Europe/Madrid	Central European Time	GMT+1
Europe/Malta	Central European Time	GMT+1
Europe/Mariehamn	Eastern European Time	GMT+2
Europe/Minsk	Eastern European Time	GMT+2
Europe/Monaco	Central European Time	GMT+1
Europe/Moscow	Moscow Standard Time	GMT+3
Europe/Nicosia	Eastern European Time	GMT+2
Europe/Oslo	Central European Time	GMT+1
Europe/Paris	Central European Time	GMT+1
Europe/Podgorica	Central European Time	GMT+1
Europe/Prague	Central European Time	GMT+1
Europe/Riga	Eastern European Time	GMT+2
Europe/Rome	Central European Time	GMT+1
Europe/Samara	Samara Time	GMT+4
Europe/San_Marino	Central European Time	GMT+1
Europe/Sarajevo	Central European Time	GMT+1
Europe/Simferopol	Eastern European Time	GMT+2
Europe/Skopje	Central European Time	GMT+1
Europe/Sofia	Eastern European Time	GMT+2

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Europe/Stockholm	Central European Time	GMT+1
Europe/Tallinn	Eastern European Time	GMT+2
Europe/Tirane	Central European Time	GMT+1
Europe/Tiraspol	Eastern European Time	GMT+2
Europe/Uzhgorod	Eastern European Time	GMT+2
Europe/Vaduz	Central European Time	GMT+1
Europe/Vatican	Central European Time	GMT+1
Europe/Vienna	Central European Time	GMT+1
Europe/Vilnius	Eastern European Time	GMT+2
Europe/Volgograd	Moscow Standard Time	GMT+3
Europe/Warsaw	Central European Time	GMT+1
Europe/Zagreb	Central European Time	GMT+1
Europe/Zaporozhye	Eastern European Time	GMT+2
Europe/Zurich	Central European Time	GMT+1
GB	Greenwich Mean Time	GMT
GB-Eire	Greenwich Mean Time	GMT
GMT	Greenwich Mean Time	GMT
GMT-0	Greenwich Mean Time	GMT
GMT+0	Greenwich Mean Time	GMT
GMT0	Greenwich Mean Time	GMT
Greenwich	Greenwich Mean Time	GMT
Hongkong	Hong Kong Time	GMT+8
HST	Hawaii Standard Time	GMT-10
Iceland	Greenwich Mean Time	GMT
IET	Eastern Standard Time	GMT-5
Indian/Antananarivo	Eastern African Time	GMT+3
Indian/Chagos	Indian Ocean Territory Time	GMT+6
Indian/Christmas	Christmas Island Time	GMT+7
Indian/Cocos	Cocos Islands Time	GMT+6
Indian/Comoro	Eastern African Time	GMT+3
Indian/Kerguelen	French Southern and Antarctic Lands Time	GMT+5
Indian/Mahe	Seychelles Time	GMT+4
Indian/Maldives	Maldives Time	GMT+5
Indian/Mauritius	Mauritius Time	GMT+4
Indian/Mayotte	Eastern African Time	GMT+3
Indian/Reunion	Reunion Time	GMT+4
Iran	Iran Standard Time	GMT+3
Israel	Israel Standard Time	GMT+2
IST	India Standard Time	GMT+5

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Jamaica	Eastern Standard Time	GMT-5
Japan	Japan Standard Time	GMT+9
JST	Japan Standard Time	GMT+9
Kwajalein	Marshall Islands Time	GMT+12
Libya	Eastern European Time	GMT+2
MET	Middle Europe Time	GMT+1
Mexico/BajaNorte	Pacific Standard Time	GMT-8
Mexico/BajaSur	Mountain Standard Time	GMT-7
Mexico/General	Central Standard Time	GMT-6
Mideast/Riyadh87	Arabia Standard Time	GMT+3
Mideast/Riyadh88	Arabia Standard Time	GMT+3
Mideast/Riyadh89	Arabia Standard Time	GMT+3
MIT	West Samoa Time	GMT-11
MST	Mountain Standard Time	GMT-7
MST7MDT	Mountain Standard Time	GMT-7
Navajo	Mountain Standard Time	GMT-7
NET	Armenia Time	GMT+4
NST	New Zealand Standard Time	GMT+13
NZ	New Zealand Standard Time	GMT+13
NZ-CHAT	Chatham Standard Time	GMT+13
Pacific/Apia	West Samoa Time	GMT-11
Pacific/Auckland	New Zealand Standard Time	GMT+13
Pacific/Chatham	Chatham Standard Time	GMT+13
Pacific/Easter	Easter Is. Time	GMT-5
Pacific/Efate	Vanuatu Time	GMT+11
Pacific/Enderbury	Phoenix Is. Time	GMT+13
Pacific/Fakaofu	Tokelau Time	GMT-10
Pacific/Fiji	Fiji Time	GMT+12
Pacific/Funafuti	Tuvalu Time	GMT+12
Pacific/Galapagos	Galapagos Time	GMT-6
Pacific/Gambier	Gambier Time	GMT-9
Pacific/Guadalcanal	Solomon Is. Time	GMT+11
Pacific/Guam	Chamorro Standard Time	GMT+10
Pacific/Honolulu	Hawaii Standard Time	GMT-10
Pacific/Johnston	Hawaii Standard Time	GMT-10
Pacific/Kiritimati	Line Is. Time	GMT+14
Pacific/Kosrae	Kosrae Time	GMT+11
Pacific/Kwajalein	Marshall Islands Time	GMT+12
Pacific/Majuro	Marshall Islands Time	GMT+12

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
Pacific/Marquesas	Marquesas Time	GMT-9
Pacific/Midway	Samoa Standard Time	GMT-11
Pacific/Nauru	Nauru Time	GMT+12
Pacific/Niue	Niue Time	GMT-11
Pacific/Norfolk	Norfolk Time	GMT+11
Pacific/Noumea	New Caledonia Time	GMT+11
Pacific/Pago_Pago	Samoa Standard Time	GMT-11
Pacific/Palau	Palau Time	GMT+9
Pacific/Pitcairn	Pitcairn Standard Time	GMT-8
Pacific/Ponape	Ponape Time	GMT+11
Pacific/Port_Moresby	Papua New Guinea Time	GMT+10
Pacific/Rarotonga	Cook Is. Time	GMT-10
Pacific/Saipan	Chamorro Standard Time	GMT+10
Pacific/Samoa	Samoa Standard Time	GMT-11
Pacific/Tahiti	Tahiti Time	GMT-10
Pacific/Tarawa	Gilbert Is. Time	GMT+12
Pacific/Tongatapu	Tonga Time	GMT+13
Pacific/Truk	Truk Time	GMT+10
Pacific/Wake	Wake Time	GMT+12
Pacific/Wallis	Wallis & Futuna Time	GMT+12
Pacific/Yap	Yap Time	GMT+10
PLT	Pakistan Time	GMT+5
PNT	Mountain Standard Time	GMT-7
Poland	Central European Time	GMT+1
Portugal	Western European Time	GMT
PRC	China Standard Time	GMT+8
PRT	Atlantic Standard Time	GMT-4
PST	Pacific Standard Time	GMT-8
PST8PDT	Pacific Standard Time	GMT-8
ROC	China Standard Time	GMT+8
ROK	Korea Standard Time	GMT+9
Singapore	Singapore Time	GMT+8
SST	Solomon Is. Time	GMT+11
Turkey	Eastern European Time	GMT+2
Universal	Coordinated Universal Time	GMT
US/Alaska	Alaska Standard Time	GMT-9
US/Aleutian	Hawaii-Aleutian Standard Time	GMT-10
US/Arizona	Mountain Standard Time	GMT-7
US/Central	Central Standard Time	GMT-6

Table 77. Variable length notation time zones list (continued)

Long Name	Description	Relative to GMT
US/Eastern	Eastern Standard Time	GMT-5
US/East-Indiana	Eastern Standard Time	GMT-5
US/Hawaii	Hawaii Standard Time	GMT-10
US/Indiana-Starke	Eastern Standard Time	GMT-5
US/Michigan	Eastern Standard Time	GMT-5
US/Mountain	Mountain Standard Time	GMT-7
US/Pacific	Pacific Standard Time	GMT-8
US/Pacific-New	Pacific Standard Time	GMT-8
US/Samoa	Samoa Standard Time	GMT-11
UTC	Coordinated Universal Time	GMT
VST	Indochina Time	GMT+7
WET	Western European Time	GMT
W-SU	Moscow Standard Time	GMT+3
Zulu	Central African Time	GMT+2

Chapter 14. The auditing feature

Tivoli Workload Scheduler provides you with an auditing feature to track changes applied to the database and to the plan. By default the auditing feature is disabled.

This chapter is divided into the following sections:

- “Auditing overview”
- “Enabling the auditing feature”
- “Audit log format” on page 468
- “Sample audit log files” on page 472

Auditing overview

Tivoli Workload Scheduler provides you with two types of audit log files:

Plan audit log file

Logs all user action performed against the plan. Actions are logged whether or not they are successful.

Database audit log file

Logs all actions against scheduling objects, even if an object is opened and saved without changing it.

The auditing logs are created once a day at 00:00:00 UTC in the following directories:

TWS_home/audit/plan

TWS_home/audit/database

on each workstation in the Tivoli Workload Scheduler network to minimize the risk of audit failure due to network problems.

Records are written to audit logs using Coordinated Universal Time (UTC), therefore audit logs will contain events from 00:00:00 UTC to 23:59:59 UTC regardless of the time zone of the local workstation. Note however, that if the workstation is restarted prior to 00:00:00 UTC on the new calendar day (local time zone), new empty audit files will be created, but not written to prior to 00:00:00 UTC.

Enabling the auditing feature

Auditing is disabled by default when the product is installed. You can enable or disable the auditing feature by setting the value of these two global options:

`plan audit level`

`database audit level`

using the **optman** command line on the master domain manager.

Set the value to **yes** to enable auditing or to **no** to disable auditing. If these options are not present among the global options then auditing is disabled. For information on the **optman** command line, refer to *IBM Tivoli Workload Scheduler Administration Guide*.

Plan auditing settings take effect when **JnextPlan** is run.

Audit log format

The audit log formats are basically the same for both the plan and the database audit log files. They consist of a header portion containing information about when the log was created, the workstation it relates to, and whether it is a plan or a database audit log, and a body containing a record for each action performed.

All data is stored in normal text and formatted to be readable and editable from a text editor such as *vi* or *notepad*.

In the audit log files:

The format for dates is *yyyymmdd* where:

- *yyyy* is the year
- *mm* is the month
- *dd* is the day

The format for times is *hhmmss* where:

- *hh* is the hour
- *mm* is the minutes
- *ss* is the seconds

The formats of the audit log files header and body are described in the following sections.

Audit log header format

The header contains the following set of information ordered in fields separated by vertical bars (|):

Fields Contents

Log Type
HEADER

GMT Date
The GMT date when the log file was created.

GMT Time
The GMT time when the log file was created.

Local Date
The local date when the log file was created. The local date is defined by the time zone option of the workstation.

Local Time
The local time when the log file was created. The local time is defined by the time zone option of the workstation.

Object Type
DATABASE for a database log file and PLAN for a plan log file.

Workstation Name
The Tivoli Workload Scheduler workstation name for which this file was created. Each workstation in the Tivoli Workload Scheduler network creates its own log.

User ID

The Tivoli Workload Scheduler user ID that created the log file.

Version

The version of the file.

Level The logging level.

Audit log body format

The log contains, for each action logged, the following set of information ordered in fields separated by vertical bars (|):

Log Type

An eight-character value indicating the source of the log record. The following log types are supported:

HEADER

The log file header

conman

conman command text

PLAN Plan action

STAGEMAN

stageman run

RELEASE

release command text

DATABASE

Database action

PARMS

Parameter command text

MAKESEC

makesec run

GMT Date

The GMT date when the action was performed.

GMT Time

The GMT time when the action was performed.

Local Date

The local date when the action was performed. The local date is defined by the time zone option of the workstation.

Local Time

The local time when the action was performed. The local time is defined by the time zone option of the workstation.

Object Type

The type of the object that the action was performed on. It is one of the following:

DATABASE

Database definition

DBWKSTN

Database workstation definition

DBWKCLS	Database workstation class definition
DBDOMAIN	Database domain definition
DBUSER	Database user definition
DBJBSTRM	Database job stream definition
DBJOB	Database job definition
DBCAL	Database calendar definition
DBPROMPT	Database prompt definition
DBPARAM	Database parameter (variable) definition
DBVARTAB	Database variable table definition
DBRES	Database resource definition
DBSEC	Database security
PLAN	Plan
PLWKSTN	Plan workstation
PLDOMAIN	Plan domain
PLJBSTRM	Plan job stream
PLJOB	Plan job
PLPROMPT	Plan prompt
PLRES	Plan resource
PLFILE	Plan file

Action Type

The action taken against the object. Valid action types are **add**, **delete**, **modify**, **expand**, and **install**. The appropriate values for this field depend on the action being taken.

For database audit logs

Tivoli Workload Scheduler records **add**, **delete**, and **modify** actions for workstation, workstation classes, domains, users, jobs, job streams, calendars, prompts, resources, parameters (variables), and variable tables in the database.

For plan audit log

Tivoli Workload Scheduler records **add**, **delete**, **modify**, **expand**, and **install**. The **install** action is recorded when the **makesec** command is run to install a new security file.

List and **display** actions for objects are not logged.

For parameters, the command line with arguments is logged.

Workstation Name

The Tivoli Workload Scheduler workstation on which the user performs the action.

User ID

The logon user who performed the particular action. On Windows operating systems, if the user who installed WebSphere Application Server was domain user, for Log Type **stageman** and **conman** this field contains the fully qualified user name *domain\user*.

Object Name

The fully qualified name of the object. The format of this field depends on the object type as follows:

DBWKSTN

workstation

DBWKCLS

workstation_class

DBDOMAIN

domain

DBUSER

[workstation#]user

DBJBSTRM

workstation#jobstream

DBJOB

workstation#job

DBCAL

calendar

DBPROMPT

prompt

DBPARM

vartable.parameter

DBVARTAB

vartable

DBRES

workstation#resource

PLWKSTN

workstation

PLDOMAIN

domain

PLJBSTRM

workstation#jobstream_instance

PLJOB*workstation#jobstream_instance.job***PLPROMPT***[workstation#]prompt***PLRES***workstation#resource***PLFILE***workstation#path(qualifier)***Action Dependent Data**

The action-specific data fields. The format of this data is dependent on the Action Type field.

Sample audit log files

This is a sample database audit log:

```

HEADER |20060207|084124|20060207|094124|DATABASE|      |WK1|      | | |Version=A1.0| Level=1
DATABASE|20060207|084124|20060207|094124|DBRES  |ADD  |WK1|operator1| |res=WK1#RESOURCE  |
DATABASE|20060207|100524|20060207|110524|DBWKSTN |MODIFY|WK1|operator1| |ws=TIVOLI10      |
DATABASE|20060207|100525|20060207|110525|DBWKSTN |MODIFY|WK1|operator1| |ws=ASLUTRI1     |
DATABASE|20060207|100525|20060207|110525|DBWKSTN |MODIFY|WK1|operator1| |ws=WK1          |
DATABASE|20060207|100526|20060207|110526|DBDOMAIN|MODIFY|WK1|operator1| |dom=MASTERDM   |
DATABASE|20060207|100610|20060207|110610|DBWKSTN |MODIFY|WK1|operator1| |ws=TIVOLI10     |
DATABASE|20060207|100610|20060207|110610|DBWKSTN |MODIFY|WK1|operator1| |ws=ASLUTRI1     |
DATABASE|20060207|100611|20060207|110611|DBWKSTN |MODIFY|WK1|operator1| |ws=WK1          |
DATABASE|20060207|100611|20060207|110611|DBWKSTN |ADD   |WK1|operator1| |ws=WK2          |
DATABASE|20060207|100612|20060207|110612|DBDOMAIN|MODIFY|WK1|operator1| |dom=MASTERDM   |

```

This is a sample plan audit log:

```

HEADER |20060207|100758|20060207|110758|PLAN  |      |WK1|admin| |      |Version=A1.0|Level=1
STAGEMAN|20060207|100758|20060207|110758|PLAN  |INSTALL|WK1|admin| |C:\IBM\TWS\oper1\Symphony|
          AWSBH030I The new Symphony file is installed.
STAGEMAN|20060207|100758|20060207|110758|PLAN  |INSTALL|WK1|admin| |C:\IBM\TWS\oper1\Sinfonia|
          AWSBH036I Multi-workstation Symphony file copied to C:\IBM\TWS\oper1\Sinfonia
STAGEMAN|20060207|100758|20060207|110758|ADITLEVL|MODIFY |WK1|admin| |      |
          AWSBH077I Audit level changing from 0 to 1.
CONMAN  |20060207|100800|20060207|110800|PLWKSTN |MODIFY | |admin| |WK1
          continue & start
CONMAN  |20060207|100941|20060207|110941|PLWKSTN |MODIFY | |admin| |SLUTRI1
          limit cpu=slutri1;10
PLAN    |20060207|101018|20060207|111018|PLWKSTN |MODIFY |WK1|oper1| |WK1
          limit cpu=SLUTRI1;20
PLAN    |20060207|101028|20060207|111028|PLDOMAIN|MODIFY |WK1|oper1| |ECCOLO
          reply ECCOLO;yes

```

A **ResetPlan** command run against the current production plan is stored in the plan audit log file as follows:

```
STAGEMAN|20060207|100758|20060207|110758|PLAN|DELETE|WK1|admin|  
|/home/WK1/schedlog/M200603140127|
```

```
AWSBHV025I The old Symphony file renamed /home/WK1/schedlog/M200603140127
```

Chapter 15. Managing extended agents

Workstations are generally physical assets (that is computers) but they might also be logical definitions, hosted by a physical workstation, representing operating systems or applications where you want to run jobs or job streams. In this case they are defined as extended agents.

This chapter describes the extended agents, their interface and provides information for programmers who want to create custom access methods. Extended Agents are used to extend the job scheduling functions of IBM Tivoli Workload Scheduler to other systems and applications such as: local or remote UNIX operating systems, Peoplesoft, SAP R/3, z/OS, JES, OPC, Oracle EBS, and VMS.

This chapter is divided into the following sections:

- “What are extended agents?”
- “Access method interface” on page 476
- “Running methods” on page 480
- “Troubleshooting” on page 483

The following example shows a definition for a z/OS extended agent workstation named MVSCPU that uses the *mvsjes* access method.

```
cpuname MVSCPU description "zOS extended agent"
  os other
  node mvsesa36.rome.tivoli.com
  tcpaddr 5000
  domain masterdm
  for maestro
    type x-agent
    host ROCIOUS
    access mvsjes
end
```

The following example describes a Tivoli Workload Scheduler job named orajob2 that runs in an Oracle Applications extended agent workstation named ora002. It logs on to UNIX as orajobs and launches a job under Oracle Applications. The Oracle Applications job is named poxacr and its owner is global. If recovery is needed, Tivoli Workload Scheduler will run job recov2 and then continue processing.

```
ora002#orajob2
  streamlogon orajobs
  scriptname "-user global -job fnd 'application developer' po poxacr -prn ps4 2 -v1 'abc'"
  description "oracle apps job #2"
  recovery continue after recov2
```

The arguments of scriptname differ by application. For additional information, refer to *IBM Tivoli Workload Scheduler for Applications: User's Guide*

What are extended agents?

Extended agents are used to extend the job scheduling functions of Tivoli Workload Scheduler to other systems and applications.

An extended agent is defined as a workstation that has a host and an access method. The host is any other workstation, except another extended agent. The access method is an IBM-supplied or user-supplied script or program that is run by the host whenever the extended agent is referenced in the production plan. For example, to launch a job on an extended agent, the host runs the access method, passing it job details as command line options. The access method communicates with the external system or application to launch the job and return the status of the job.

Workstation definition

Each extended agent must have a logical workstation definition. This logical workstation must be hosted by a Tivoli Workload Scheduler physical workstation, either a master domain manager, domain manager, or a fault-tolerant agent. The extended agent workstation definition references the name of the access method and the host workstation. When jobs are launched on the extended agent workstation, the access method is called and passes the job information to the external system.

A physical workstation can host a maximum of 255 extended agents.

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** Checks the availability of a file. Use this option to check file opens dependencies.
- GS** 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” on page 477 for more information.

taskstring

A string of up to 255 characters associated with the task. See “Task options” on page 477.

Task options

The task options are listed in Table 78. An X means that the option is valid for the task.

Table 78. 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 extended agent, the host, and the master domain manager separated by commas.

-n *nodename*

Specifies the node name of the computer associated with the extended 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 extended agent, if any. This is defined in the extended agent's 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” 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 [*cpu*time]

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** [*cpu*time]. 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** [*cpu*time].

UT [*error*message]

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

You can use a method options file to specify special login information and other options. Tivoli Workload Scheduler reads the file, if it exists, before running a method. If the file is modified after Tivoli Workload Scheduler is started, the changes only take effect when it is stopped and restarted.

The file can contain Tivoli Workload Scheduler options and any other method information. The options recognized by Tivoli Workload Scheduler are as follows:

LJuser=*username*

CFuser=*username*

GSuser=*username*

GStimeout=*seconds*

where:

LJuser=*username*

Specifies the login to use for the **LJ** and **MJ** tasks. The default is the login from the job definition.

CFuser=*username*

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 Tivoli Workload Scheduler was installed.

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.

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.

Note: If the extended agent's host is a Windows computer, these users must be defined as Tivoli Workload Scheduler user objects.

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\netmth.opts

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 79 on page 481.

Table 79. Launch job task (LJ) messages

Task	Method Response	Tivoli Workload Scheduler Action
LJ and MJ	%CJ <i>state</i> [<i>mjstring</i>]	Sets job state to <i>state</i> . Includes <i>mjstring</i> in any subsequent MJ task.
	%JS [<i>cputime</i>]	Sets job state to SUCC.
	Exit code=non-zero	Sets job state to ABEND.
	%UT [<i>errormessage</i>] and Exit code=2	Sets job state to ABEND and displays <i>errormessage</i> .

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 480 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)

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 80 on page 482.

Table 80. 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)

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 479 for more information.

Method responses are summarized in Table 81:

Table 81. Get status task (GS) messages

Task	Method Response	Tivoli Workload Scheduler Action
GS	%CJ state [jobid]	Sets job state to <i>state</i> and includes <i>jobid</i> 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

The **cpuinfo** command can be used in an access method to return information from a workstation definition. See “Cpuinfo command” 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

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

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, 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 16. 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 487
- “Defining an internetwork dependency” on page 489
- “Managing internetwork dependencies” on page 490
- “Internetwork dependencies in a mixed environment” on page 493

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 ELISCHE 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.

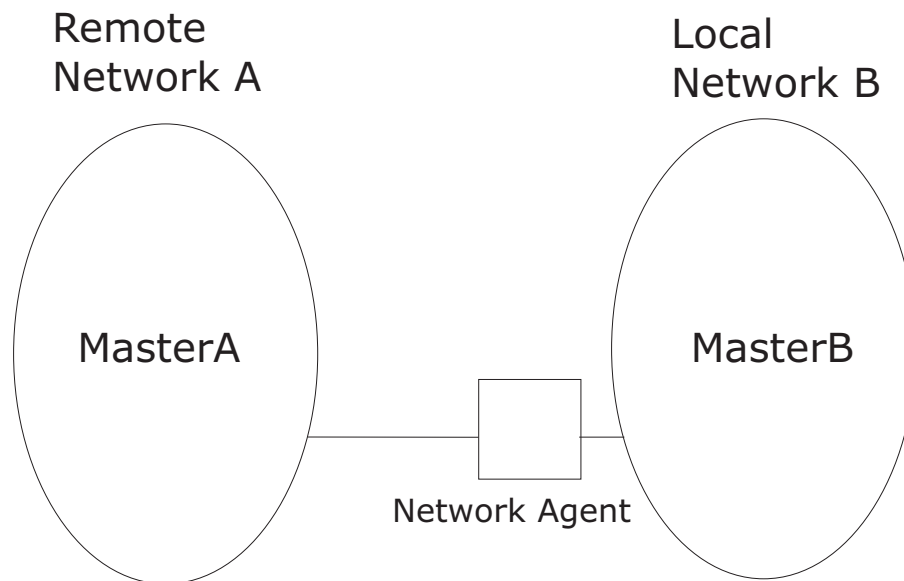


Figure 28. Local and remote networks

Assuming that:

- MasterA is the master domain manager of the remote network, Network A, and that:
 - **twsmasterA** 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:
 - **twsmasterB** 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=twsmasterB
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.

A sample internetwork dependency definition

See Also

For the equivalent Job Scheduling Console task, see "Creating Job Streams" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

Managing internetwork dependencies

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:

Ennnmms

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 is an 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 is 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 `rsched`
- 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 afile";follows=netagt::remote1#rsched
```

See Also

For the equivalent Job Scheduling Console task, see "Creating Job Streams" in the *IBM Tivoli Workload Scheduler Job Scheduling Console User's Guide*.

Internetwork dependencies in a mixed environment

Table 82 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 82. 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 <i>schedtime</i> 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 <i>schedtime</i> 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 <i>schedtime</i> keyword in the job definition is automatically removed by Wks_B.	Net_A sends the information to Wks_B. If defined, the <i>schedtime</i> keyword in the job definition is automatically removed by Wks_B.	Net_A sends the information to Wks_B. If defined, the <i>schedtime</i> 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 <i>schedtime</i> keyword is parsed by Wks_B.	This is a version 8.3 environment.

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

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

The following tables show the parameters of each event type.

JobStatusChanged

This event is sent when the status of a job changes.

Table 83. Parameters of JobStatusChanged event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
JobWorkstation	The name of the workstation associated with the job instance.	string	✓		✓	✓	✓	1	16	
JobStreamWorkstation	The name of the workstation associated with the job stream containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance containing the job instance.	string								
JobStreamName	The name of the job stream instance containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance containing the job instance.	datetime	✓			✓				
JobName	The name of the job instance.	string	✓	✓	✓	✓	✓	1	40	*
Priority	The job priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job instance is key job.	boolean	✓			✓				
ActualStart	The actual start time of the job instance. Is significant only if the job has already started.	datetime	✓			✓				
EstimatedDuration	The estimated duration of the job.	duration	✓			✓				
ActualDuration	The actual duration of the job.	duration	✓			✓				

Table 83. Parameters of JobStatusChanged event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
ReturnCode	The return code of the job. It is significant only for completed or abended jobs.	numeric	✓		✓	✓			
LatestStart	The value of the until restriction of the job instance.	datetime	✓			✓			
Deadline	The value of the deadline restriction of the job instance.	datetime	✓			✓			
Status	The new status of the job instance.	string Value can be: Error, Held, Ready, Running, Successful, Undecided, Waiting	✓		✓			1	
InternalStatus	The new internal status of the job instance.	string Value can be: ABEND, ABENP, CANCP, DONE, ERROR, EXEC, EXTRN, FAIL, FENCE, HOLD, INTRO, PEND, READY, RJOB, SCHED, SUCC, SUCCP, SUSP, USER, WAIT, WAITD	✓		✓			1	
Login	The login of the user used to run the job.	string	✓		✓	✓	✓	1	
EveryFrequency	The every frequency of the job instance.	duration	✓			✓			
ErrorMessage	A message describing the cause of the job failure. It has significance only when the job instance goes in error (Fail) status.	string	✓		✓	✓	✓	1	
TimeStamp	When the event was generated.	datetime							
Hostname	The fully qualified host name of the computer that sent the event.	string						1	

Table 83. Parameters of JobStatusChanged event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
IPAddress	The IP address of the computer that sent the event.	string							
PlanNumber	The run number of the Plan running when the event was generated.	numeric							

JobUntil

This event is sent when the latest start time of a job has elapsed.

Table 84. Parameters of JobUntil event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
JobWorkstation	The name of the workstation associated with the job instance.	string	✓		✓	✓	✓	1 16	
JobStreamWorkstation	The name of the workstation associated with the job stream containing the job instance.	string	✓	✓	✓	✓	✓	1 16	*
JobStreamID	The ID of the job stream instance containing the job instance.	string							
JobStreamName	The name of the job stream instance containing the job instance.	string	✓	✓	✓	✓	✓	1 16	*
JobStreamSchedTime	The scheduled time of the job stream instance containing the job instance.	datetime	✓			✓			
JobName	The name of the job instance.	string	✓	✓	✓	✓	✓	1 40	*
Priority	The job priority.	numeric (0-101)	✓		✓	✓			
Monitored	True if the job instance is key job.	boolean	✓			✓			
EstimatedDuration	The estimated duration of the job.	duration	✓			✓			
LatestStart	The value of the until restriction of the job instance.	datetime	✓			✓			
Deadline	The value of the deadline restriction of the job instance.	datetime	✓			✓			

Table 84. Parameters of JobUntil event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
Status	The new status of the job instance.	string Value can be: Cancelled, Error, Held, Ready, Running, Successful, Undecided, Waiting	✓		✓			1	
InternalStatus	The new internal status of the job instance.	string Value can be: ABEND, ABENP, CANCL, CANCP, DONE, ERROR, EXEC, EXTRN, FAIL, FENCE, HOLD, INTRO, PEND, READY, RJOB, SCHED, SUCC, SUCCP, SUSP, USER, WAIT, WAITD	✓		✓			1	
Login	The login of the user used to run the job.	string	✓		✓	✓	✓	1	
EveryFrequency	The every frequency of the job instance.	duration	✓			✓			
UntilAction	The action taken after the until restriction time has elapsed.	string Value can be: Cancel, Continue, Suppress	✓		✓			1	
TimeStamp	When the event was generated.	datetime							
Hostname	The fully qualified host name of the computer that sent the event.	string						1	
IPAddress	The IP address of the computer that sent the event.	string							
PlanNumber	The run number of the Plan running when the event was generated.	numeric							

JobSubmit

This event is sent when a job is submitted.

Table 85. Parameters of JobSubmit event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
JobWorkstation	The name of the workstation associated with the job instance.	string	✓		✓	✓	✓	1	16	
JobStreamWorkstation	The name of the workstation associated with the job stream containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance containing the job instance.	string								
JobStreamName	The name of the job stream instance containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance containing the job instance.	datetime	✓			✓				
JobName	The name of the job instance.	string	✓	✓	✓	✓	✓	1	40	*
Priority	The job priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job instance is key job.	boolean	✓			✓				
EstimatedDuration	The estimated duration of the job.	duration	✓			✓				
LatestStart	The value of the until restriction of the job instance.	datetime	✓			✓				
Deadline	The value of the deadline restriction of the job instance.	datetime	✓			✓				
Login	The login of the user used to run the job.	string	✓		✓	✓	✓	1		
EveryFrequency	The every frequency of the job instance.	duration	✓			✓				
TimeStamp	When the event was generated.	datetime								
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

JobCancel

This event is sent when a job is cancelled.

Table 86. Parameters of JobCancel event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
JobWorkstation	The name of the workstation associated with the job instance.	string	✓		✓	✓	✓	1	16	
JobStreamWorkstation	The name of the workstation associated with the job stream containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance containing the job instance.	string								
JobStreamName	The name of the job stream instance containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance containing the job instance.	datetime	✓			✓				
JobName	The name of the job instance.	string	✓	✓	✓	✓	✓	1	40	*
Priority	The job priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job instance is key job.	boolean	✓			✓				
ActualStart	The actual start time of the job instance. Is significant only if the job has already started.	datetime	✓			✓				
EstimatedDuration	The estimated duration of the job.	duration	✓			✓				
ActualDuration	The actual duration of the job.	duration	✓			✓				
ReturnCode	The return code of the job. It is significant only for completed or abended jobs.	numeric	✓		✓	✓				
LatestStart	The value of the until restriction of the job instance.	datetime	✓			✓				
Deadline	The value of the deadline restriction of the job instance.	datetime	✓			✓				
Status	The new status of the job instance.	string Value can be: Cancelled, Error, Held, Ready, Running, Successful, Undecided, Waiting	✓		✓			1		

Table 86. Parameters of JobCancel event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
InternalStatus	The new internal status of the job instance.	string Value can be: ABEND, ABENP, CANCL, CANCP, DONE, ERROR, EXEC, EXTRN, FAIL, FENCE, HOLD, INTRO, PEND, READY, RJOB, SCHED, SUCC, SUCCP, SUSP, USER, WAIT, WAITD	✓		✓			1	
Login	The login of the user used to run the job.	string	✓		✓	✓	✓	1	
EveryFrequency	The every frequency of the job instance.	duration	✓			✓			
TimeStamp	When the event was generated.	datetime							
Hostname	The fully qualified host name of the computer that sent the event.	string						1	
IPAddress	The IP address of the computer that sent the event.	string							
PlanNumber	The run number of the Plan running when the event was generated.	numeric							

JobRestart

This event is sent when a job is restarted.

Table 87. Parameters of JobRestart event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
JobWorkstation	The name of the workstation associated with the job instance.	string	✓		✓	✓	✓	1 16	
JobStreamWorkstation	The name of the workstation associated with the job stream containing the job instance.	string	✓	✓	✓	✓	✓	1 16	*

Table 87. Parameters of JobRestart event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
JobStreamID	The ID of the job stream instance containing the job instance.	string								
JobStreamName	The name of the job stream instance containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance containing the job instance.	datetime	✓			✓				
JobName	The name of the job instance.	string	✓	✓	✓	✓	✓	1	40	*
Priority	The job priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job instance is key job.	boolean	✓			✓				
ActualStart	The actual start time of the job instance. Is significant only if the job has already started.	datetime	✓			✓				
EstimatedDuration	The estimated duration of the job.	duration	✓			✓				
ActualDuration	The actual duration of the job.	duration	✓			✓				
ReturnCode	The return code of the job. It is significant only for completed or abended jobs.	numeric	✓		✓	✓				
LatestStart	The value of the until restriction of the job instance.	datetime	✓			✓				
Deadline	The value of the deadline restriction of the job instance.	datetime	✓			✓				
Login	The login of the user used to run the job.	string	✓		✓	✓	✓	1		
EveryFrequency	The every frequency of the job instance.	duration	✓			✓				
TimeStamp	When the event was generated.	datetime								
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								

Table 87. Parameters of JobRestart event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

JobLate

This event is sent when a job becomes late.

To properly use this event type and to trigger an action, you must configure local option:

bm check deadline = 300

This does not apply to critical jobs. See the *Administration Guide* for further reference.

Table 88. Parameters of JobLate event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
JobWorkstation	The name of the workstation associated with the job instance.	string	✓		✓	✓	✓	1	16	
JobStreamWorkstation	The name of the workstation associated with the job stream containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance containing the job instance.	string								
JobStreamName	The name of the job stream instance containing the job instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance containing the job instance.	datetime	✓			✓				
JobName	The name of the job instance.	string	✓	✓	✓	✓	✓	1	40	*
Priority	The job priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job instance is key job.	boolean	✓			✓				
ActualStart	The actual start time of the job instance. Is significant only if the job has already started.	datetime	✓			✓				

Table 88. Parameters of JobLate event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
EstimatedDuration	The estimated duration of the job.	duration	✓			✓			
ActualDuration	The actual duration of the job.	duration	✓			✓			
ReturnCode	The return code of the job. It is significant only for completed or abended jobs.	numeric	✓		✓	✓			
LatestStart	The value of the until restriction of the job instance.	datetime	✓			✓			
Deadline	The value of the deadline restriction of the job instance.	datetime	✓			✓			
Status	The new status of the job instance.	string Value can be: Cancelled, Error, Held, Ready, Running, Successful, Undecided, Waiting	✓		✓			1	
InternalStatus	The new internal status of the job instance.	string Value can be: ABEND, ABENP, CANCL, CANCP, DONE, ERROR, EXEC, EXTRN, FAIL, FENCE, HOLD, INTRO, PEND, READY, RJOB, SCHED, SUCC, SUCCP, SUSP, USER, WAIT, WAITD	✓		✓			1	
Login	The login of the user used to run the job.	string	✓		✓	✓	✓	1	
EveryFrequency	The every frequency of the job instance.	duration	✓			✓			
TimeStamp	When the event was generated.	datetime							

Table 88. Parameters of JobLate event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

JobStreamStatusChanged

This event is sent when the status of a job stream changes.

Table 89. Parameters of JobStreamStatusChanged event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the workstation associated with the job stream instance. This is the same value as the Job Stream Workstation property, but is used for correlation with other events.	string						1		
JobStreamWorkstation	The name of the workstation associated with the job stream instance (same value as in Workstation).	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance.	string								
JobStreamName	The name of the job stream instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance.	datetime	✓			✓				
Priority	The job stream priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job stream instance contains key jobs.	boolean	✓			✓				
LatestStart	The value of the until restriction of the job stream instance.	datetime	✓			✓				

Table 89. Parameters of JobStreamStatusChanged event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
Deadline	The value of the deadline restriction of the job stream instance.	datetime	✓			✓			
Status	The new status of the job stream instance.	string Value can be: Blocked, Cancelled, Error, Held, Ready, Running, Successful, Undecided, Waiting	✓		✓			1	
InternalStatus	The new internal status of the job stream instance.	string Value can be: ABEND, ABENP, CANCP, DONE, ERROR, EXEC, EXTRN, FAIL, FENCE, HOLD, INTRO, PEND, READY, RJOB, SCHED, SUCC, SUCCP, SUSP, USER, WAIT, WAITD	✓		✓			1	
TimeStamp	When the event was generated.	datetime							
Hostname	The fully qualified host name of the computer that sent the event.	string						1	
IPAddress	The IP address of the computer that sent the event.	string							
PlanNumber	The run number of the Plan running when the event was generated.	numeric							

JobStreamCompleted

This event is sent when a job stream has completed.

Table 90. Parameters of JobStreamCompleted event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the workstation associated with the job stream instance. This is the same value as the Job Stream Workstation property, but is used for correlation with other events.	string						1		
JobStreamWorkstation	The name of the workstation associated with the job stream instance (same value as in Workstation).	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance.	string								
JobStreamName	The name of the job stream instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance.	datetime	✓			✓				
Priority	The job stream priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job stream instance contains key jobs.	boolean	✓			✓				
LatestStart	The value of the until restriction of the job stream instance.	datetime	✓			✓				
Deadline	The value of the deadline restriction of the job stream instance.	datetime	✓			✓				
Status	The new status of the job stream instance.	string Value can be: Cancelled, Error, Successful	✓		✓			1		
InternalStatus	The new internal status of the job stream instance.	string Value can be: ABEND, CANCL, SUCC	✓		✓			1		
TimeStamp	When the event was generated.	datetime								

Table 90. Parameters of JobStreamCompleted event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

JobStreamUntil

This event is sent when the latest start time of a job stream has elapsed.

Table 91. Parameters of JobStreamUntil event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the workstation associated with the job stream instance. This is the same value as the Job Stream Workstation property, but is used for correlation with other events.	string						1		
JobStreamWorkstation	The name of the workstation associated with the job stream instance (same value as in Workstation).	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance.	string								
JobStreamName	The name of the job stream instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance.	datetime	✓			✓				
Priority	The job stream priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job stream instance contains key jobs.	boolean	✓			✓				
LatestStart	The value of the until restriction of the job stream instance.	datetime	✓			✓				
Deadline	The value of the deadline restriction of the job stream instance.	datetime	✓			✓				

Table 91. Parameters of JobStreamUntil event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Status	The new status of the job stream instance.	string Value can be: Blocked, Cancelled, Error, Held, Ready, Running, Successful, Undecided, Waiting	✓		✓			1		
InternalStatus	The new internal status of the job stream instance.	string Value can be: ABEND, ABENP, CANCL, CANCP, DONE, ERROR, EXEC, EXTRN, FAIL, FENCE, HOLD, INTRO, PEND, READY, RJOB, SCHED, SUCC, SUCCP, SUSP, USER, WAIT, WAITD	✓		✓			1		
UntilAction	The action taken after the until restriction time has elapsed.	string Value can be: Cancel, Continue, Suppress	✓		✓			1		
TimeStamp	When the event was generated.	datetime								
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

JobStreamSubmit

This event is sent when a job stream is submitted.

Table 92. Parameters of JobStreamSubmit event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the workstation associated with the job stream instance. This is the same value as the Job Stream Workstation property, but is used for correlation with other events.	string						1		
JobStreamWorkstation	The name of the workstation associated with the job stream instance (same value as in Workstation).	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance.	string								
JobStreamName	The name of the job stream instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance.	datetime	✓			✓				
Priority	The job stream priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job stream instance contains key jobs.	boolean	✓			✓				
LatestStart	The value of the until restriction of the job stream instance.	datetime	✓			✓				
Deadline	The value of the deadline restriction of the job stream instance.	datetime	✓			✓				
TimeStamp	When the event was generated.	datetime								
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

JobStreamCancel

This event is sent when a job stream is cancelled.

Table 93. Parameters of JobStreamCancel event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the workstation associated with the job stream instance. This is the same value as the Job Stream Workstation property, but is used for correlation with other events.	string						1		
JobStreamWorkstation	The name of the workstation associated with the job stream instance (same value as in Workstation).	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance.	string								
JobStreamName	The name of the job stream instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance.	datetime	✓			✓				
Priority	The job stream priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job stream instance contains key jobs.	boolean	✓			✓				
LatestStart	The value of the until restriction of the job stream instance.	datetime	✓			✓				
Deadline	The value of the deadline restriction of the job stream instance.	datetime	✓			✓				
Status	The new status of the job stream instance.	string Value can be: Blocked, Cancelled, Error, Held, Ready, Running, Successful, Undecided, Waiting						1		

Table 93. Parameters of JobStreamCancel event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
InternalStatus	The new internal status of the job stream instance.	string Value can be: ABEND, ABENP, CANCL, CANCP, DONE, ERROR, EXEC, EXTRN, FAIL, FENCE, HOLD, INTRO, PEND, READY, RJOB, SCHED, SUCC, SUCCP, SUSE, USER, WAIT, WAITD						1	
TimeStamp	When the event was generated.	datetime							
Hostname	The fully qualified host name of the computer that sent the event.	string						1	
IPAddress	The IP address of the computer that sent the event.	string							
PlanNumber	The run number of the Plan running when the event was generated.	numeric							

JobStreamLate

This event is sent when a job stream becomes late.

To properly use this event type and to trigger an action, you must configure local option:

bm check deadline = 300

See the *Administration Guide* for further reference.

Table 94. Parameters of JobStreamLate event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
Workstation	The name of the workstation associated with the job stream instance.	string						1	

Table 94. Parameters of JobStreamLate event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
JobStreamWorkstation	The name of the workstation associated with the job stream instance (same value as in Workstation).	string	✓	✓	✓	✓	✓	1	16	*
JobStreamID	The ID of the job stream instance.	string								
JobStreamName	The name of the job stream instance.	string	✓	✓	✓	✓	✓	1	16	*
JobStreamSchedTime	The scheduled time of the job stream instance.	datetime	✓			✓				
Priority	The job stream priority.	numeric (0-101)	✓		✓	✓				
Monitored	True if the job stream instance contains key jobs.	boolean	✓			✓				
LatestStart	The value of the until restriction of the job stream instance.	datetime	✓			✓				
Deadline	The value of the deadline restriction of the job stream instance.	datetime	✓			✓				
Status	The new status of the job stream instance.	string Value can be: Blocked, Cancelled, Error, Held, Ready, Running, Successful, Undecided, Waiting	✓		✓			1		
InternalStatus	The new internal status of the job stream instance.	string Value can be: ABEND, ABENP, CANCL, CANCP, DONE, ERROR, EXEC, EXTRN, FAIL, FENCE, HOLD, INTRO, PEND, READY, RJOB, SCHED, SUCC, SUCCP, SUSP, USER, WAIT, WAITD	✓		✓			1		
TimeStamp	When the event was generated.	datetime								

Table 94. Parameters of JobStreamLate event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

WorkstationStatusChanged

This event is sent when a workstation is started or stopped.

Table 95. Parameters of WorkstationStatusChanged event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the workstation.	string	✓	✓	✓	✓	✓	1	16	
Running	TRUE if the workstation is running. FALSE if it is stopped.	boolean	✓			✓				
StopReason	The reason why batchman (or jobman) was terminated. Significant only if one of them is not running.	string Value can be: Abend, Stop						1		
TimeStamp	The date and time when the event was generated.	datetime								
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

ApplicationServerStatusChanged

This event is sent when the embedded WebSphere Application Server has stopped or is restarting.

Table 96. Parameters of ApplicationServerStatusChanged event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the workstation	string	✓	✓	✓	✓	✓	1	16	*
Running	True if the application server on the workstation is running, False if stopped.	boolean	✓			✓				
StopReason	The reason why the application server was terminated. Significant only if it is not running.	string Value can be: Unexpected end, Stop	✓					1		
Restarting	True if the system will retry automatically to start the application server. Significant only if the application server is not running.	boolean	✓			✓				
NoRestartReason	The reason why the system will not automatically start the application server. Is significant only if the application server is not running and Restarting is false.	string Value can be: No auto start, Maximum restart exceeded, Terminated too quickly	✓					1		
TimeStamp	The date and time when the event was generated.	datetime								
Hostname	The fully qualified hostname of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

ChildWorkstationLinkChanged

This event is sent when the workstation defined in the event rule links or unlinks one of its child workstations.

Table 97. Parameters of ChildWorkstationLinkChanged event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the child workstation that is changing link status.	string	✓	✓	✓	✓	✓	1	16	*
ParentWorkstation	The name of the workstation that is sending the event.	string	✓		✓	✓	✓	1	16	
LinkStatus	The new link status of the workstation.	string Value can be: Linked, Unlinked	✓					1		
UnlinkReason	Shows the reason why the workstation was unlinked. Available only if LinkStatus is Unlinked.	string Value can be: Broken, Dropped						1		
TimeStamp	When the event was generated.	datetime								
Hostname	The fully qualified hostname of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

ParentWorkstationLinkChanged

This event is sent when the parent workstation of the workstation defined in the event rule is linked or unlinked. The parent workstation will send a ChildWorkstationLinkChanged event when the link in the other direction is established or lost.

Table 98. Parameters of ParentWorkstationLinkChanged event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Workstation	The name of the workstation that is sending the event.	string	✓	✓	✓	✓	✓	1	16	*
ParentWorkstation	The name of the parent workstation that is changing link status.	string	✓		✓	✓	✓	1	16	

Table 98. Parameters of ParentWorkstationLinkChanged event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
LinkStatus	The new link status of the workstation.	string Value can be: Linked, Unlinked	✓					1		
UnlinkReason	Shows the reason why the workstation was unlinked. Available only if LinkStatus is Unlinked.	string Value can be: Broken, Dropped						1		
TimeStamp	When the event was generated.	datetime								
Hostname	The fully qualified host name of the computer that sent the event.	string						1		
IPAddress	The IP address of the computer that sent the event.	string								
PlanNumber	The run number of the Plan running when the event was generated.	numeric								

PromptStatusChanged

This event is sent when a prompt is asked or replied.

Table 99. Parameters of PromptStatusChanged event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
Name	The name or the number of the prompt.	string	✓	✓	✓	✓	✓	1	8	*
Text	The text of the prompt.	string	✓		✓	✓	✓			
Status	The status of the prompt.	string Value can be: Asked, RepliedNo, RepliedYes	✓		✓			1		
Type	The type of the prompt.	string Value can be: Global, JobLocal, JobStreamLocal	✓		✓			1		
RecoveryPrompt	Indicates if the prompt is a recovery prompt. Available only when the status is asked.	boolean	✓			✓				

Table 99. Parameters of PromptStatusChanged event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
Workstation Note: not for use with global prompts	The name of the workstation associated with the job instance or job stream instance that owns the prompt. Available only for local prompts.	string	✓		✓	✓	✓	1	
JobStreamWorkstation Note: not for use with global prompts	The name of the workstation associated with the job stream instance that owns the prompt (or that contains the job instance that owns the prompt). Available only for local prompts.	string	✓		✓	✓	✓	1	
JobStreamID Note: not for use with global prompts	The ID of the job stream instance that owns the prompt (or that contains the job instance that owns the prompt). Available only for local prompts.	string							
JobStreamName Note: not for use with global prompts	The name of the job stream instance that owns the prompt (or that contains the job instance that owns the prompt). Available only for local prompts.	string	✓		✓	✓	✓	1	
JobStreamSchedTime Note: not for use with global prompts	The scheduled time of the job stream instance that owns the prompt (or that contains the job instance that owns the prompt). Available only for local prompts.	datetime	✓			✓			
JobName Note: not for use with global prompts	The name of the job instance that owns the prompt. Available only for local prompts owned by jobs.	string	✓		✓	✓	✓	1	

Table 99. Parameters of PromptStatusChanged event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
TimeStamp	When the event was generated.	datetime							
Hostname	The fully qualified host name of the computer that sent the event.	string						1	
IPAddress	The IP address of the computer that sent the event.	string							
PlanNumber	The run number of the Plan running when the event was generated.	numeric							

FileMonitor events

FileMonitor events are:

- FileCreated
- FileDeleted
- ModificationCompleted
- LogMessageWritten

The following tables show the parameters of each event type.

1 **Note:** FileMonitor events are not supported for workload broker agents. To work
 1 around this problem, when creating event rules for workload broker agents,
 1 use the workstation name of the workstation hosting the workload broker
 1 server.

FileCreated

This event is sent when a specified file is created.

Table 100. Parameters of FileCreated event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
FileName	The absolute path and filename(s) of the specified file(s).	string	✓	✓			✓	1	
SampleInterval	The interval (in seconds) with which the specified file is monitored. numeric	numeric	✓	✓		✓			60
Workstation	The name of the workstation for which the event is generated.	string	✓	✓				1 16	
TimeStamp	The time when the event is sent.	datetime							
Hostname	The fully qualified host name of the computer that sends the event.	string	✓		✓	✓	✓	1	

Table 100. Parameters of FileCreated event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
IPAddress	The IP address of the computer that sends the event.	string							
EventRuleId	The identifier of the event rule.	string							

FileDeleted

This event is sent when a specified file is deleted.

Table 101. Parameters of FileDeleted event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
FileName	The absolute path and filename(s) of the specified file(s).	string	✓	✓			✓	1	
SampleInterval	The interval (in seconds) with which the specified file is monitored. numeric	numeric	✓	✓		✓			60
Workstation	The name of the workstation for which the event is generated.	string	✓	✓				1 16	
TimeStamp	The time when the event is sent.	datetime							
Hostname	The fully qualified host name of the computer that sends the event.	string	✓		✓	✓	✓	1	
IPAddress	The IP address of the computer that sends the event.	string							
EventRuleId	The identifier of the event rule.	string							

ModificationCompleted

This event is sent when a specified file has not been modified in two consecutive monitoring cycles after any of the following:

- Its creation
- A detected modification.

1 Table 102. Parameters of ModificationCompleted event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
FileName	The absolute path and filename(s) of the specified file(s).	string	✓	✓				1	
SampleInterval	The interval (in seconds) with which the specified file is monitored. numeric	numeric	✓	✓		✓			60
LastWriteTime	The time when the specified file was last modified.	datetime	✓		✓	✓			

Table 102. Parameters of ModificationCompleted event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
Workstation	The name of the workstation for which the event is generated.	string	✓	✓				1 16	
TimeStamp	The time when the event is sent.	datetime							
Hostname	The fully qualified host name of the computer that sends the event.	string	✓		✓	✓	✓	1	
IPAddress	The IP address of the computer that sends the event.	string							
EventRuleId	The identifier of the event rule.	string							

LogMessageWritten

This event is sent when a specific string is found in a file. Be aware of the following when you change or delete and create the file again during the monitoring activity:

- The matches parameter counts the total number of matches found since monitoring started. It does not report the number of matches currently in the file. If you change the file while monitoring is in progress, there may be a discrepancy between the number of matches recorded and the ones presently shown in the file.
- The scanning mechanism is such that at each sample period, scanning is picked up from the last know position. Already scanned portions of the file are not checked again. If you make changes to a file section that has already undergone scanning, these changes will not be detected in the current monitoring process.

Table 103. Parameters of LogMessageWritten event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
FileName	The absolute path and filename(s) of the specified file(s).	string	✓	✓			✓	1	
Matches	The number of matches found since monitoring started.	numeric	✓		✓	✓			
MatchExpression	The string to look for in the log file(s) being monitored.	string	✓	✓				1	
LastWriteTime	The time when the specified file was last modified.	datetime	✓		✓	✓			
Size	The size of the log file (in bytes) when the most recent matching log file entry was found.	filesize	✓		✓	✓			
SampleInterval	The interval (in seconds) with which the specified file is monitored. numeric	numeric	✓	✓		✓			60

Table 103. Parameters of LogMessageWritten event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
Workstation	The name of the workstation for which the event is generated.	string	✓	✓				1 16	
TimeStamp	The time when the event is sent.	datetime							
Hostname	The fully qualified host name of the computer that sends the event.	string	✓		✓	✓	✓	1	
IPAddress	The IP address of the computer that sends the event.	string							
Linetxt	The contents of the line where the search string was found.	string							
EventRuleId	The identifier of the event rule.	string							

TWSApplicationMonitor events

TWSApplicationMonitor events are:

- “TWSMessageQueues”
- “TWSProcessMonitor” on page 524
- “TWSDiskMonitor” on page 524

The following tables show the parameters of each event type.

TWSMessageQueues

This event is sent when the filling percentage of the message queues changes according to the settings you defined. The event is started when the mailbox reaches a given percentage of the maximum allowed size.

Table 104. Parameters of TWSMessageQueues event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
FillingPercentage	The filling percentage	Numeric (0-100)	✓	✓			✓		
MailBoxName	The name of the Tivoli Workload Scheduler mailbox to monitor	String Value can be: • appserverbox • clbox • courier • intercom • mailbox • monbox • moncmd • server • tomaster • pobox	✓	✓	✓				

Table 104. Parameters of TWSTMessageQueues event types. (continued)

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
SampleInterval	The interval, in seconds, to check for the mailbox filling percentage	Numeric	✓						60
Workstation	The Tivoli Workload Scheduler workstation on which the event is generated	String	✓	✓				1 16	

TWSTProcessMonitor

This event is sent when the a critical Tivoli Workload Scheduler process is not running. The event is notified only the first time the problem is detected.

Table 105. Parameters of TWSTProcessMonitor event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)	Default value
ProcessName	The name of the Tivoli Workload Scheduler process to monitor	String Value can be • agent • appservman • batchman • jobman • mailman • monman • netman	✓	✓	✓				
TWSTPath	The directory containing the Symphony file and the bin directory	String	✓	✓				1	
SampleInterval	The interval, in seconds, to check for the mailbox filling percentage	Numeric	✓						60
Workstation	The Tivoli Workload Scheduler workstation on which the event is generated	String	✓	✓				1 16	

TWSDiskMonitor

This event is sent when the file system is filling up. The event is sent when the file system reaches a given percentage of the maximum available size.

Table 106. Parameters of TWSDiskMonitor event types.

Property name	Description	Type	Filtering allowed	Required	Multiple values allowed	Multiple filter predicates allowed	Wildcard allowed	Length (min-max)		Default value
MountPoint	The mount point of the file system where Tivoli Workload Scheduler is installed	String	✓	✓				1		
Percentage	The percentage of used disk space	Numeric (0-100)	✓	✓						
SampleInterval	The interval, in seconds, to check for the mailbox filling percentage	Numeric	✓							60
Workstation	The Tivoli Workload Scheduler workstation on which the event is generated	String	✓	✓				1	16	

Action providers and definitions

This section gives details on the action types of the following action providers:

- TECEventForwarder
- MailSender
- MessageLogger
- TWSAction
- GenericAction

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.

The following table lists the parameters of TECFWD:

Table 107. Parameters of TECFWD action types.

Property name	Description	Type	Required	Length (min-max)		Default value
Host	Host name or IP of the TEC recipient	string		1		
Port	Port number of the TEC recipient	numeric		1		
Message	The event message	string	✓	1		
Severity	The severity of the event message	string Value can be: Critical, Fatal, Normal, Warning	✓	1		
LOB	The line of business impacted by the situation reported by the event	string		1		

Table 107. Parameters of TECFWD action types. (continued)

Property name	Description	Type	Required	Length (min-max)		Default value
Parm_1	Custom parameters	string		1		
Parm_2		string		1		
Parm_3		string		1		
Parm_4		string		1		
Parm_5		string		1		
Parm_6		string		1		
Parm_7		string		1		
Parm_8		string		1		
Parm_9		string		1		
Parm_10		string		1		

MailSender actions

This provider implements a single action named SendMail that connects to an SMTP server to send an e-mail. You can use optman to customize the following related attributes:

- Mail sender
- SMTP server
- SMTP port number
- Mail user name
- Mail user password
- SSL

The following table lists the parameters of SendMail:

Table 108. Parameters of SendMail action types.

Property name	Description	Type	Required	Length (min-max)		Default value
To	The main receiver list. Multiple addresses can be specified using commas as separators.	string	✓	1		
Cc	The copied receiver list. Multiple addresses can be specified using commas as separators.	string		1		
Bcc	The blank copied receiver list. Multiple addresses can be specified using commas as separators.	string		1		
Subject	The subject of the mail.	string	✓	1		
Body	The body of the mail.	string		1		

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.

The following table lists the parameters of MSGLOG:

Table 109. Parameters of MSGLOG action types.

Property name	Description	Type	Required	Length (min-max)		Default value
Message	The message	string	✓	1		

Table 109. Parameters of MSGLOG action types. (continued)

Property name	Description	Type	Required	Length (min-max)		Default value
Severity	The severity of the message	string Value can be: Info, Warning, Error Default is Info	✓	1		Info
ObjectKey	A key identifying the object to which the message pertains	string	✓	1		
Source	A custom string that identifies the source	string		1		
LOB	The line of business impacted by the situation reported by the message	string		1		
Group	The group of operators in whose queue the message is placed	string		1		

TWSAction actions

TWSAction actions are:

- SubmitJobStream
- SubmitJob
- SubmitAdHocJob
- ReplyPrompt

The following tables show the parameters of each action type.

SubmitJobStream

This action submits a job stream.

Table 110. Parameters of SubmitJobStream action types.

Property name	Description	Type	Required	Length (min-max)		Default value
JobStreamName	The name of the job stream	string	✓	1	16	
JobStreamWorkstationName	The name of the workstation associated with the job stream	string Value can be: Info, Warning, Error	✓	1	16	
JobStreamSchedTime	The scheduled time of the job stream instance	datetime				
JobStreamAlias	The job stream alias	string		1	16	
Parm_1	Custom parameters	string		1		
Parm_2		string		1		
Parm_3		string		1		
Parm_4		string		1		
Parm_5		string		1		
Parm_6		string		1		
Parm_7		string		1		
Parm_8		string		1		
Parm_9		string		1		
Parm_10		string		1		

You can use the custom parameters (Parm_1 through Parm_10) to override the values of variables or local parameters already specified in the job stream definition. The variables or local parameters must be already defined in the

database (with the \$parm or “Variable table definition” on page 147 composer commands on the Tivoli Workload Scheduler database or with the “parms” on page 398 utility command on the local database).

When you enter the value of a Parm_ *n* tag, you must write it in the form:

variablename=variablevalue

where:

variablename

The name of the variable or local parameter as defined in the database.

variablevalue

The value it has to take in this submit sched action.

In the xml definition of the rule, for example, the tagging for Parm_1 within the sbs action is:

```
<parameter name="Parm_1">
<value>variablename=variablevalue</value>
</parameter>
```

SubmitJob

This action submits a job.

Table 111. Parameters of SubmitJob action types.

Property name	Description	Type	Required	Length (min-max)		Default value
JobDefinitionName	The name of the job as defined in the database	string	✓	1	40	
JobDefinitionWorkstationName	The name of the workstation associated with the job as defined in the database	string	✓	1	16	
JobJStreamName	The name of the job stream that contains the job	string		1	16	
JobJStreamWorkstationName	The name of the workstation associated with the job stream	string		1	16	
SchedTimeResolutionCriteria	The resolution criteria for the scheduled time of the job stream instance	string Value can be: Previous (closest previous job stream), Next (closest next job stream), Any (closest previous or closest next job stream)				
JobAlias	The job alias	string		1	40	
JobUseUniqueAlias	Generate a unique alias for the job	boolean	✓			true
Parm_1	Custom parameters	string		1		
Parm_2		string		1		
Parm_3		string		1		
Parm_4		string		1		
Parm_5		string		1		
Parm_6		string		1		
Parm_7		string		1		
Parm_8		string		1		
Parm_9		string		1		
Parm_10		string		1		

SubmitAdHocJob

This action submits an ad hoc job.

Table 112. Parameters of SubmitAdHocJob action types.

Property name	Description	Type	Required	Length (min-max)		Default value
JobTask	The task to execute	string	✓	1		
JobLogin	The login name	string	✓	1		
JobPriority	the job priority	numeric 0-101 Default is 10		1		10
JobWorkstationName	The name of the workstation associated with the job	string	✓	1	16	
JobStreamName	The name of the job stream that contains the job	string		1	16	
JobStreamWorkstationName	The name of the workstation associated with the job stream	string		1	16	
SchedTimeResolutionCriteria	The resolution criteria for the scheduled time of the job stream instance	string Value can be: Previous (closest previous job stream), Next (closest next job stream), Any (closest previous or closest next job stream)				
JobAlias	The job alias	string		1	40	
JobUseUniqueAlias	Generate a unique alias for the job	boolean	✓			true
JobType	The job type	string Value can be: Script or Command	✓	1		Script

ReplyPrompt

This action replies to a prompt.

Table 113. Parameters of ReplyPrompt action types.

Property name	Description	Type	Required	Length (min-max)		Default value
PromptName	The name of the prompt	string	✓	1	8	
PromptAnswer	The reply to the prompt	string Value can be: ReplyYes or ReplyNo	✓	1		

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.

1

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.

The following table lists the parameters of RunCommand:

Table 114. Parameters of RunCommand action types.

Property name	Description	Type	Required	Length (min-max)	Default value
Command	The command that must be run	string	✓		
WorkingDir	The location where the command is to be run	string			
Param1	Command parameters	string			
Param2		string			
Param3		string			
Param4		string			
Param5		string			
Param6		string			
Param7		string			
Param8		string			
Param9		string			
Param10		string			
Param11		string			
Param12		string			
Param13		string			
Param14		string			
Param15		string			
Env1	Environment parameters for the command	string			
Env2		string			
Env3		string			
Env4		string			
Env5		string			
Env6		string			
Env7		string			
Env8		string			
Env9		string			
Env10		string			
Env11		string			
Env12		string			
Env13		string			
Env14		string			
Env15		string			

Appendix B. Web services interface

Using the Web Services Interface you can use the Web Services access mechanism to create your own client application to perform a subset of Tivoli Workload Scheduler functions to manage jobs and job streams in the plan.

Whenever you install a Tivoli Workload Scheduler workstation the following WSDL files:

```
SchedulingFactory.wsdl  
JobService.wsdl  
JobstreamService.wsdl
```

are automatically installed on the machine in the directory where the Tivoli Workload Scheduler is installed under the following path:

```
.\appserver\profiles\twsprofile\installedApps\DefaultNode\  
TWSEngineModel.ear\PlanServicesWeb.war\WEB-INF
```

These WSDL files provide you with:

- The server part interfacing the master domain manager to perform the supported subset of scheduling operations against jobs and job streams in production.
- A means of creating your own client interface from where service requesters can request to perform a subset of operations from any system in your environment by accessing the following location:
https://localhost:31116/PlanServicesWeb/services/service_name for distributed environment
https://localhost:31126/PlanServicesWeb/services/service_name for z/OS environment

where *localhost* must be replaced manually with the hostname of the master domain manager, and *service_name* is the name of the service you invoke, that is SchedulingFactory, JobService, or JobStreamService.

You can also modify the port number to use when binding to the master domain manager, 31116 for distributed environment or 31126 for z/OS environment.

Service requesters can create job streams and jobs in the plan by submitting actions (**submitJobStream**, **submitJob**, **submitJobAdHoc**), to monitor the submitted objects by getting actions (**queryJobStreams**, **queryJobs**, **getJobsList**, **getProperties**, **getJobOutput**), and to manage them by setting actions (**setProperties**, **cancel**, **kill**, **releaseAllDependencies**). Neither database actions nor other plan actions can be implemented and invoked using this interface.

Table 115 on page 532 shows, for each of the three services provided, the messages implementing the JAVA methods invoked when performing actions and an explanation of the action you need to perform.

Table 115. Available messages in the Web services interface

Services	Messages	Actions performed
SchedulingFactory	submitJobStream	Submits job streams.
	submitJob	Submits jobs.
	submitAdHocJob	Submits jobs in production.
	queryJobs	Lists all job instances matching the filtering criteria.
	queryJobStreams	Lists all job stream instances matching the filtering criteria.
JobStreamService	getProperties	Displays job stream properties.
	setProperties	Sets specified properties for a job stream instance.
	getJobsList	Lists all jobs contained in a job stream.
	cancel	Cancel a job stream instance.
	releaseAllDependencies	Releases all dependencies for a job stream instance.
JobService	getProperties	Displays job properties.
	setProperties	Sets specified properties for a job instance.
	getOutput	Gets a job instance output.
	kill	Kills a job instance.
	cancel	Cancel a job instance.
	releaseAllDependencies	Releases all dependencies for a job instance.

For more information on how to manage WSDL files to create your own Web Services-based user interface you can refer to the IBM redbook *WebSphere® Version 5.1 Application Developer 5.1.1 Web Services Handbook*.

Appendix C. Quick reference for commands

This appendix is divided into four sections:

- “Managing the plan”
- “Managing objects in the database” on page 534
- “Managing objects in the plan” on page 541
- “Utility commands” on page 545
- “Report commands” on page 547

Managing the plan

This section describes the operations you can perform against the plan using the **JnextPlan** script and the **planman** command line:

Table 116. Commands used against the plan

Command or script syntax	Action performed
JnextPlan [-from <i>mm/dd/[yy]yy[hhmm[tz timezone tzname]]</i>] {-to <i>mm/dd/[yy]yy[hhmm[tz timezone tzname]]</i> -for [<i>h</i>]hhmm [-days <i>n</i>] -days <i>n</i> }	Creates or extends the production plan.
planman [<i>connection_parameters</i>] crt [-from <i>mm/dd/[yy]yy [hhmm [tz timezone tzname]]</i>] {-to <i>mm/dd/[yy]yy[hhmm[tz timezone tzname]]</i> -for [<i>h</i>]hhmm [-days <i>n</i>] -days <i>n</i> }	Creates an intermediate production plan.
planman [<i>connection_parameters</i>] deploy [-scratch]	Deploys all rules that are not in draft state.
planman [<i>connection_parameters</i>] ext {-to <i>mm/dd/[yy]yy[hhmm[tz timezone tzname]]</i> -for [<i>h</i>]hhmm [-days <i>n</i>] -days <i>n</i> }	Creates an intermediate plan for a plan extension.
planman [<i>connection_parameters</i>] showinfo	Retrieves the production plan information.
planman [<i>connection_parameters</i>] crtrial <i>file_name</i> [-from <i>mm/dd/[yy]yy [hhmm [tz timezone tzname]]</i>] {-to <i>mm/dd/[yy]yy[hhmm[tz timezone tzname]]</i> -for [<i>h</i>]hhmm [-days <i>n</i>] -days <i>n</i> }	Creates a trial plan.
planman [<i>connection_parameters</i>] exttrial <i>file_name</i> {-to <i>mm/dd/[yy]yy[hhmm[tz timezone tzname]]</i> -for [<i>h</i>]hhmm [-days <i>n</i>] -days <i>n</i> }	Creates a trial plan of a production plan extension.

Table 116. Commands used against the plan (continued)

Command or script syntax	Action performed
planman [<i>connection_parameters</i>] crtfc <i>file_name</i> [-from <i>mm/dd/[yy]yy [hhmm [tz timezone tzname]]</i>] { -to <i>mm/dd/[yy]yy[hhmm[tz timezone tzname]]</i> } -for [<i>h</i>] <i>hhmm [-days n] -days n</i>	Creates a forecast plan.
planman [<i>connection_parameters</i>] unlock	Unlocks the production plan.
ResetPlan [<i>connection_parameters</i>] [-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]
    
```

Managing objects in the database

The section is divided into the following subsections:

- “General purpose commands”
- “Scheduling objects” on page 535
- “Composer commands” on page 538

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 117. General purpose commands

Command	Syntax	User Authorization
continue	continue & <i>command argument</i> & <i>command argument</i>	Authorization for using composer
edit	edit <i>filename</i>	Authorization for using composer
exit	exit	Authorization for using composer
help	help <i>commandname</i>	Authorization for using composer
redo	redo <i>directives</i>	Authorization for using composer
validate	validate <i>filename</i> [; <i>syntax</i>]	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"}
streamlogon username
[description "description"]
[tasktype tasktype]
[interactive]1
[rcondsucc "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"]
[variable 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* are the following:

```
[-host hostname]
[-port port_name]
[-protocol protocol_name]
[-proxy proxy_name]
[-proxyport proxy_port_number]
[-password user_password]
[-timeout seconds]
[-username user_name]
| [-file filename]
```

These operations can only be run from any composer client command line installed.

In Table 118 on page 539 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 118. Commands used against scheduling objects in the database

Command	Syntax	User Authorization
add	add <i>filename</i> [<i>unlock</i>]	<i>add or modify</i>
authenticate	authenticate [<i>username=</i> <i>username</i> <i>password=</i> <i>password</i>]	
create extract	create extract <i>filename</i> from {[<i>calendars</i> <i>calendar</i> <i>cal=calname</i>] [<i>eventrule</i> <i>erule</i> <i>er=eventrulename</i>] [<i>parms</i> <i>parm</i> <i>vb=[tablename.]variablename</i>] [<i>variable</i> <i>vt=tablename</i>] [<i>prompts</i> <i>prom=promptname</i>] [<i>resources</i> <i>resource</i> <i>res=[workstationname#]resourcename</i>] [<i>cpu={workstationname workstationclassname domainname}</i>] [<i>workstation</i> <i>ws=workstationname</i>] [<i>workstationclass</i> <i>wscl=workstationclassname</i>] [<i>domain</i> <i>dom=domainname</i>] [<i>jobs</i> <i>jobdefinition</i> <i>jd=[workstationname#]jobname</i>] [<i>sched</i> <i>jobstream</i> <i>js= [workstationname#]jstreamname</i> [<i>validfrom</i> <i>date</i> <i>validto</i> <i>date</i> <i>validin</i> <i>date</i> <i>date</i>] [<i>full</i>]] [<i>users</i> <i>user=[workstationname#]username</i>]} [; <i>lock</i>]	<i>display</i>
delete	delete {[<i>calendars</i> <i>calendar</i> <i>cal=calname</i>] [<i>eventrule</i> <i>erule</i> <i>er=eventrulename</i>] [<i>parms</i> <i>parm</i> <i>vb=[tablename.]variablename</i>] [<i>variable</i> <i>vt=tablename</i>] [<i>prompts</i> <i>prom=promptname</i>] [<i>resources</i> <i>resource</i> <i>res=[workstationname#]resourcename</i>] [<i>cpu={workstationname workstationclassname domainname}</i>] [<i>workstation</i> <i>ws=workstationname</i>] [<i>workstationclass</i> <i>wscl=workstationclassname</i>] [<i>domain</i> <i>dom=domainname</i>] [<i>jobs</i> <i>jobdefinition</i> <i>jd=[workstationname#]jobname</i>] [<i>sched</i> <i>jobstream</i> <i>js= [workstationname#]jstreamname</i> [<i>validfrom</i> <i>date</i> <i>validto</i> <i>date</i> <i>validin</i> <i>date</i> <i>date</i>]] [<i>users</i> <i>user=[workstationname#]username</i>]} [; <i>noask</i>]	<i>delete</i>
display	display {[<i>calendars</i> <i>calendar</i> <i>cal=calname</i>] [<i>eventrule</i> <i>erule</i> <i>er=eventrulename</i>] [<i>parms</i> <i>parm</i> <i>vb=variablename.]variablename</i>] [<i>variable</i> <i>vt=tablename</i>] [<i>prompts</i> <i>prom=promptname</i>] [<i>resources</i> <i>resource</i> <i>res=[workstationname#]resourcename</i>] [<i>cpu={workstationname workstationclassname domainname}</i>] [<i>workstation</i> <i>ws=workstationname</i>] [<i>workstationclass</i> <i>wscl=workstationclassname</i>] [<i>domain</i> <i>dom=domainname</i>] [<i>jobs</i> <i>jobdefinition</i> <i>jd=[workstationname#]jobname</i>] [<i>sched</i> <i>jobstream</i> <i>js= [workstationname#]jstreamname</i> [<i>validfrom</i> <i>date</i> <i>validto</i> <i>date</i> <i>validin</i> <i>date</i> <i>date</i>] [<i>full</i>]] [<i>users</i> <i>user=[workstationname#]username</i>]} [; <i>offline</i>]	<i>display</i>

Table 118. Commands used against scheduling objects in the database (continued)

Command	Syntax	User Authorization
list print	list {[calendars calendar cal= <i>calname</i>] [eventrule erule er= <i>eventrulename</i>] [parms parm vb= <i>[tablename.]variablename</i>] [variable vt= <i>tablename</i>] [prompts prom= <i>promptname</i>] [resources resource res= <i>[workstationname#]resourcename</i>] [cpu= <i>{workstationname workstationclassname domainname}</i>] [workstation ws= <i>workstationname</i>] [workstationclass wscl= <i>workstationclassname</i>] [domain dom= <i>domainname</i>] [jobs jobdefinition jd= <i>[workstationname#]jobname</i>] [sched jobstream js= <i>[workstationname#]jstreamname</i> <i>[validfrom date </i> <i>validto date validin date date]</i> [users user= <i>[workstationname#]username</i>] [:offline]	<i>display</i>
lock	lock {[calendars calendar cal= <i>calname</i>] [eventrule erule er= <i>eventrulename</i>] [parms parm vb= <i>[tablename.]variablename</i>] [variable vt= <i>tablename</i>] [prompts prom= <i>promptname</i>] [resources resource res= <i>[workstationname#]resourcename</i>] [cpu= <i>{workstationname workstationclassname domainname}</i>] [workstation ws= <i>workstationname</i>] [workstationclass wscl= <i>workstationclassname</i>] [domain dom= <i>domainname</i>] [jobs jobdefinition jd= <i>[workstationname#]jobname</i>] [sched jobstream js= <i>[workstationname#]jstreamname</i> <i>[validfrom date validto date validin date date]</i>] [users user= <i>[workstationname#]username</i>]}	<i>modify</i>
modify	modify {[calendars calendar cal= <i>calname</i>] [eventrule erule er= <i>eventrulename</i>] [parms parm vb= <i>[tablename.]variablename</i>] [variable vt= <i>tablename</i>] [prompts prom= <i>promptname</i>] [resources resource res= <i>[workstationname#]resourcename</i>] [cpu= <i>{workstationname workstationclassname domainname}</i>] [workstation ws= <i>workstationname</i>] [workstationclass wscl= <i>workstationclassname</i>] [domain dom= <i>domainname</i>] [jobs jobdefinition jd= <i>[workstationname#]jobname</i>] [sched jobstream js= <i>[workstationname#]jstreamname</i> <i>[validfrom date validto date validin date date]</i> [:full]] [users user= <i>[workstationname#]username</i>]}	<i>modify or add</i>
new	new	<i>add or modify</i>

Table 118. Commands used against scheduling objects in the database (continued)

Command	Syntax	User Authorization
rename	rename {calendars calendar cal eventrule erule er parms parm vartable vt prompts prom resources resource res workstation ws workstationclass wscl domain dom jobs jobdefinition jd jobsched jb sched jobstream js users user } old_object_identifier new_object_identifier	add and delete
replace	replace filename [;unlock]	modify or add
unlock	unlock {[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 [validfrom date validto date validin date date]] [users user=[workstationname#]username]} [;forced]	modify and unlock
validate	validate 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.

Conman commands

This section lists the commands you can run from the **conman** program.

Accessing the command line

This is how you access the **conman** command line:

```
conman [connection_parameters] ["command[&[command]...] [&]"]
```

where *connection_parameters* are the following:

```
[-host hostname]  

[-port port_name]  

[-protocol protocol_name]  

[-proxy proxy_name]  

[-proxyport proxy_port_number]
```

```
[-password user_password]
[-timeout seconds]
[-username user_name]
| [-file filename]
```

Selecting jobs in commands

This is how you select jobs in commands:

```
[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}
```

Selecting job streams in commands

This is how you select job streams in commands:

```
[workstation#]
jobstreamname(hhmm[ date])
[{|~}jobstreamqualifier[...]]
```

or:

```
[workstation#]
jobstream_id
;schedid
```

Command syntax summary

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 119. Commands that can be run from conman

Command	Syntax	Workstation types	User Authorization
adddep job	adj jobselect[;dependency[;...]]];noask]	F	adddep - (use when using prompts and needs)
adddep sched	ads jstreamselect[;dependency[;...]]];noask]	F	adddep - (use when using prompts and needs)
altpass	altpass [workstation#]username["password"]	F	altpass

Table 119. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
altpri	ap <i>jobselect</i> <i>jstreamselect</i> [:pri][;noask]	F	<i>altpri</i>
bulk_discovery	bulk	F	<i>display</i>
cancel job	cj <i>jobselect</i> [:pend][;noask]	F	<i>cancel</i>
cancel sched	cs <i>jstreamselect</i> [:pend][;noask]	F	<i>cancel</i>
1 checkhealthstatus	chs [<i>workstation</i>]	M,E,S	
confirm	confirm <i>jobselect</i> [:succ abend][;noask]	F	<i>confirm</i>
console	console [<i>sess</i> <i>sys</i>][;level= <i>msglevel</i>]	F-S	<i>console</i>
continue	conman " continue &command <i>argument</i> &command <i>argument</i> "	F-S	
deldep job	ddj <i>jobselect</i> [:dependency[:...]][;noask]	F	<i>deldep</i>
deldep sched	dds <i>jstreamselect</i> [:dependency[:...]][;noask]	F	<i>deldep</i>
deployconf	deployconf [<i>domain!</i>] <i>workstation</i>	E,S	Permission to start actions on <i>cpu</i> objects
display	df <i>filename</i> [:offline] dj <i>jobselect</i> [:offline] ds <i>jstreamselect</i> [valid { at <i>date</i> in <i>date date</i> }][:offline]	F-S ¹	<i>display</i>
exit	e [<i>xit</i>]	F-S	
fence	fence <i>workstation</i> [:pri][;noask]	F	<i>fence</i>
help (UNIX only)	help <i>command</i>	F-S	
kill	kill <i>jobselect</i> [:noask]	F	<i>kill</i>
limit cpu	lc <i>workstation</i> [:limit][;noask]	F	<i>limit</i>
limit sched	ls <i>jstreamselect</i> [:limit][;noask]	F	<i>limit</i>
link	lk [<i>domain!</i>] <i>workstation</i> [:noask]	F-S	<i>link</i>
listsym	listsym [trial forecast] [:offline]	F	
recall	rc [<i>workstation</i>][:offline]	F	<i>display</i>
redo	redo	F-S	
release job	rj <i>jobselect</i> [:dependency[:...]][;noask]	F	<i>release</i>
release sched	rs <i>jstreamselect</i> [:dependency[:...]][;noask]	F	<i>release</i>
reply	reply { <i>promptname</i> [<i>workstation#</i>] <i>msgnum</i> }; <i>reply</i> [:noask]	F	<i>reply</i>
rerun	rr <i>jobselect</i> [: from =[<i>wkst</i> at#] <i>job</i> [: at = <i>time</i>] [: pri = <i>pri</i>]] [:noask] rr <i>jobselect</i> [: step = <i>step</i>][;noask]	F	<i>rerun</i>
resource	resource [<i>workstation#</i>] <i>resource</i> [:num][;noask]	F	<i>resource</i>
setsym	setsym [trial forecast] [<i>file</i> num]	F	
showcpus	sc [[<i>domain!</i>] <i>workstation</i>][: getmon][: info link] [:offline]	F-S	<i>list</i> ²
showdomain	showdomain [<i>domain</i>][: info][:offline]	F-S	<i>list</i> ²
showfiles	sf [[<i>workstation#</i>] <i>file</i>][: state [:...]][; keys] [:offline] sf [[<i>workstation#</i>] <i>file</i>][: state [:...]][; deps [: keys info logon][:offline]	F	

Table 119. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
showjobs	sj [<i>jobselect</i>] [<i>;keys info step logon crit keys retcod</i>]; <i>short single</i> ; <i>offline</i>] [<i>;showid</i>] sj [<i>jobselect</i>] [<i>;deps</i> ; <i>keys info logon</i>]]; <i>short single</i> ; <i>offline</i>] [<i>;showid</i>] sj [<i>jobselect [workstation#]jobnumber.hhmm</i>] [<i>;stdlist</i> ; <i>keys</i>]]; <i>short single</i> ; <i>offline</i>] [<i>;showid</i>]	F	<i>list</i> ²
showprompts	sp [<i>promptselect</i>]; <i>keys</i> ; <i>offline</i>] sp [<i>promptselect</i>] [<i>;deps</i> ; <i>keys info logon</i>]]; <i>offline</i>]	F	<i>list</i> ²
showresources	sr [[<i>workstation#</i>] <i>resourcename</i>]; <i>keys</i> ; <i>offline</i>] sr [[<i>workstation#</i>] <i>resourcename</i>]; <i>deps</i> ; <i>keys info logon</i>]] [<i>;offline</i>]	F	<i>list</i> ²
showschedules	ss [<i>jstreamselect</i>]; <i>keys</i> ; <i>offline</i>]; <i>showid</i>] ss [<i>jstreamselect</i>]; <i>deps</i> ; <i>keys info logon</i>]]; <i>offline</i>] [<i>;showid</i>]	F	<i>list</i> ²
shutdown	shut [<i>down</i>] [<i>;wait</i>]	F-S	<i>shutdown</i>
start	start [<i>domain!</i>] <i>workstation</i> ; <i>mgr</i>]; <i>noask</i>]; <i>demgr</i>]	F-S	<i>start</i>
startappserver	startappserver [<i>domain!</i>] <i>workstation</i> [<i>;wait</i>]	F-S	Permission to <i>start</i> actions on <i>cpu</i> objects
startevtp	starteventprocessor [<i>domain!</i>] <i>workstation</i>	M ⁴	Permission to <i>start</i> actions on <i>cpu</i> objects
startmon	startmon [<i>domain!</i>] <i>workstation</i> [<i>;noask</i>]	F-S	Permission to <i>start</i> actions on <i>cpu</i> objects
status	status	F-S	appserver
stop	stop [<i>domain!</i>] <i>workstation</i> ; <i>wait</i>]; <i>noask</i>]	F-S	<i>stop</i>
stop ;progressive	stop ;progressive		<i>stop</i>
stopappserver	stopappserver [<i>domain!</i>] <i>workstation</i> [<i>;wait</i>]	F-S	Permission to <i>stop</i> actions on <i>cpu</i> objects
stopevtp	stopeventprocessor [<i>domain!</i>][<i>workstation</i>]	M ⁴	Permission to <i>stop</i> actions on <i>cpu</i> objects
stopmon	stopmon [<i>domain!</i>] <i>workstation</i> [<i>;wait</i>] [<i>;noask</i>]	F-S	Permission to <i>stop</i> actions on <i>cpu</i> objects
submit docommand	sbd [<i>workstation#</i>]" <i>cmd</i> "[<i>;alias</i> [= <i>name</i>]] [<i>;into</i> =[<i>workstation#</i>]{ <i>jobstream_id</i> ; <i>schedid</i> <i>jobstreamname</i> (<i>hhmm</i> [<i>date</i>)]}] [<i>;joboption</i> [<i>;</i> ...]]]	F-S	<i>submit</i> - (use when using prompts and needs)
submit file	sbf " <i>filename</i> "[<i>;alias</i> [= <i>name</i>]] [<i>;into</i> =[<i>workstation#</i>]{ <i>jobstream_id</i> ; <i>schedid</i> <i>jobstreamname</i> (<i>hhmm</i> [<i>date</i>)]}] [<i>;joboption</i> [<i>;</i> ...]]]; <i>noask</i>]	F-S	<i>submit</i> - (use when using prompts and needs)

Table 119. Commands that can be run from conman (continued)

Command	Syntax	Workstation types	User Authorization
submit job	sbj [<i>workstation#</i>]{ <i>jobname</i> [: <i>alias</i> [= <i>name</i>]] [: <i>into</i> =[<i>workstation#</i>]{ <i>jobstream_id</i> ; <i>schedid</i> <i>jobstreamname</i> (<i>hmm</i> [<i>date</i>)]}] [<i>joboption</i> [:...]][: noask]	F-S ³	<i>submit</i> - (use when using prompts and needs)
submit sched	sbs [<i>workstation#</i>]{ <i>jstreamname</i> [: <i>alias</i> [= <i>name</i>]] [: <i>jstreamoption</i> [:...]][: noask]	F-S ³	<i>submit</i> - (use when using prompts and needs)
switchevtp	switcheventprocessor <i>workstation</i>	M ⁴	Permission to <i>start</i> and <i>stop</i> actions on <i>cpu</i> objects
switchmgr	switchmgr <i>domain</i> ; <i>newmgr</i>	F	<i>start stop</i>
<i>system</i>	[: !] <i>sys-command</i>	F-S	
tellop	to [<i>text</i>]	F-S	
unlink	unlink [<i>domain</i> !]{ <i>workstation</i> [: noask]	F-S	<i>unlink</i>
version	v [<i>ersion</i>]	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 120. Utility commands available for both UNIX and Windows

Command	Syntax
at	at -V -U at -s <i>jstream</i> -q <i>queue time-spec</i>
batch	batch -V -U batch [-s <i>jstream</i>]
cpuinfo	cpuinfo -V -U cpuinfo <i>workstation</i> [<i>infotype</i>] [...]

Table 120. Utility commands available for both UNIX and Windows (continued)

Command	Syntax
datecalc	datecalc -V -U datecalc <i>base-date</i> [<i>offset</i>] [pic <i>format</i>][freedays <i>Calendar_Name</i> [-sa] [-su]] datecalc -t <i>time</i> [<i>base-date</i>] [<i>offset</i>] [pic <i>format</i>] datecalc <i>yyyymmddhhht</i> [<i>offset</i>] [pic <i>format</i>]
delete	delete -V -U delete <i>filename</i>
evtdef	evtdef -U -V evtdef [<i>connection parameters</i>] dumpdef <i>file-path</i> evtdef [<i>connection parameters</i>] loaddef <i>file-path</i>
evtsize	evtsize -V -U evtsize <i>filename</i> <i>size</i> evtsize -compact <i>filename</i> [<i>size</i>] evtsize -info <i>filename</i> evtsize -show <i>filename</i> evtsize -info -show pobox
jobinfo	jobinfo -V -U jobinfo <i>job-option</i> [...]
jobstdl	jobstdl -V -U jobstdl [<i>-day num</i>] [{ <i>-first</i> <i>-last</i> <i>-num n</i> <i>-all</i> }] [<i>-twslog</i>] [{ <i>-name</i> [" <i>jobstreamname</i> [(<i>hhmm date</i>),(<i>jobstream_id</i>)]. <i>jobname</i> " <i>jobnum</i> <i>-schedid jobstream_id.jobname</i>]}]
maestro	maestro [-V -U]
makecal	makecal [<i>-c name</i>] <i>-d n</i> <i>-e</i> { <i>-f 1 2 3 -s date</i> } <i>-l</i> <i>-m</i> <i>-p n</i> { <i>-r n -s date</i> } <i>-w n</i> [<i>-i n</i>] [<i>-x</i> <i>-z</i>][<i>-freedays Calendar_Name</i> [-sa] [-su]]
metronome.pl	Is replaced by tws_inst_pull_info .
morestdl	morestdl -V -U morestdl [<i>-day num</i>] [<i>-first</i> <i>-last</i> <i>-num n</i> <i>-all</i>] [<i>-twslog</i>] [{ <i>-name</i> [" <i>jobstreamname</i> [(<i>hhmm date</i>),(<i>jobstream_id</i>)]. <i>jobname</i> " <i>jobnum</i> <i>-schedid jobstream_id.jobname</i>]}]
parms	parms [{ <i>-V</i> <i>-U</i>] <i>-build</i> } parms { <i>-replace</i> <i>-extract</i> } <i>filename</i> parms [<i>-d</i>] <i>parametername</i> parms -c <i>parametername value</i>
release	release -V -U release [<i>-s</i>] [<i>workstation#</i>] <i>resourcename</i> [<i>count</i>]
rmstdlist	rmstdlist -V -U rmstdlist [<i>-p</i>] [<i>age</i>]

1

1

1

Table 120. Utility commands available for both UNIX and Windows (continued)

Command	Syntax
sendevent	sendevent -V ? -help -U -usage sendevent [-hostname <i>hostname</i>][-port <i>port</i>] <i>eventType source</i> [[<i>attribute=value</i>]...]
showexec	showexec [-V -U INFO]
StartUp	StartUp [-V -U]
twins_inst_pull_info	twins_inst_pull_info -twinsuser <i>userid</i> -log_dir_base <i>path</i> [-u [-run_db2_module [y n] -extract_db_defs [y n] -date <i>yyyymmdd</i>]

Utility commands available for UNIX operating system only

Table 121. Utility commands available for UNIX only

Command	Syntax
at	at -V -U at -s<i>stream</i> -<i>queue</i>time-spec
batch	batch -V -U batch [-s <i>stream</i>]
showexec	showexec [-V -U -info]
version	version -V -U -h version [-a] [-f <i>vfile</i>] [<i>file</i> [...]]

Utility commands available for Windows operating system only

Table 122. Utility commands available for Windows only

Command	Syntax
listproc	listproc
(UNSUPPORTED)	
killproc	killproc <i>pid</i>
(UNSUPPORTED)	
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 123. 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]

Table 123. Report commands (continued)

Name	Output produced	Syntax
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 <i>wkstat</i>] [-s <i>jstream_name</i>] [-j <i>job</i>] [-f <i>date</i> -t <i>date</i>] [-l]
rep8	Report 08 - Job Histogram	rep8 -V -U rep8 [-f <i>date</i> -b <i>time</i> -t <i>date</i> -e <i>time</i>] [-i <i>file</i>] [-p] rep8 [-b <i>time</i> -e <i>time</i>] [-i <i>file</i>] [-p]
rep11	Report 11 - Planned Production Schedule	rep11 -V -U rep11 [-m <i>mm[yy]</i> [...]] [-c <i>wkstat</i> [...]] [-s <i>jstream_name</i>] [-o <i>output</i>]
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}] [<i>symfile</i>] reptr -post [-{summary detail}] [<i>logfile</i>]
xref	Report 12 - Cross Reference Report	xref [-V -U] xref [-cpu <i>wkstat</i>] [-s <i>jstream_name</i>] [-depends -files -jobs -prompts -resource -schedules -when [...]]

Report extract programs

Table 124. Report extract programs

Extract Program	Used to generate	Syntax
jbextract	Report 01 Report 07	jbextract [-V -U] [-j <i>job</i>] [-c <i>wkstat</i>] [-o <i>output</i>]
prxtract	Report 02	prxtract [-V -U] [-o <i>output</i>] prxtract [-V -U] [-m <i>mm[yyyy]</i>] [-c <i>wkstat</i>] [-o <i>output</i>]
caxtract	Report 03	caxtract [-V -U] [-o <i>output</i>]
paxtract	Report 04A	paxtract [-V -U] [-o <i>output</i>]
retract	Report 04B	retract [-V -U] [-o <i>output</i>]
r11xtr	Report 11	r11xtr [-V -U] [-m <i>mm[yyyy]</i>] [-c <i>wkstat</i>] [-o <i>output</i>] [-s <i>jstream_name</i>]

Table 124. Report extract programs (continued)

Extract Program	Used to generate	Syntax
xrxtct	Report 12	xrxtct [-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

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

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

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

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

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

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

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

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

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

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

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

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

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

The licensed program described in this 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.

Intel, and Itanium, are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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



Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

Other company, product, and service names might be trademarks or service marks of others.

Index

Special characters

.jobmanrc configuration script 39

\$MASTER keyword 126

A

abend

job state 266

job stream state 273

abend prompt 3

abenp

job state 266

access

extended and network agents 127

workstation definition 127

accessibility xiii, 551

ad-hoc prompt 3

add

job state 266

job stream state 273

add command 213

adddep job command 277

adddep sched command 278

altpass command 280

altpri command 281

architecture 21

at command 376

ATSCRIPT variable 376

at keyword 153, 157

auditing

body format 469

enabling 467

feature 467

header format 468

log format 468

overview 467

sample log file 472

authenticate command 215

autolink

workstation definition 127

automating plan processing

final job stream 82

automating processing

production plan 82

autostart monman 355

average run time 81

B

backup master domain manager 4

batch command 376

batchman process 22

behindfirewall

workstation definition 128

broker

workstation definition 127

bulk_discovery command 282

C

calendar

freedays 3

holidays 3

run cycle 3, 174

calendar definition 143

cancel command 282

cancel sched command 284

carryforward

customizing 56

job stream keywords 56

stageman 56

variable 77

carryforward keyword 158

carryStates

variable 56, 60

caxtract command 429

checkhealthstatus command 285

checking mailbox health 285

class

workstation 5

closest preceding

follows 165

follows previous 46

matching criteria 46

command

logman 79

stageman 76

command line

composer 15

conman 15

optman 15

command line interface

setting 41

commands

adddep job 277

adddep sched 278

altpass 280

altpri 281

at 376

batch 376

bulk_discovery 282

cancel job 282

cancel sched 284

caxtract 429

checkhealthstatus 285

confirm 286

console 287

continue 288

cpuinfo 379

datecalc 380

deldep job 288

deldep sched 290

delete 384

deployconf 291

display 292

evtdef 385

evtsize 388

exit 294

fence 295

getmon 313

commands (*continued*)

help 296

jbextract 427

jobinfo 389

jobstdl 391

kill 297

limit cpu 297

limit sched 299

link 299

listsym 302

maestro 394

makecal 394

metronome 396, 406

morestdl 396

parms 398

paxtract 430

prxtract 429

r11xtr 431

recall 303

redo 304

release 400

release job 305

release sched 307

rep1 410

rep11 414

rep2 410

rep3 410

rep4a 410

rep4b1 410

rep7 411

rep8 413

reply 308

reprr 415

rerun 309

resource 312

rextract 431

rmstdlist 401

sendevent 402

setsym 313

showcpus 313

showdomain 319

showexec 403

showfiles 320

showjobs 322

showprompts 335

showresources 337

showschedules 340

shutdown 345

start 345

startappserver 348

starteventprocessor 348

startmon 349

StartUp 405

status 349

stop 350

stop ;progressive 352

stopappserver 353

stopeventprocessor 354

stopmon 355

submit docommand 355

submit file 359

- commands (*continued*)
 - submit job 362
 - submit sched 365
 - switcheventprocessor 368
 - switchmgr 369
 - tellop 370
 - unlink 371
 - version 374, 406
 - xref 416
 - xrtrct 432
- comment keyword 158
- composer
 - command line 15
- composer program 199
 - connection parameters 201
 - control characters 203
 - delimiters 207
 - editor 200
 - filters 204
 - list of commands 207
 - offline output 199
 - prompt 201
 - running commands 201
 - setup 199
 - variables 409
 - variables on UNIX 200
 - variables on Windows 199
 - special characters 207
 - terminal output 199
 - wild cards 204
 - XML editor 200
- composer reference 119
- COMPUTERNAME variable 34
- configuration scripts
 - .jobmanrc 39
 - jobmanrc 37
 - jobmanrc.cmd 41
- configuring
 - local properties 33
- confirm command 286
- confirmed keyword 159
- conman
 - command line 15
- conman program 253
 - control characters 257
 - delimiters 258
 - list of commands 274
 - offline output 254
 - processing 259
 - prompt 254
 - running commands 254, 257
 - selecting job 259
 - arguments 260
 - jobstream_ID 260
 - jobstreamname 260
 - schedid 260
 - using at 261
 - using confirm 262
 - using critical 262
 - using critnet 262
 - using deadline 262
 - using every 262
 - using finished 263
 - using follows 263
 - using logon 264
 - using needs 264
 - using opens 264
- conman program (*continued*)
 - selecting job (*continued*)
 - using priority 265
 - using prompt 265
 - using recovery 265
 - using scriptname 265
 - using started 265
 - using state 266
 - using until 267
 - selecting job streams 267
 - arguments 268
 - jobstream_id 268
 - jobstreamname 268
 - schedid 268
 - using at 269
 - using carriedforward 269
 - using carryforward 269
 - using finished 269
 - using follows 270
 - using limit 271
 - using needs 271
 - using opens 271
 - using priority 272
 - using prompt 272
 - using started 272
 - using state 273
 - using until 273
 - setup 253
 - set variables 253
 - special characters 258
 - terminal output 253
 - user prompting 257
 - wildcards 258
- conman reference 253
- conman startappserver
 - JnextPlan 65
- connection parameters
 - setting 41
- console command 287
- continue command 215, 288
- conventions used in publications xiii
- conventions, typeface xiv
- Courier.msg 28
- cpuclass
 - workstation class definition 132
- cpuinfo command 379
- cpuname
 - workstation definition 125
- create command 216
- CreatePostReports
 - JnextPlan 66
- creating forecast
 - planman command line 73
- creating trial
 - planman command line 71
- critical jobs
 - global options 86
 - local options 88
 - security file 89
- critical keyword 159
- custom events
 - defining 116, 385
 - sending 116, 402
- customizing the workload
 - using variable table 95

D

- daily run cycle 2
- data integrity
 - variable table 97
- database objects
 - add command 213
 - authenticate command 215
 - calendars 143
 - continue command 215
 - create command 216
 - delete command 219
 - display command 223
 - displaying composer banner 252
 - domains 133
 - edit command 227
 - event rules 188
 - exit command 228
 - extract command 216
 - help command 228
 - job stream 152
 - jobs 134
 - list command 229
 - lock command 233
 - modify command 237
 - new command 241
 - print command 229
 - prompts 3, 149
 - redo command 243
 - rename command 244
 - replace command 247
 - resources 151
 - unlock command 248
 - validate command 251
 - variable table 147
 - variables 144
 - windows users 141
 - workstation classes 131
 - workstations 121
- date
 - run cycle 173
- datecalc command 380
- day
 - run cycle 173
- deadline keyword 160
- default variable table
 - using 96
- defining
 - database objects
 - calendars 143
 - domains 133
 - event rules 188
 - job stream 152
 - jobs 134
 - prompts 3, 149
 - resources 151
 - variables 144
 - windows users 141
 - workstation classes 131
 - workstations 121
 - dependencies
 - follows 165
 - needs 172
 - opens 178
 - prompts 181
- Defining objects
 - in the database 119

- definition
 - variable table 147
- deldep job command 288
- deldep sched command 290
- delete command 219, 384
- dependencies
 - orphaned 50
- dependency
 - internetwork 485, 490
- deployconf command 291
- Deploying rules
 - planman command line 74
- description keyword 161
- directory names, notation xv
- disk filling
 - monitoring 524
- disk full
 - monitoring 524
- disk space
 - monitoring 524
- display command 223, 292
- docommand
 - job definition 136
- domain 6
 - workstation definition 126
- domain definition 133
 - ismaster 133
 - manager 133
 - parent 133
- domain manager 4
- done
 - job state 266
- draft keyword 161
- dynamic jobs 121
- dynamic scheduling 121
- dynamic workstations 121

E

- edit command 227
- education xiii
- enabling
 - SSL communication 128
 - time zone 445
- enCarryForward
 - variable 56, 60
- enCFInterNetworkDeps
 - variable 60
- enCFResourceQuantity
 - variable 61
- end keyword 161
- enLegacyId
 - variable 61
- enLegacyStartOfDayEvaluation
 - variable 63, 446
- enPreventStart
 - variable 61
- enTimeZone
 - variable 63, 445
- environment variables, notation xv
- error
 - job state 266
- estimated run time 81
- event rule 9
- event rule definition 188
 - eventRules.xsd 189

- event rule definition (*continued*)
 - keywords
 - actionProvider 194
 - actionType 194
 - activeTime 191
 - correlationAttributes 194
 - daylight saving time 191
 - description 191, 195
 - eventCondition 192
 - eventProvider 192
 - eventRule 190
 - eventType 192
 - filteringPredicate 193
 - isDraft 191
 - name 190
 - onDetection 195
 - onTimeOut 195
 - operator 193
 - responseType 195
 - ruleType 190
 - scope 193, 196
 - timeInterval 192
 - timeZone 191
 - validity 191
- event rules
 - instances 116
 - sample scenarios 102, 109
 - timeout option 108
 - variable substitution 109
- every keyword 162
- evtdef command 385
- evtsize command 388
- except keyword 164
- exclusive run cycle 2
- exec
 - job state 266
 - job stream state 273
- exec+
 - job state 266
- exit command 228, 294
- extended agent
 - access method 476
 - option file 479
 - response messages 478
 - running 480
 - syntax 476
 - troubleshooting 483
- definition 476
- overview 475
- reference 475
- workstation definition 127

EXTERNAL

- job stream 490
- jobs 490

extract command 216

extrn

- job state 266

F

- fail
 - job state 266
- fault-tolerant agent 4
- fdignore
 - except 164
 - on 176

- fdnext
 - except 164
 - on 176
- fdprev
 - except 164
 - on 176
- fence
 - job state 266
- fence command 295
- files
 - at.allow 378
 - at.deny 378
 - Courier.msg 28
 - Intercom.msg 28
 - Mailbox.msg 28
 - NetReq.msg 28
- filling mailboxes
 - monitoring 523
- filling message queues
 - monitoring 523
- filters
 - composer 204
- final job stream
 - automating plan processing 82
- FNCJSI
 - preproduction plan 45
- follows
 - matching criteria 165
- follows absolute to
 - matching criteria 47
 - within an absolute interval 47
- follows keyword 165
- follows previous
 - closest preceding 46
 - matching criteria 46
- follows relative to
 - matching criteria 47
 - within a relative interval 47
- follows sameday
 - matching criteria 48
 - same day 48
- forecast plan
 - creating 73
 - description 59
- freedays keyword 167
- freedays run cycle 2
- fta
 - workstation definition 127
- fullstatus
 - workstation definition 128

G

- global option
 - database audit variable 467
 - plan audit variable 467
- global options
 - carryforward variable 77
 - carryStates 56
 - carryStates variable 60
 - enCarryForward variable 56, 60
 - enCFInterNetworkDeps variable 60
 - enCFResourceQuantity variable 61
 - enLegacyId variable 61
 - enLegacyStartOfDayEvaluation
 - variable 63, 446
 - enPreventStart variable 61

- global options (*continued*)
 - enTimeZone variable 63, 445
 - logmanMinMaxPolicy variable 62
 - logmanSmoothPolicy variable 62
 - maxLen variable 60
 - minLen variable 60
 - startOfDay variable 60, 446
- global parameter
 - definition 144
 - variable table 95
- global prompt 3
- glossary xiii

H

- help command 228, 296
- hold
 - job state 266
 - job stream state 273
- HOME variable 34, 35, 41
- HOMEDRIVE variable 34
- HOMEPATH variable 34
- host
 - extended agents 126
 - workstation definition 126

I

- icalendar
 - run cycle 174
- identifying job stream instances
 - in the plan 46
 - at 46
 - scheddateandtime 46
 - preproduction plan 46
- ignore
 - workstation class definition 132
- inclusive run cycle 2
- Integration with IBM Tivoli Monitoring 6.1
 - bulk_discovery 282
- integrity of data
 - variable table 97
- interactive
 - job definition 137
- Intercom.msg 28
- intermediate plan
 - extending with 69
 - generating 68
- internetwork dependency
 - creating 489
 - managing using conman 490
- intro
 - job state 266
- intro+
 - job state 267
- ismaster
 - domain definition 133

J

- jbextract command 427
- JnextPlan
 - conman startappserver 65
 - CreatePostReports 66
 - MakePlan 65

- JnextPlan (*continued*)
 - SwitchPlan 65
 - UpdateStats 66
- job 1
 - calculating run time 81
- job definition 134
 - docommand 136
 - interactive 137
 - recovery option 138
 - scriptname 135
 - streamlogon 136
 - success condition 137
 - tasktype 136
 - using variables and parameters 140
- Job Scheduling Console
 - accessibility xiii
- job statement
 - in job streams 169
- job states
 - abend 266
 - abnsp 266
 - add 266
 - done 266
 - error 266
 - exec 266
 - exec+ 266
 - extrn 266
 - fail 266
 - fence 266
 - hold 266
 - intro 266
 - intro+ 267
 - pend 267
 - ready 267
 - sched 267
 - succ 267
 - succp 267
 - wait 267
- job stream 2
 - calculating run time 81
 - EXTERNAL 490
- job stream definition 152
- job stream keywords
 - at 157
 - carryforward 56, 158
 - comment 158
 - confirmed 159
 - critical 159
 - deadline 160
 - description 161
 - draft 161
 - end 161
 - every 162
 - except 164
 - follows 165
 - freedays 167
 - job statement 169
 - keyjob 170
 - keysched 170
 - limit 171
 - matching 171
 - needs 172
 - on 173
 - opens 178
 - priority 180
 - prompt 181
 - schedtime 182

- job stream keywords (*continued*)
 - schedule 183
 - timezone 184
 - until 185
 - validfrom 187
 - vartable 188
- job stream states
 - abend 273
 - add 273
 - exec 273
 - hold 273
 - ready 273
 - stuck 273
 - succ 273
- job streams keywords
 - at 153
 - onuntil 185
- jobinfo command 389
- jobman
 - environment variables 34
- jobman process 23
 - limit cpu 23
- jobmanrc configuration script 37, 41
- jobstdl command 391, 396

K

- keyjob keyword 170
- keysched keyword 170
- keywords
 - at 153, 157
 - carryforward 158
 - comment 158
 - confirmed 159
 - critical 159
 - deadline 160
 - description 161
 - draft 161
 - end 161
 - every 162
 - except 164
 - follows 165
 - freedays 167
 - keyjob 170
 - keysched 170
 - limit 171
 - matching 171
 - needs 172
 - on 173
 - opens 178
 - priority 180
 - prompt 181
 - schedtime 182
 - schedule 183
 - timezone 184
 - until 185
 - validfrom 187
 - vartable 188
- kill command 297

L

- LANG variable 34, 35
- late status 160
- LD_LIBRARY_PATH variable 35
- LD_RUN_PATH variable 35

- limit cpu
 - jobman process 23
- limit cpu command 297
- limit keyword 171
- limit sched command 299
- link command 299
- list command 229
- listsym command 302
- local option
 - mm retry link variable 82
- local parameter
 - database 398
 - definition 144
 - exporting 398
 - importing 398
 - managing 398
- local prompt 3
- local properties 33
- LOCAL_RC_OK variable 37
- lock command 233
- lock mechanism
 - variable table 97
- logging job statistics 79
- logical resource 9
- logmanMinMaxPolicy
 - variable 62
- logmanSmoothPolicy
 - variable 62
- LOGNAME variable 34, 35
- long term plan
 - preproduction plan 44

M

- maestro command 394
- MAESTRO_OUTPUT_STYLE
 - variable 34, 35
- MAIL_ON_ABEND variable 38
- mailbox files
 - Courier.msg 28
 - Intercom.msg 28
 - Mailbox.msg 28
 - NetReq.msg 28
 - setting size 388
- mailbox full
 - monitoring 523
- Mailbox.msg 28
- mailboxes
 - monitoring 523
- mailman process 22
 - ServerID 22
- makecal command 394
- MakePlan
 - JnextPlan 65
- manager
 - workstation definition 127
- managing
 - external follows dependencies 46
 - matching criteria 46
 - objects in the database 119
 - production cycle 43
- managing events
 - starting the event processing server 348
 - starting the monitoring engine 349
 - stopping the event processing server 354

- managing events (*continued*)
 - stopping the monitoring engine 355
 - switching the event processing server 368
- managing objects
 - in plan 253
 - in the database 199
- managing plan
 - adding dependency to job streams 278
 - adding dependency to jobs 277
 - altering priority 281
 - altering user password 280
 - assigning console 287
 - cancelling job streams 284
 - cancelling jobs 282
 - confirming job completion 286
 - deleting dependency to job streams 290
 - deleting dependency to jobs 288
 - displaying conman banner 374
 - displaying help information 296
 - displaying jobs or job streams 292
 - displaying production plan status 349
 - displaying workstation information 313
 - exiting conman 294
 - get active monitors 313
 - ignoring command 288
 - limiting jobs running in job stream 299
 - linking workstations 299
 - listing processed plans 302
 - listing unresolved prompts 303
 - modifying job fence 295
 - modifying jobs running on workstation 297
 - modifying resource units 312
 - releasing job streams from dependency 307
 - releasing jobs from dependency 305
 - replying to prompts 308
 - requesting a bulk_discovery 282
 - rerunning commands 304
 - rerunning jobs 309
 - selecting processed plan 313
 - sending messages to operator 370
 - setting message level 287
 - showing domain information 319
 - showing file dependencies 320
 - showing job information 322
 - showing job streams information 340
 - showing prompts information 335
 - showing resource information 337
 - shutting down workstation processes 345
 - starting the application server 348
 - starting workstation processes 345
 - stopping behindfirewall workstation processes 352
 - stopping jobs 297
 - stopping the application server 353
 - stopping workstation processes 350
 - submitting commands as jobs 355
 - submitting file as jobs 359
 - submitting job streams 365

- managing plan (*continued*)
 - submitting jobs 362
 - switching domain management 369
 - unlinking workstations 371
 - updating the monitoring configuration file 291
- managing time zone 445
 - time zone name
 - with variable length 445, 450
 - time zone table
 - all supported 450
 - backward compatibility 449
- master domain manager 4
- matching criteria
 - closest preceding 46, 171
 - follows 165
 - follows absolute to 47
 - follows previous 46
 - follows relative to 47
 - follows sameday 48
 - pending predecessor 49
 - predecessor 49
 - same day 48, 171
 - successor 49
 - within a relative interval 47, 171
 - within an absolute interval 47, 171
- matching keyword 171
- maxLen
 - variable 60
- mechanism of lock
 - variable table 97
- members
 - workstation class definition 132
- message queues
 - monitoring 523
- metronome command 396, 406
- minLen
 - variable 60
- modify command 237
- monman process 22

N

- named prompt 3
- needs keyword 172
- netman process 22
- netmth access method 487
- NetReq.msg 28
- network agent
 - access method
 - options file 487
 - access method netmth 487
 - definition 487
 - EXTERNAL 490
 - EXTERNAL state
 - ERROR 490
 - EXTRN 490
 - internetwork dependency 485
 - creating 489
 - managing using conman 490
 - overview 485
 - reference 485
 - sample scenario 487
- network communication 29
 - job processing 29
 - start of day 29
- new command 241

notation
 environment variables xv
 path names xv
 typeface xv

O

offset-based run cycle 2
on
 run cycle 173
on keyword 173
 run cycle 173
onuntil keyword 185
opens keyword 178
optman
 command line 15
orphaned
 dependencies 50
os type
 workstation definition 125

P

parameter 9
parameter definition 144
parameters
 in job definitions 140
parent
 domain definition 133
parms command 398
path names, notation xv
PATH variable 35
paxtract command 430
pend
 job state 267
pending predecessor
 matching criteria 49
 orphaned dependencies 50
 successor 49
physical resource 9
plan
 quick start 16
plan management
 basic concepts 43
 customizing 56, 60
 logman 79
 stageman 76
planman command line
 connection parameters 66
 creating forecast 73
 creating trial 71
 Deploying rules 74
 intermediate plan 68, 69
 resetting plan 75
 retrieving plan info 70
 trial extension 72
 unlocking plan 75
POSIXHOME variable 41
predecessor
 matching criteria 49
 successor 45, 49
preproduction plan
 description 44
 FNCJSI 45
 long term plan 44
print command 229

priority keyword 180
processes
 batchman 22
 jobman 23
 mailman 22
 monitoring 524
 monman 22
 netman 22
 ssmagent 22
 writer 22
production cycle 43
 identifying job stream instances 46
 managing 43
 planman command line 66
production plan
 automating processing 82
 description 56
 generating 63
 JnextPlan 43, 63
 resetting plan 75
 retrieving info 70
 starting processing 81
 Symphony file 43, 63
 unlocking plan 75
prompt
 abend 3
 ad-hoc 3
 global 3
 local 3
 named 3
 recovery 3
prompt definition 3, 149
prompt keyword 181
prxtract command 429
publications xiii

R

r11xtract command 431
ready
 job state 267
 job stream state 273
recall command 303
recovery
 job definition 138
recovery prompt 3
redo command 243, 304
referential integrity check 209
release command 400
release job command 305, 308
release sched command 307
remote engine
 workstation 121
rename command 244
rep1 command 410
rep11 command 414
rep2 command 410
rep3 command 410
rep4a command 410
rep4b command 410
rep7 command 411
rep8 command 413
replace command 247
report commands 409
 Actual Production Detail
 sample output 423
 Actual Production Details 415

report commands (*continued*)
 Actual Production Summary 415
 Calendar Listing 410
 sample output 420
 changing date format 410
 Cross Reference 416
 sample output 425
 extract programs 427
 caxtract 429
 jbxtract 427
 paxtract 430
 prxtract 429
 r11xtr 431
 reextract 431
 rxrtract 432
 Job Details Listing 410
 sample output 417
 Job Histogram 413
 sample output 422
 Job History Listing
 sample output 421
 Parameter Listing
 sample output 420
 Parameters Listing 410
 Planned Production Detail
 sample output 422
 Planned Production Details 415
 Planned Production Schedule 414
 sample output 424
 Planned Production Summary 415
 Prompt Listing 410
 sample output 419
 Resource Listing 410
 sample output 421
 sample outputs 417
 setup 409
reports commands
 Job History Listing 411
 list of commands 410
reptr command 415
rerun command 309
reserved words
 for job streams 120
 for user definitions 121
 for workstations 120
resetting plan
 planman command line 75
resolution
 variable 98
resource
 logical 9
 physical 9
 scheduling 9
resource command 312
resource definition 151
reextract command 431
rmstdlist command 401
rrcondsucc
 job definition 137
rule 9
rule-based run cycle 2
run cycle
 calendar 174
 daily 2
 date 173
 day 173
 exclusive 2, 3

- run cycle (*continued*)
 - freedays 2
 - icalendar 174
 - inclusive 2, 3
 - offset-based 2
 - on 173
 - rule-based 2
 - simple 2
 - weekly 2
 - yearly 2
- running system commands
 - from composer 248
 - from conman 257, 370

S

- same day
 - follows 165
 - follows sameday 48
 - matching criteria 48
- sched
 - job state 267
- schedtime keyword 182
- schedule keyword 183
- scheduling language 152
- scheduling resource 9
- scriptname
 - job definition 135
- secureaddr
 - workstation definition 126
- security
 - variable tables 97
- securitylevel 128
 - workstation definition 128
- sendevent command 402
- ServerID
 - mailman process 22
 - workstation definition 129
- setsym command 313
- setting
 - connection parameters 41
- SHELL_TYPE variable 38
- showcpus command 313
- showdomain command 319
- showexec command 403
- showfiles command 320
- showjobs command 322
- showprompts command 335
- showresources command 337
- showschedules command 340
- shutdown
 - utility command 404
- shutdown command 345
- simple run cycle 2
- SSL communication
 - enabling 128
- stageman
 - carryforward 56
 - SwitchPlan 76
- standard agent
 - workstation definition 127
- start command 345
- start of day
 - establishing communication 29
- startappserver command 348
- starteventprocessor command 348

- starting
 - WebSphere Application Server 26
 - workstation processes 26
- starting processing
 - production plan 81
- startmon command 349
- startOfDay
 - variable 60, 446
- StartUp command 405
- status
 - late 160
- status command 349
- stop command 350
- stop; progressive command 352
- stopappserver command 353
- stopeventprocessor command 354
- stopmon command 355
- stopping
 - WebSphere Application Server 26
 - workstation processes 26
- streamlogon
 - job definition 136
 - windows user definition 141
- stuck
 - job stream state 273
- submit docommand command 355
- submit file command 359
- submit job command 362
- submit sched command 365
- succ
 - job state 267
 - job stream state 273
- successor
 - matching criteria 49
 - pending predecessor 49
 - predecessor 45, 49
- succp
 - job state 267
- switcheventprocessor command 368
- switching extended agents
 - \$MASTER keyword 126
- switchmgr command 369
- SwitchPlan
 - JnextPlan 65
- Symphony file
 - JnextPlan 63
 - production plan 56, 63
- SystemDrive variable 34
- SystemRoot variable 34

T

- table of variables
 - using 95
- tasktype
 - job definition 136
- tcpaddr
 - workstation definition 125
- technical training xiii
- tellop command 370
- TEMP variable 34
- templates
 - for scheduling object definitions 121
- time zone
 - enabling 445
- timezone
 - in job streams 184

- timezone (*continued*)
 - workstation definition 126
- timezone keyword 184
- Tivoli Dynamic Workload Console
 - accessibility xiii
- Tivoli technical training xiii
- Tivoli Workload Scheduler
 - architecture 21
 - basic concepts 1
 - controlling job processing 12
 - defining activities 11
 - managing production 14
 - network 10
 - object 1
 - overview 1
 - processes 21
 - quick start 16
 - running event management 14
 - runtime environment 10
 - user interfaces 15
- TIVOLL_JOB_DATE variable 34, 35
- TMPDIR variable 34
- TMPTEMP variable 34
- training
 - technical xiii
- trial extension
 - planman command line 72
- trial plan
 - creating 71
 - description 58
 - extension 72
- trialsked
 - forecast plan 59
 - trial plan 58
- trigger action 9
- TWS disk space
 - monitoring 524
- TWS processes
 - monitoring 524
- TWS_PROMOTED_JOB 480
- TWS_PROMOTED_JOB variable 34, 35
- TWS_TISDIR variable 35
- type
 - workstation definition 127
- typeface conventions xiv
- TZ variable 34, 35

U

- UNISON_CPU variable 34, 36
- UNISON_DATE variable 35
- UNISON_DATE_FORMAT variable 36
- UNISON_DIR variable 34, 36
- UNISON_EXEC_PATH variable 34, 36
- UNISON_EXIT variable 37
- UNISON_HOST variable 34, 36
- UNISON_JCL variable 37
- UNISON_JOB variable 34, 36
- UNISON_JOBNUM variable 34, 36
- UNISON_MASTER variable 34, 36
- UNISON_RUN variable 35, 36
- UNISON_SCHED variable 35, 36
- UNISON_SCHED_DATE variable 36
- UNISON_SCHED_EPOCH variable 35, 36
- UNISON_SCHED_IA variable 35, 36
- UNISON_SCHED_ID variable 35, 36

- UNISON_SHELL variable 35, 36
- UNISON_STDLIST variable 35, 36, 37
- UNISON_SYM variable 35, 36
- UNISONHOME variable 34, 36
- UNIXTASK 136
- UNKNOWN 136
- unlink command 371
- unlock command 248
- unlocking plan
 - planman command line 75
- until keyword 185
- UpdateStats
 - JnextPlan 66
- USE_EXEC variable 38
- used disk space
 - monitoring 524
- user definition 141
 - trusted domain 143
- user interfaces
 - composer 15
 - conman 15
 - Dynamic Workload Console 15
 - Java API 16
 - Job Scheduling Console 16
 - optman 16
 - planman 16
 - Web Services Interface 16
- USERDOMAIN variable 35
- USERNAME variable 35
- USERPROFILE variable 35
- using
 - default variable table 96
 - variable table 95
- utility commands 375
 - at 376
 - at.allow file 378
 - at.deny file 378
 - ATSCRIPT variable 376
 - batch 376
 - changing date format 380
 - creating calendars 394
 - defining custom events 385
 - deleting files 384
 - displaying content of standard list files 396
 - displaying product version 406
 - displaying running jobs 403
 - displaying standard list files 401
 - getting HTML reports 396, 406
 - getting job information 389
 - getting TWS_home path 394
 - getting workstation information 379
 - list of commands 375
 - listing standard list files 391
 - managing parameters locally 398
 - releasing resource units 400
 - removing standard list files 401
 - sending custom events 402
 - setting mailbox file size 388
 - shutdown 404
 - starting up netman 405

V

- validate command 251
- validfrom keyword 187
- variable 9

- variable (*continued*)
 - definition 144
 - in job definitions 140
 - resolution 98
- variable table 9
 - data integrity 97
 - default 96
 - definition 147
 - lock mechanism 97
 - using 95
- variable table definition
- vartable 125
- variable tables
 - security 97
 - security file migration 96
- variables
 - ATSCRIPT 376
 - carryforward 77
 - carryStates 56, 60
 - COMPUTERNAME 34
 - enCarryForward 56, 60
 - enCFInterNetworkDeps 60
 - enCFResourceQuantity 61
 - enLegacyId 61
 - enLegacyStartOfDayEvaluation 63, 446
 - enPreventStart 61
 - enTimeZone 63, 445
 - exported locally by .jobmanrc 37, 39, 41
 - exported on UNIX 35
 - HOME 34, 35, 41
 - HOMEDRIVE 34
 - HOMEPATH 34
 - LANG 34, 35
 - LD_LIBRARY_PATH 35
 - LD_RUN_PATH 35
 - local variables 35, 37, 39, 41, 199, 253
 - LOCAL_RC_OK 37
 - logmanMinMaxPolicy 62
 - logmanSmoothPolicy 62
 - LOGNAME 34, 35
 - MAESTRO_OUTPUT_STYLE 34, 35
 - MAIL_ON_ABEND 38
 - maxLen 60
 - minLen 60
 - PATH 35
 - POSIXHOME 41
 - SHELL_TYPE 38
 - startOfDay 60, 446
 - SystemDrive 34
 - SystemRoot 34
 - TEMP 34
 - TIVOLI_JOB_DATE 34, 35
 - TMPDIR 34
 - TMPTEMP 34
 - TWS_PROMOTED_JOB 34, 35
 - TWS_TISDIR 35
 - TZ 34, 35
 - UNISON_CPU 34, 36
 - UNISON_DATE 35
 - UNISON_DATE_FORMAT 36
 - UNISON_DIR 34, 36
 - UNISON_EXEC_PATH 34, 36
 - UNISON_EXIT 37
 - UNISON_HOST 34, 36
 - UNISON_JCL 37

- variables (*continued*)
 - UNISON_JOB 34, 36
 - UNISON_JOBNUM 34, 36
 - UNISON_MASTER 34, 36
 - UNISON_RUN 35, 36
 - UNISON_SCHED 35, 36
 - UNISON_SCHED_DATE 36
 - UNISON_SCHED_EPOCH 35, 36
 - UNISON_SCHED_IA 35, 36
 - UNISON_SCHED_ID 35, 36
 - UNISON_SHELL 35, 36
 - UNISON_STDLIST 35, 36, 37
 - UNISON_SYM 35, 36
 - UNISONHOME 34, 36
 - USE_EXEC 38
 - USERDOMAIN 35
 - USERNAME 35
 - USERPROFILE 35
- variables, environment, notation xv
- vartable
 - variable table definition 125
- vartable keyword 188
- version command 252, 374, 406

W

- wait
 - job state 267
- WebSphere Application Server
 - infrastructure 21
 - starting 26
 - stopping 26
- weekly run cycle 2
- wild cards
 - composer 204
- Windows OS
 - special characters, handling 41, 67, 80, 202, 256
- windows user definition 141
- WINDOWSTASK 136
- within a relative interval
 - follows 165
 - follows relative to 47
 - matching criteria 47
- within an absolute interval
 - follows 165
 - follows absolute to 47
 - matching criteria 47
- workload customizing
 - using variable table 95
- workstation
 - backup master domain manager 4
 - class 5
 - creating 121
 - defining 121
 - domain manager 4
 - fault-tolerant agent 4
 - mailbox files
 - NetReq.msg 28
 - master domain manager 4
 - processes 21
 - remote engine type 121
- workstation class 5
- workstation class definition 131
 - cpuclass 132
 - ignore 132
 - members 132

- workstation definition 121, 128
 - access 127
 - autolink 127
 - behinfirewall 128
 - broker 127
 - cpuname 125
 - domain 126
 - extended agent 127
 - fta 127
 - fullstatus 128
 - host 126
 - manager 127
 - os type 125
 - secureaddr 126
 - ServerID 129
 - standard agent 127
 - tcpaddr 125
 - timezone 126
 - type 127
- workstation link status 316
- workstation process status 316
- workstation processes 22
 - batchman 22
 - inter-process communication 28
 - jobman 23
 - mailman 22
 - ServerID 22
 - managing change of job states 30
 - monman 22
 - netman 22
 - processes tree on UNIX 23
 - processes tree on Windows 24
 - start of day
 - establishing communication 29
 - starting 26
 - stopping 26
 - writer 22
- workstation properties 318
- workstation status 316, 318
- writer process 22

X

- XATASK 136
- xref command 416
- xrxtct command 432

Y

- yearly run cycle 2



Program Number: 5698-WSH

Printed in USA

SC32-1274-10



Spine information:

IBM Tivoli Workload Scheduler **Version 8.5.1 (Revised October 2010)**

User's Guide and Reference

