



Tivoli Maestro for UNIX User's Guide

Version 6.0

January 1999

Tivoli Maestro for UNIX User's Guide (January 1999)

Copyright Notice

Copyright © 1998-99 by Tivoli Systems, an IBM Company, including this documentation and all software. All rights reserved. May only be used pursuant to a Tivoli Systems Software License Agreement or Addendum for Tivoli Products to IBM Customer or License Agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without prior written permission of Tivoli Systems. Tivoli Systems grants you limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry the Tivoli Systems copyright notice. No other rights under copyright are granted without prior written permission of Tivoli Systems. The document is not intended for production and is furnished "as is" without warranty of any kind. **All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.**

Note to U.S. Government Users—Documentation related to restricted rights—Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corporation.

Trademarks

The following product names are trademarks of Tivoli Systems or IBM Corporation: AIX, IBM, OS/2, RISC System/6000, Tivoli Management Environment, and TME 10.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Pentium, MMX, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the United States and other countries.

Other company, product, and service names mentioned in this document may be trademarks or servicemarks of others.

Notice

References in this publication to Tivoli Systems or IBM products, programs, or services do not imply that they will be available in all countries in which Tivoli Systems or IBM operates. Any reference to these products, programs, or services is not intended to imply that only Tivoli Systems or IBM products, programs, or services can be used. Subject to Tivoli Systems' or IBM's valid intellectual property or other legally protectable right, any functionally equivalent product, program, or service can be used instead of the referenced product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Tivoli Systems or IBM, are the responsibility of the user.

Tivoli Systems or IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, New York 10504-1785, U.S.A.

Contents

Preface	xi
1. Maestro Basics	1-1
General Description	1-1
User Interfaces	1-6
Commands	1-6
Production Processes	1-8
Getting Started	1-9
Set Up Tasks	1-10
Network Mounting	1-14
Using the Maestro Interface	1-16
Maestro Main Window	1-16
Using On-line Help	1-19
Printing from Maestro Windows	1-20
2. Options and Security	2-1
Global Options	2-1
Local Options	2-8
Maestro Security	2-16
User Definitions	2-17
User-Attributes	2-20
Object-Attributes	2-22
Access Capabilities	2-27
Sample Security File	2-32
dumpsec	2-36
makesec	2-37
3. The Production Cycle	3-1
Job Execution	3-1
Scheduled Jobs	3-1
Defining Jobs within Schedules	3-2
Launching Jobs	3-2
Standard Configuration Script - jobmanrc	3-4
Local Configuration Script - \$HOME/ jobmanrc	3-5
Job Termination	3-7
Standard List Files	3-7
Submitted Jobs	3-9
The Production Cycle	3-10
Definitions	3-11

Pre-Production	3-11
scheduling	3-12
compiler	3-14
reptr	3-16
stageman	3-18
Post-Production.....	3-23
logman Command	3-23
Production.....	3-26
Starting and Stopping In a Network.....	3-26
Automating the Production Cycle.....	3-28

4. Composing Scheduling Objects	4-1
Composer Interface Basics.....	4-1
Running Composer.....	4-2
Using the Composer Main Window	4-2
Common Composer Elements	4-2
Printing and Displaying Data	4-5
Maestro Cpus.....	4-6
CPU Definition	4-9
Example Cpu Definitions for a Maestro Network	4-14
Cpu Classes.....	4-18
CPU Class Definition.....	4-21
Domains	4-24
List of Domains	4-24
DomainDefinition Window.....	4-26
Jobs	4-28
Creating a New Job.....	4-29
Modifying a Job.....	4-30
Displaying a Job	4-30
Deleting a Job.....	4-30
Filtering the List of Jobs	4-31
Defining Jobs	4-32
MPE Job Definition	4-38
Calendars	4-39
Creating a New Calendar	4-39
Modifying a Calendar	4-40
Displaying a Calendar.....	4-40
Deleting a Calendar	4-41
Filtering the List of Calendars.....	4-41
Calendar Definition	4-42
Parameters	4-45
Defining Parameters.....	4-45
Using Parameters in a Job Script: parms Command	4-47
Using Parameters in Schedules, Jobs and Prompts	4-47
Global Prompts.....	4-48
Defining Global Prompts.....	4-48
Resources.....	4-51
Defining Resources	4-51
User Definitions	4-54

Defining Users	4-54
5. Composing Schedules	5-1
Scheduling Basics	5-1
Creating a New Schedule	5-2
Modifying a Schedule	5-3
Displaying a Schedule	5-3
Deleting a Schedule	5-4
Filtering the List of Schedules	5-4
Debugging a Schedule	5-5
Selecting Schedule Dependencies	5-6
Defining Objects While Scheduling	5-8
Saving a Schedule	5-8
On/Except Panel: Schedule Execution Days	5-10
Options Panel: Selecting Schedule Options	5-13
Follows Sched/Job Panel: Select Processing for a Schedule to Follow	5-17
Prompts Panel: Selecting Prompts for a Schedule	5-19
Opens Files Panel: Selecting Schedule File Dependencies	5-21
Needs Resources Panel: Assigning Resources to a Schedule	5-25
Jobs Panel: Placing Jobs in a Schedule	5-28
Selecting Job Dependencies	5-30
Options Panel: Selecting Job Options	5-33
Follows Sched/Job Panel: Selecting Processing for a Job to Follow	5-36
Prompts Panel: Selecting Prompts for a Job	5-39
Opens Files Panel: Selecting Job File Dependencies	5-41
Needs Resources Panel: Assigning Resources to a Job	5-44
6. Managing Production	6-1
Console Manager Interface Basics	6-1
Running the Console Manager	6-1
Using the Console Manager Main Window	6-2
Object Filters	6-12
Common Filter Attributes	6-12
Ad Hoc Scheduling	6-15
Common Submit Dialog Attributes	6-15
Submitting Schedules	6-17
Submitting Jobs	6-18
Submitting Files	6-19
Submitting Commands	6-20
SHOW Window Basics	6-21
Opening and Closing a SHOW Window	6-21
Reading the Display Area	6-21
Refreshing a SHOW Window	6-22
Opening an Object Filter Dialog	6-22
Taking Action on Production Elements	6-22
List Title	6-23
Command Line Editing of Menu Actions	6-23
SHOWCPUS Window	6-24
Reading the Display Area	6-24
Filtering the Cpu Display	6-25

Viewing Schedules for a Cpu	6-25
Viewing Jobs for a Cpu	6-26
Viewing Resources for a Cpu	6-26
Viewing File Dependencies for a Cpu	6-26
Changing a Cpu's Job Limit	6-26
Changing a Cpu's Fence	6-27
Starting and Stopping a Cpu	6-27
Linking and Unlinking a Cpu	6-28
SHOWDOMAINS Window	6-29
Reading the Display Area	6-29
Filtering the Domain Display	6-29
Starting and Stopping Cpus	6-30
Linking and Unlinking Cpus	6-30
Switching the Domain Manager	6-30
SHOWSCHEDULES Window	6-32
Reading the Display Area	6-32
Filtering the Schedule Display	6-34
Viewing the Jobs for a Schedule	6-37
Changing a Schedule's Job Limit	6-38
Changing a Schedule's Priority	6-38
Releasing a Schedule from its Dependencies	6-39
Canceling a Schedule	6-39
Adding Dependencies to a Schedule	6-39
Deleting Dependencies from a Schedule	6-39
Submitting Ad Hoc Production	6-40
Viewing the Scheduling Language	6-40
SHOWJOBS Window	6-41
Reading the Display Area	6-41
Filtering the Job Display	6-45
Changing a Job's Priority	6-48
Releasing a Job from its Dependencies	6-49
Canceling a Job	6-49
Rerunning a Job	6-49
Killing a Job	6-49
Confirming a Job	6-50
Viewing a Job's Standard List File	6-51
Viewing Dependent Objects	6-52
Adding Dependencies to a Job	6-52
Deleting Dependencies from a Job	6-52
Viewing Job Details	6-53
Working with Internetwork Dependencies	6-55
SHOWRESOURCES Window	6-56
Reading the Display Area	6-56
Filtering the Resource Display	6-57
Changing Available Units of a Resource	6-57
Viewing Dependent Objects	6-58
SHOWPROMPTS Window	6-59
Reading the Display Area	6-59
Filtering the Prompt Display	6-60

Replying to a Prompt	6-60
Viewing Dependent Objects.....	6-61
SHOWFILES Window.....	6-62
Reading the Display Area.....	6-62
Filtering the File Display.....	6-63
Viewing Dependent Objects.....	6-63
Adding Dependencies to Schedules	6-64
Add Schedule Dependencies Window: Basic Operation.....	6-64
Options Panel: Selecting Schedule Options	6-66
Follows Sched/Job Panel: Selecting Schedule Follows Dependencies	6-68
Prompts Panel: Selecting Prompts for a Schedule	6-70
Opens Files Panel: Selecting Schedule File Dependencies	6-72
Needs Resources Panel: Assigning Resources to a Schedule	6-75
Adding Dependencies to Jobs.....	6-77
Add Job Dependencies Window: Basic Operation	6-77
Options Panel: Selecting Job Options	6-79
Follows Sched/Job Panel: Selecting Job Follows Dependencies	6-81
Prompts Panel: Selecting Prompts for a Job	6-83
Opens Files Panel: Selecting Job File Dependencies	6-85
Needs Resources Panel: Assigning Resources to a Job	6-87
7. Reports	7-1
Report Commands.....	7-1
Offline Output.....	7-2
rep1-rep4b	7-3
rep7	7-4
rep8	7-5
rep11	7-7
reptr	7-8
xref	7-10
Sample Reports.....	7-12
Report 01-Job Details Listing.....	7-12
Report 02-Prompt Messages Listing	7-13
Report 03-User Calendar Listing	7-14
Report 04A-User Parameters Listing	7-15
Report 04B-Maestro Resource Listing	7-16
Report 07-Job History Listing	7-17
Report 08-Job Histogram	7-18
Report 09B-Planned Production Detail	7-19
Report 10B-Actual Production Detail	7-20
Report 11-Planned Production Schedule	7-21
Report 12-Cross Reference Report	7-22
Report Extract Programs.....	7-23
Offline Output.....	7-23
jbxtract Program.....	7-23
prxtract Program	7-25
caxtract Program	7-26
paxtract Program	7-27
retract Program	7-28

r11xtr Program	7-29
rxtrct Program	7-31
8. Composer Command Line Reference	8-1
Running Composer.....	8-1
Terminal Output	8-2
Offline Output	8-2
Composer's Editor	8-3
Composer's Command Prompt.....	8-3
Composer Commands.....	8-4
Wildcards	8-4
Delimiters and Special Characters.....	8-4
Command Descriptions	8-6
List of Commands	8-6
add	8-7
build	8-8
Command	8-9
continue	8-10
create	8-11
delete	8-13
display, list, print	8-15
edit	8-19
exit	8-20
help	8-21
modify	8-22
new	8-24
redo	8-25
replace	8-27
validate	8-28
version	8-30
Managing Objects	8-31
Managing Master Files.....	8-31
Cpu Definitions	8-32
Cpu Class Definitions	8-36
Domain Definitions	8-37
Jobs	8-39
User Definitions	8-43
Calendars	8-45
Parameters	8-46
Prompts	8-48
Resources	8-49
The Scheduling Language	8-50
Scheduling Keywords	8-51
Scheduling Keyword Descriptions.....	8-52
at	8-53
carryforward	8-54
Comments	8-55
confirmed	8-56
end	8-57

every	8-58
except	8-59
follows	8-61
in order	8-63
Job Statement	8-65
limit	8-67
needs	8-68
on	8-70
opens	8-72
priority	8-74
prompt	8-75
schedule	8-77
until	8-78
9. Conman Command Line Reference	9-1
Running Conman.....	9-1
Terminal Output	9-3
Offline Output.....	9-3
Conman's Command Prompt.....	9-4
Command Syntax.....	9-5
Delimiters and Special Characters	9-6
List of Commands.....	9-7
Job Qualifiers	9-10
Selecting and Qualifying Schedules.....	9-18
Schedule Qualifiers.....	9-18
Command Descriptions	9-25
addep job	9-26
addep sched	9-28
altpass	9-30
altpri	9-31
cancel job	9-33
cancel sched	9-35
Command	9-37
confirm	9-38
console	9-40
continue	9-42
deldep job	9-43
deldep sched	9-45
display file	9-47
display job	9-48
display sched	9-49
exit	9-50
fence	9-51
help	9-52
kill	9-53
limit cpu	9-54
limit sched	9-55
link	9-56
listsym	9-58

recall	9-59
redo	9-61
release job	9-63
release sched	9-65
reply	9-67
rerun	9-68
resource	9-71
setsym	9-72
showcpus	9-73
showdomain	9-76
showfiles	9-77
showjobs	9-80
showprompts	9-87
showresources	9-90
showschedules	9-92
shutdown	9-95
start	9-96
status	9-98
stop	9-99
submit docommand	9-101
submit file	9-104
submit job	9-107
submit sched	9-111
switchmgr	9-114
tellop	9-115
unlink	9-116
version	9-118
10. Utility Commands	10-1
Command Descriptions	10-1
at and batch	10-2
cpuinfo	10-6
datecalc	10-8
dbexpand	10-13
delete	10-14
evtsize	10-15
jobinfo	10-16
jobstdl	10-18
maestro	10-20
makecal	10-21
morestdl	10-23
parms	10-25
release	10-27
rmstdlist	10-29
showexec	10-30
StartUp	10-32
version	10-33
A. Maestro Networks	A-1
Definitions	A-1

Maestro for MPE	A-2
Network Communications	A-3
Network Links	A-3
Network Operation	A-5
Network Processes	A-5
Extended Agents	A-7
UNIX Extended Agents	A-7
Managing Production for Extended Agents	A-8
Failure Launching Jobs on an X-Agent	A-8
Netman Configuration File	A-9
Network IP Address Validation	A-10
System Configuration (UNIX only)	A-10
Error/Warning Messages	A-11
Network Recovery	A-13
Initialization Problems	A-13
Network Link Problems	A-14
Setting Up a Standby Domain Manager	A-14
A Note About Network Security	A-15
Losing a Domain Manager	A-15
Switching a Domain Manager	A-16
Extended Loss of Master Domain Manager	A-16
B. Networking with MPE	B-1
Software Versions	B-1
Network Considerations	B-2
Why choose an MPE master?	B-2
Why choose a UNIX or Windows NT master?	B-3
Installation	B-3
Set Up and Configuration	B-3
Setting Up a Network with MPE Master	B-4
Setting Up a Network with UNIX or Windows NT Master	B-5
Operation with MPE Master	B-7
On the MPE Master	B-7
On UNIX and Windows NT Fault-Tolerant Agents	B-8
On the MPE Slaves	B-8
Operation with UNIX or Windows NT Master	B-9
On the UNIX or Windows NT Master	B-9
On the MPE Slaves	B-10
Network Operation	B-11
Processes	B-11
Operational Overview	B-12
C. Extended Agent Reference	C-1
Extended Agents	C-1
Cpu Definition	C-1
Method Interface	C-2
Method Command Line Syntax	C-2
Method Response Messages	C-5
Method Options File	C-6
Method Execution	C-7

Launch Job (LJ) Task.....	C-7
Manage Job (MJ) Task.....	C-8
Check File (CF) Task.....	C-8
Get Status (GS) Task.....	C-9
The cpuserinfo Command.....	C-10
Trouble Shooting.....	C-11
Job Standard List Error Messages.....	C-11
Method Not Executable.....	C-11
Other Error Messages.....	C-11
D. Internetwork Dependencies.....	D-1
Network Agents.....	D-1
Configuring a Network Agent.....	D-2
Method Options File.....	D-4
Specifying Internetwork Dependencies.....	D-5
Using the Command Line.....	D-5
Internetwork Dependencies and Conman.....	D-6
Ad Hoc Scheduling and Internetwork Dependencies.....	D-6
EXTERNAL Schedule and SHOWJOBS Window.....	D-6
Taking Action on Internetwork Dependencies for Jobs and Schedules.....	D-8
Conman Command Line Specification.....	D-9
Index.....	Index-1

Preface

The User Guide provides information on how to set up and manage job scheduling with Tivoli Maestro.

Who Should Read This Guide

The target audience for this guide is system managers who will install and configure Maestro, and system administrators who perform daily administration tasks.

Users of the guide should have some knowledge of

- The UNIX and Windows NT operating systems
- Windows and X Windows interface operations

Prerequisite and Related Documents

The following document contains information related to the Tivoli Maestro:

- *Tivoli Maestro Installation Guide*
Provides information about installing Tivoli Maestro. It also provides a quick-start procedure that allows you to begin using Maestro immediately. (Prerequisite)
- *Tivoli Maestro Network Management Platform Integration Guide*
Provides information about Tivoli Maestro's integration with IBM's NetView, and HP's OpenView and OperationCenter.

What This Guide Contains

The information in this guide is presented in a task-oriented manner. The sections are:

- Section 1 **Maestro Basics** —provides an introduction to Maestro and its interfaces and programs.
- Section 2 **Options and Security** —describes the Global and Local options and security.
- Section 3 **The Production Cycle** —describes how jobs are executed, Maestro's processing day, and the pre-production and post-production steps.
- Section 4 **Composing Scheduling Objects** —describes how to create and manage object definitions for cpus, cpu classes, domains, calendars, jobs, parameters, prompts, resources, and users.

Section 5	Composing Schedules —describes how to define schedules and their dependencies.
Section 6	Managing Production —describes how to control and manage Maestro's production environment.
Section 7	Reports —describes Maestro's reporting commands, and includes examples of the reports.
Section 8	Composer Command Line Reference —provides syntax and usage for Composer's command line interface.
Section 9	Conman Comand Line Reference —provides syntax and usage for Console Manager's command line interface.
Section 10	Utility Commands —describes Maestro's utility commands.
Appendix A	Maestro Networks —provides an overview of Maestro networking and the processes involved with network communications.
Appendix B	Networking with MPE —describes the configuration requirements and operation of Maestro networks containing HP MPE computers.
Appendix C	Extended Agent Programmer Reference —provides reference information about extended agents and access methods.
Appendix D	Internetwork Dependencies —describes using Maestro jobs and schedules on remote networks as dependencies.

Conventions Used in This Guide

The following conventions are used in this manual to describe the graphical and command line user interfaces.

Literal Text Commands, keywords, programs, and pathnames are shown in **courier bold** typeface. They must be spelled as shown. In the following example, **console** is a command name, and **sess** and **sys** are option keywords.

```
console [sess|sys]
```

Variables Variables are placeholders that must be replaced by actual values. They are shown in *courier italic* typeface. In the following example, *filename* must be replaced with an actual file name.

```
add filename
```


-
- Special Characters** Special characters in syntax statements must be entered as shown with the exception of the following, which are used as notation operators:
- [] brackets indicate optional items
 - { } braces indicate required items
 - | vertical bars indicate choices between items
 - ... ellipses indicate repetition
- Menu Commands** Selection of menu items more than two levels deep is shown in **bold** typeface with greater than signs (>) indicating the order items are selected. For example:
- From the Run menu, select **Programs > Utils > Zapper**.
- Hierarchical Displays** Selection of objects in a hierarchical display is shown in **bold** typeface with greater than signs (>) indicating the order objects are selected. For example:
- In the All Folders list, select **My Computer > FAT (D:) > Applications > Tally**, and then double-click **ReadMe.doc**.
- For object names that are variable, the placeholders are shown in italic typeface. For example:
- In the Configuration window, select **Local Configuration > *domain_name* > Nodes > *node_name* > Queues**.

Contacting Customer Support

If you encounter difficulties with any Tivoli products, you can enter <http://www.support.tivoli.com> to view the Tivoli Support home page. After you link to and submit the customer registration form, you will be able to access many customer support services on the Web.

Use the following phone numbers to contact customer support in the United States: the Tivoli number is 1-800-848-6548 (1-800-TIVOLI8) and the IBM number is 1-800-237-5511 (press or say 8 after you reach this number). Both of these numbers direct your call to the Tivoli Customer Support Call Center.

We are very interested in hearing from you about your experience with Tivoli products and documentation. We welcome your suggestions for

improvements. If you have comments or suggestions about this documentation, please send e-mail to pubs@tivoli.com.

1

Maestro Basics

Tivoli Maestro for UNIX is a fully-automated batch job scheduling system that markedly improves job throughput, and greatly reduces operations costs. This section introduces you to Maestro and its interfaces and programs. Initial Maestro set up and GUI behavior are also discussed.

General Description

Maestro's scheduling features help you plan every phase of production. During the processing day, Maestro's production control programs manage the production environment and automate most operator activities. Maestro prepares your jobs for execution, resolves interdependencies, and launches and tracks each job. Because your jobs begin as soon as their dependencies are satisfied, idle time is minimized, and throughput improves significantly. Jobs never run out of sequence, and, if a job fails, Maestro handles the recovery process with little or no operator intervention.

Schedules and Calendars

Central to Maestro's ability to manage batch job execution are the user schedules written in an intuitive scheduling language. Each schedule is dated, and consists of a list of jobs that execute as a unit (such as the weekly backup application), along with times, priorities, and other dependencies that determine the exact order of execution.

Schedules are dated using actual dates, days of the week (**mo** for every Monday, **weekdays** for every Monday through Friday, etc.), or calendars. A calendar is a set of specific dates. You can create as many calendars as required to meet your scheduling needs. For example, you can define a calendar named **paydays** containing a list of pay dates, a calendar named **monthend** containing a list of month ending dates, and a calendar named **holidays** containing a list of your company's holidays. At the start of each processing day, Maestro automatically selects all the schedules that run on that day, and carries forward uncompleted schedules from the previous day.

Times and Priorities

Time constraints can be specified for both jobs and schedules. You can specify the time that execution will begin, or the time after which execution will not be attempted. By specifying both, you can define a window within which a job or schedule will execute. For jobs, you can also specify a repetition rate; for example, you can have Maestro launch the same job every 30 minutes between the hours of 8:30 a.m. and 1:30 p.m.

Maestro has its own queuing system, consisting of 102 priority levels. Assigning priorities to your jobs and schedules gives you added control over their precedence and order of execution.

Job Fence and Job Limits

Maestro's job fence provides a type of master control over job execution. It can be set to a priority level that a job's priority must exceed before it will be allowed to execute. Setting the fence to 40, for example, will prevent jobs with priorities of 40 or less from being launched.

Job limits provide a means of allocating a specific number of job slots into which Maestro is allowed to launch jobs. A job limit can be set for each schedule, and for each cpu on which Maestro is running. Setting the cpu job limit to 25, for example, permits Maestro to have no more than 25 jobs executing concurrently.

Dependencies

Dependencies are prerequisites that must be satisfied before execution can proceed. They can be specified for both schedules and jobs to ensure the correct order of execution. The four types of dependencies are:

- You can specify that a job or schedule must not begin execution until other jobs and schedules have completed successfully.
- You can specify that a job or schedule needs one or more system resources before it can begin execution.
- You can specify that a job or schedule needs to have access to one or more files before it can begin execution.
- You can specify that a job or schedule needs to wait for an affirmative response to a prompt before it can begin execution.

In a network, dependencies can cross cpu boundaries. For example, you can make Job1, which runs on host Site1, dependent on the successful completion of Job2, which runs on host Site2.

Job Confirmation

There are instances when the completion status of a job cannot be determined until you have performed some task. You may want to check the results printed in a report, for example. In this case, you can flag the job as requiring confirmation, and Maestro will wait for your response before it marks the job as successful or abended.

Custom Resources

Maestro resources can be used to represent physical or logical resources on your system. Each consists of 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 acquires two units of the "tapes" resource would then prevent other jobs requiring more than the one remaining unit from being launched.

Global Parameters

Parameters can be used as substitutes for defining Maestro jobs and dependencies. Parameters are local to the cpu they are defined on, but can be used repeatedly. Using parameters for user logon and script file names in job definitions and file and prompt dependencies permits the use of variables that only have to be maintained in the parameters database.

Maestro parameters can also be used as scripting aids, permitting you to insert runtime values into your job scripts prior to being executed. These parameters have the advantage of being global in nature, unlike shell variables which must be exported.

Automatic Job Documentation

To launch a job, Maestro must know its name, the name of its script file (also referred to as a *job file* or *jcl file*), the user name under which it will run, and any recovery options you have chosen. This is all handled at the time you write your schedules. That is, you supply the required job information, and Maestro automatically documents the job at the same time it adds the schedule to its master schedule file. Once a job is documented, it can be used in other schedules simply by specifying its name. Jobs can also be defined independent of schedules.

Job Recovery

When you schedule a job, you can specify the type of recovery you wish to have Maestro perform if the job abends (returns an exit code other than zero). The recovery options are:

- Continue with the next job.
- Stop and do not permit the next job to execute.
- Rerun the abended job.

In addition, you can specify other actions to be taken in the form of recovery jobs and recovery prompts. For example, if a job abends, you can have Maestro automatically run a recovery job, issue a recovery prompt that requires an affirmative response, and then rerun the abended job.

Maestro Options

Two sets of options are available, Global and Local, to allow you to dictate how Maestro executes on your system. In a Maestro network, where Global options apply to all cpus, and Local options apply independently to each cpu.

Security

Every Maestro program and command checks the user's capabilities against the definitions contained in a Security file. The security structure is comprehensive, permitting the system administrator to control access to every Maestro object (schedule, job, resource, etc.), and to specify exactly what types of access will be permitted (add, modify, use, etc.).

Maestro network communication includes IP address validation to prevent access by foreign hosts.

Replicated Systems

Maestro makes duplicating a schedule across multiple systems easy. Any number of cpus can be placed in a class. Maestro schedules can then be defined for a class, replicating production on each cpu in the class.

Console Management

At the start of each processing day, Maestro selects the schedules that run on that day, and carries forward uncompleted schedules from the previous day. All of the required information is placed into a production control file, which is continually updated during the day to indicate work completed, work in

progress, and work to be done. Maestro's Conman (Console Manager) program is used to manage the information in the production control file. Among other things, Conman can be used to:

- Start and stop Maestro's control processes.
- Display the status of schedules and jobs.
- Alter priorities and dependencies.
- Alter the job fence and job limits.
- Rerun jobs.
- Cancel jobs and schedules.
- Submit new jobs and schedules.
- Recall and reply to prompts.
- Link and unlink cpus in a network.

Ad Hoc Job Submission

Jobs that are unknown to Maestro (that is, undocumented jobs) can be submitted for execution using Maestro's **at** and **batch** commands. These commands are similar to their UNIX namesakes, but the jobs are placed under Maestro's control, and are managed like scheduled jobs.

Logging and Reporting

For each documented job, Maestro logs all runtime statistics. These statistics can be printed with Maestro's reporting commands. In addition to statistical reports, there are reports that list Maestro's planned production at the start of a each day, and actual production at the end of each day. You can also print reports showing the documented details of all Maestro objects (jobs, calendars, prompts, etc.), and a cross-reference report showing the relationships between Maestro objects.

Networking

With Maestro you can manage batch jobs across a network of linked computers (cpus). By distributing the workload throughout a network, you can take full advantage of widely separated resources, and still maintain complete visibility of production activity. Computers communicate using TCP/IP links. All production set up tasks are performed on the master domain manager, and a copy of the production control file is then distributed to the other cpus. Maestro's control processes on each cpu work independently to launch and track their own jobs, communicating with their domain managers to resolve inter-cpu dependencies. Because the master domain manager's production control file is always kept up-to-date, you can

manage network production from a central console on the master domain manager.

A Maestro network is composed of one or more domains, each consisting of a domain manager and agent cpus. In the domain hierarchy, the master domain manager is the domain manager of the topmost domain. There are three types of agents: standard, fault-tolerant, and extended. Standard and fault-tolerant agents can be UNIX, Windows NT, or MPE (a Hewlett-Packard proprietary operating system) computers. An extended agent is an actual computer or an application that is hosted by a Maestro computer. Extended agents are available for MVS, SAP R/3, and other systems and applications. For more information about Maestro networking, refer to appendixes A, B, and C.

User Interfaces

A combination of graphical and command line interface programs are provided to run Maestro. This guide emphasizes the Maestro GUI. The command line interface (CLI), is available for certain advanced features not available in the GUI.

The interface programs are:

- | | |
|------------------|---|
| gcomposer | The Composer program is used to define scheduling objects and compose schedules. The command line version is composer . |
| gconman | The Console Management program is used to monitor and control Maestro's production environment. The command line version is conman . |

The GUI and CLI are independent, and can be run simultaneously, in the same session, to manipulate Maestro data.

Commands

Security

User privileges are defined and maintained with the following CLI commands.

- | | |
|----------------|---|
| dumpsec | Create an editable copy of Maestro's Security file. |
| makesec | Compile and install Maestro's Security file. |

Pre and Post Production Processing

The following commands are used to set up Maestro's processing day. To automate the process, the commands are normally placed in a schedule that runs at the start of each day.

compiler	Compile the Production Control file.
logman	Log job statistics.
schedulr	Select schedules for execution.
stageman	Carry forward uncompleted schedules, and install the Production Control file.

Reporting

Comprehensive reports are printed from Maestro's CLI. Report commands are often included as part of the daily turnover process.

rep1 - rep8	Print definitions of Maestro scheduling objects.
reptr	Print pre and post production reports.
xref	Print the Cross Reference report.

Utilities

A set of utility commands provides a wide range of capabilities.

at	Submit ad hoc jobs to be executed at a specific time.
batch	Submit ad hoc jobs to be executed as soon as possible.
datecalc	Calculate date expressions.
delete	Remove job standard list files.
disaster	Convert the software to a seven-day trial version.
jobinfo	Return information about a job.
jobstdl	Return pathnames of job standard list files.
maestro	Return the home directory of the maestro user.
makecal	Create custom calendars.
morestdl	Display job standard list files.
psetcode	Install the Tivoli software validation code.
rebuild	Clean and rebuild Maestro's master files.
release	Release scheduling resources.
rmstdlist	Remove job standard list files by age.
showexec	Display information about executing jobs.
StartUp	Start Maestro's network process Netman .

Production Processes

The following are Maestro's production processes.

- Batchman** The Production Control process. Working from the Production Control file (**Symphony**), it executes schedules, resolves dependencies, and directs **Jobman** to launch jobs.
- Jobman** The Job Management process. Launches and tracks jobs under the direction of **Batchman**.

The following are Maestro's network processes.

- Netman** The Network Management process. Establishes network connections between remote **Mailman** processes and local **Writer** processes.
- Mailman** The Mail Management process. Sends and receives inter-cpu messages.
- Writer** The Network **Writer** process. Passes incoming messages to the local **Mailman**.

Getting Started

There are two basic Maestro configurations: stand-alone and network. In a network, there is a master domain manager, on which most configuration and management functions are performed, and a number of agents. A stand-alone cpu, for all practical purposes, is the same as a master domain manager without agents and subordinate domains.

The following is a brief overview of the steps necessary to begin using Maestro:

1. Read the Release Notes for late-breaking information about the product.
2. Install the software from the distribution CD or tape. Installation instructions are in the *Maestro Installation Guide*.

Note: The *Maestro Installation Guide* also contains a *Quick Start Set Up* section that will help you set up Maestro for immediate use.

3. Add cpu definitions. In a network, these also define the links between cpus, that is, the paths used to communicate information throughout the network. Refer to section 4.
4. Define your scheduling objects. These include: calendars, parameters, resources, prompts, schedules, and jobs. Refer to sections 4 and 5.
5. Set up the Global and Local Options which define the characteristics of Maestro on your system. Refer to section 2.
6. Tailor Maestro's security to suit your needs. Refer to section 2.
7. Set up pre- and post-production processing procedures. Pre-production involves selecting the schedules that will run during the upcoming day and printing a set of reports showing the planned activity. Post-production involves printing a set of reports and logging statistics for the previous day. Once these are set up properly, they are handled automatically by a schedule that runs at the end of each day. Refer to section 3.

Set Up Tasks

The following paragraphs explain additional tasks to fully implement and begin using Maestro.

Working Directories

Before performing documentation tasks as the **maestro** user, you should create additional working directories under *maestrohome*; for example: *maestrohome/schedules* for your schedules, and *maestrohome/scripts* for your job scripts. Organizing files in this manner will make maintenance easier.

Changing Maestro's Date Format

Maestro's date format is stored in the message file:

```
maestrohome/catalog/unison.msg
```

The date format affects all commands that accept a date as an input option, and the headers in all reports. To select a different format, edit the file, locate set 50, message number 35, and enter the value for the desired format. The default value is 1, which selects a date format of *mm/dd/yy*. The values are:

- 0 To select: *yy/mm/dd*
- 1 To select: *mm/dd/yy*
- 2 To select: *dd/mm/yy*
- 3 To use the Native Language Support variables

After changing the value, generate a new catalog file as follows:

```
cd maestrohome/catalog  
gencat unison.cat unison.msg
```

Define Your Scheduling Objects

Scheduling objects are defined in Maestro using the Composer program. The number of objects and the types required are site-dependent. Maestro scheduling objects include the following:

- Cpu Definitions Maestro cpu definitions assign unique names to the processing objects, usually individual computers, on which Maestro will schedule jobs. See *Maestro Cpus* on page 4-6 for more information.

Cpu Classes	A cpu class is a group of Maestro cpus. Any number of cpus can be placed in a class. Maestro schedules can then be defined for a class, replicating production on each cpu class member. See <i>Cpu Classes</i> on page 4-18 for more information.
Domains	A domain is a named group of Maestro cpus, consisting of one or more agents and a domain manager that acts as the management hub. All domains, except for the master domain, have a parent domain. See <i>Domains</i> on page 4-24 for more information.
Calendars	A calendar is a named list of scheduling dates. For example, you can define calendars that contain month-end dates, paydays, holidays, or any other set of pertinent dates. Assigning a calendar name to a schedule causes it to be selected for execution on each of the listed dates. See <i>Calendars</i> on page 4-39 for more information.
Resources	A Maestro resource is a unique name associated with a number of available units. They can represent either physical or logical system resources. For example, if you have three tape units you can define a resource called "Tapes" with three available units. The resource can then be used as a dependency on jobs that use tape units, preventing them from being launched until the required number of units is available. See <i>Resources</i> on page 4-51 for more information.
Prompts	A Maestro prompt is a unique name associated with a textual message. They are used as dependencies to prevent jobs from launching until an affirmative response is received. See <i>Global Prompts</i> on page 4-48 for more information.
Parameters	A Maestro parameter is a unique name associated with a value. These are used as scripting aids to insert specific values into scripts wherever the parameter names appear. See <i>Parameters</i> on page 4-45 for more information.
Users	User objects consist of user names and passwords that enable Maestro jobs to be launched on Windows NT systems. See <i>User Definitions</i> on page 4-54 for more information.

- Jobs** A Maestro job consists of a unique job name that is documented in the master job file along with the path name of its script file, the user name under which the job runs, and optional recovery information. See [Jobs](#) on page 4-28 for more information.
- Schedules** A schedule is a list of jobs, along with dependencies that determine when, and in what order, they are launched. See [Scheduling Basics](#) on page 5-1 for more information.

You should begin with a limited number of scheduling objects, adding more as you become more familiar with Maestro. You might start, for example, with two or three of your most frequent applications, defining scheduling objects to meet their requirements only. Examples of converting **crontab** entries to Maestro schedules can be found at the end of the *Maestro for UNIX Demonstration Guide*.

Note: Cpus, calendars, parameters, prompts, resources, and jobs must be defined first before they can be referenced in your schedules. Jobs can be defined beforehand or while you are composing your schedules.

Set Up Maestro Security

Each time you run a Maestro program, or execute a Maestro command, security information is read from the Security file to determine your user capabilities. The file contains one or more user definitions, each of which defines a user's capabilities on the basis of object types and the types of access permitted or denied.

The **maestro** user is already defined in the Security file, and can be used to complete the set up procedure. You can modify the Security file at a later time to conform to your system requirements. For a complete discussion of security, refer to [Maestro Security](#) starting on page 2-16.

Set Up Global and Local Options

Some of the runtime characteristics of Maestro are determined by option settings. The Global options affect all cpus in a Maestro network, and the Local options affect the cpus individually. Refer to [Global Options](#) on page 2-1 and [Local Options](#) on page 2-8 for more information.

Set Up the Job Execution Environment

Before launching each job, Maestro executes a Standard Configuration Script called **jobmanrc**. This script can be used to establish the desired environment for all Maestro jobs. You can also implement Local Configuration Scripts that are executed automatically following the Standard Configuration Script. The Local scripts establish the desired environment for jobs that log in under different user names.

For a complete discussion of job execution, refer to *[Job Execution](#)* starting on page 3-1.

Backup Master Directories

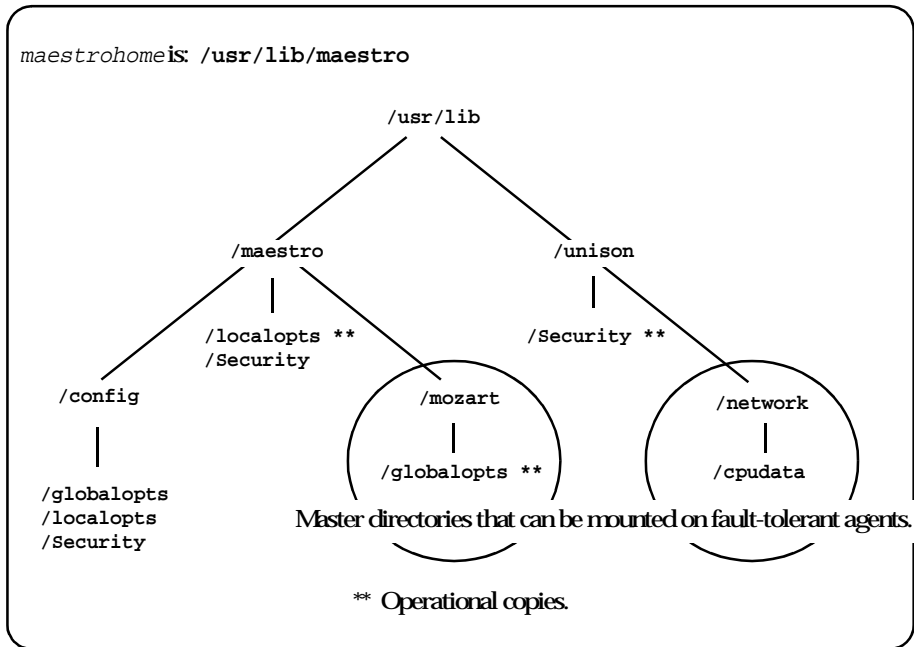
It is highly recommended that the directories *maestrohome/mozart* and *maestrohome/unison* be copied from the master domain manager to its backup on a regular basis, or at least whenever scheduling objects or Maestro security are modified.

Working Directories

During the installation process, a set of template files is installed in the *maestrohome/config* directory. Maestro's **customize** script also installs the following:

- An operational Local Options file is installed as *maestrohome/localopts*.
- An operational Global Options file is installed as *maestrohome/mozart/globalopts*
- An operational Security file is installed as *maestrohome/./unison/Security*.
- A Security template file is copied to *maestrohome/Security*. This copy can be edited to make the necessary modifications for your system. It must then be installed with the **makesec** command.

The Global Options, Local Options, and Security files are described in section 2.



Network Mounting

In a network, the master domain manager's directories can be NFS mounted on other domain managers and fault-tolerant agents as shown above. Mounting the master domain manager's directories permits a user logged in on another computer to perform the same administrative functions that are permitted on the master domain manager.

Note: NFS mounting between big endian and little endian computers will not work.

If the master domain manager's directories are not mounted, the following limitations are placed on users logged into other computers:

- You cannot use Composer except to define Maestro parameters, which are local to each cpu.

- You cannot use Conman for the following operations:
 - Submit jobs and schedules with the Submit dialogs or `submit` command.
 - Add prompt dependencies with the Add Dependencies dialog or `adddep` command.
 - Use the `;from` and `;file` options in a `rerun` command.
 - Display jobs and schedules with the `display` command.

Before mounting the directories, make certain that the file system containing the required directories has been included in the `/etc/exports` file on the master domain manager. If you choose to control the availability of the file system, make the appropriate entries in the `/etc/hosts` file, or the `/etc/netgroup` file on the master domain manager.

The mount points on other computers must be the same as the master domain manager. For example, on each fault-tolerant agent:

```
cd maestrohome
/etc/mount host:mozart    mozart
/etc/mount host:../unison/network  ../unison/network
```

Where `host` is the host name of the master domain manager. These commands may not be appropriate for your system—check with your system administrator. To have the directories mounted automatically, the mounts can be entered in the `/etc/checklist` file. If for any reason the master domain manager's directories become inaccessible, they must be unmounted on the fault-tolerant agents, so they will begin using their own local copies of required files.

Starting Maestro for the First Time

Once you have set up a basic configuration, you can start Maestro for the first time as follows:

1. Start the **Netman** process by logging in as **maestro** and executing the **StartUp** command on each cpu.

```
startUp
```

2. To automate the daily production cycle, add the **final** schedule to the Master Schedule file by entering the following command:

```
composer "add maestrohome/Sfinal"
```

Once added, the **final** schedule will run the **Jnextday** job at the end of each processing day—the default run time is 5:59 a.m.

3. Manually launch the **Jnextday** job on the master domain manager in a

Maestro network.

`Jnextday`

When Maestro is started for the first time, its cpu job limit is set to zero. No jobs will be launched until you raise the job limit. See [*Changing a Cpu's Job Limit*](#) on page 6-26.

For a complete discussion of the daily production cycle, refer to [*The Production Cycle*](#) starting on page 3-10.

Unlicensed Cpu Messages

If, after installing the software, you receive a message indicating the software is not licensed for a cpu, execute the disaster command as follows:

`disaster`

This will convert your software to a seven-day trial version. Before the trial version expires, call your local Tivoli Support Center to obtain a new validation code and convert your software back to a production version.

Using the Maestro Interface

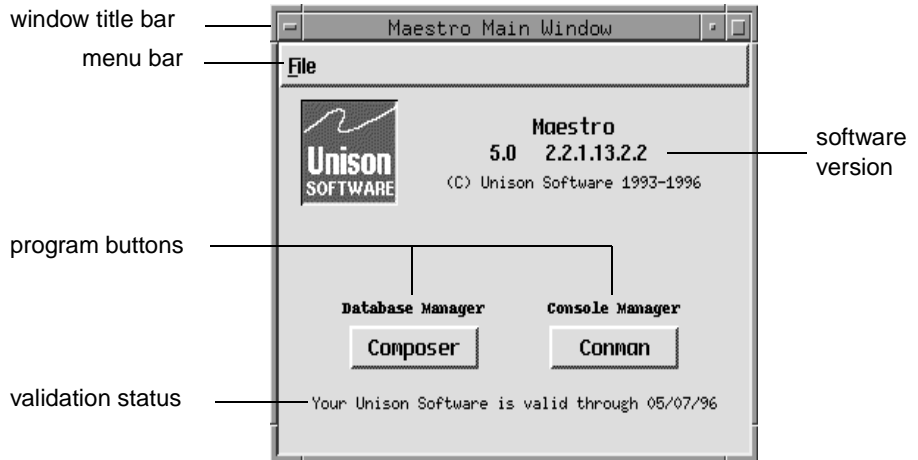
Maestro installation is an administrative task and is separated from the user-oriented focus of this manual. The *Maestro Installation Guide* contains all the necessary information to get Maestro up and running. Please refer to it to install Maestro for the first time, update an existing installation, or alter Maestro's network configuration.

To start Maestro's GUI, login as **maestro**, set your DISPLAY variable and then enter the following:

`gmaestro`

Maestro Main Window

The Maestro Main Window comes up displaying primary information about Maestro. All of the Maestro windows follow standard Motif operating conventions. You should be familiar with operating in a graphical environment before using the Maestro GUI.



- window title bar Contains the name of the window. To move the window, position the arrow pointer in the title bar, click, and drag your mouse while holding down the mouse button.
- menu bar Contains the window's menus. To open a menu, place the arrow over the desired menu and click. Additional options are displayed in the open menu.
- software version The version and patch level of the Maestro software you are running is printed in this space.
- program buttons Click these buttons to start the Composer and Conman programs. If a program is already running from the Maestro Main window, its button is dimmed. The two programs, Composer and Conman, are detailed in the chapters that follow.
- validation status The current validation status of the Maestro software you are running is printed in this space.

If your software is about to expire or has expired, consult the *Maestro Installation Guide* for information on how to re-validate the software.

Exiting Maestro

Select Quit from the File menu to exit the Maestro Main Window. This action, however, does not exit Conman or Composer, if they are running.

Navigation Notes

The number three mouse button—right button on a right-hand-configured mouse—can be used to display either pop-up Actions or Objects menus when the mouse pointer is inside the scrolled list of a window.

Menus can also be selected using the keyboard. Hold the *Alt* key (on most keyboards) while typing the underlined letter in the menu title to open the menu. To select an item in an open menu, type the underlined letter of the item and press the space bar.

Wildcards

Some text entry fields permit wildcards. The wildcards for Maestro are:

- ? Replaces one alpha character.
- % Replaces one numeric character.
- @ Replaces zero or more alphanumeric characters.

Wildcards are generally used to broaden or narrow a selection. For example, if you want to specify all cpus, you can simply enter the @ wildcard (where permitted), or, to get a listing of those cpus whose names begin with `site` and end with a number, enter `site%`.

Case Sensitivity

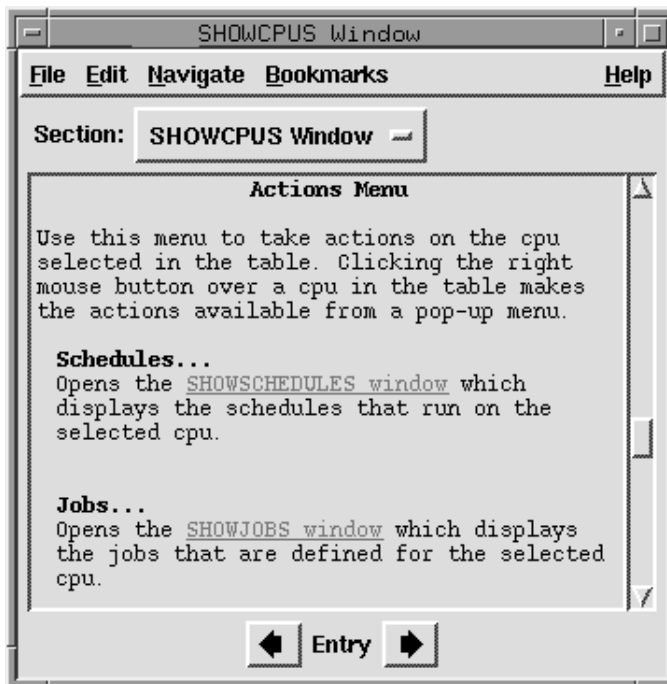
All names of scheduling objects, cpus, and schedules can be entered in either uppercase or lowercase. They are displayed in uppercase in Maestro lists. Text in description and comment fields, text for prompts, and parameter values preserve their case when displayed and are case sensitive. UNIX path names are always case sensitive.

Using On-line Help

Information is provided for every Maestro window in the form of a Help Browser Window. There is a Help menu on all Maestro windows. Dialog boxes without menus have Help buttons.



To open the Help Browser Window, select On Window... from the Help menu or click the Help button. The Help Browser Window, shown below, displays help for the current selection and provides menus and buttons to navigate through more Maestro help.



Features of the Help Browser Window include:

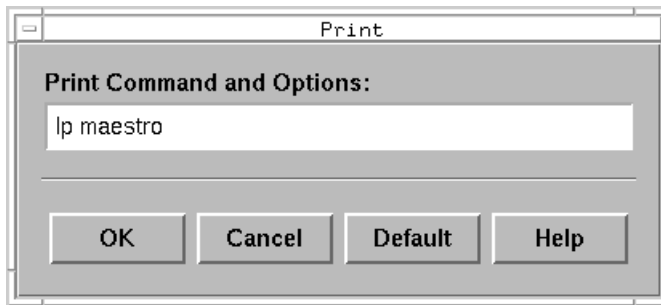
- Advanced navigation controls, including hypertext links to related topics.
- On-line table of contents for the help documents.

- Keyword search engine.
- Bookmarking capabilities.

Information and instructions for these and other help features can be found on-line from the Help Browser Window's Help menu.

Printing from Maestro Windows

When Print... is selected from any Maestro window's File menu, the Print dialog box opens.



In the entry field, enter the print arguments: for example, the print command and the printer device. Click OK and formatted text of the contents of the current window is printed. Click Default to insert the default print command (derived from the environment variable MAESTROL) in the entry field. Click Cancel to close the dialog and cancel printing.

2

Options and Security

This section describes the Global and Local Options and Maestro security. The Global and Local Options are used to define how Maestro executes on your system. Maestro's security provides a way to define the type of access permitted or denied based on the attributes of individual users or group of users.

Global Options

The Global Options are defined on the master domain manager, and apply to all cpus in the Maestro network.

Global Options File

Global Options are entered in the **globalopts** file with a text editor . Changes can be made at any time, but do not take effect until Maestro is stopped and restarted. The syntax is described below. Entries are not case-sensitive.

Global Option Syntax	Default Value
<code># comment</code>	
<code>automatically grant logon as batch = {yes no}</code>	no
<code>batchman schedule = {yes no}</code>	no
<code>carry job states = ([state[,...]])</code>	null
<code>carryforward = {yes no all}</code>	yes
<code>company = companyname</code>	null
<code>expanded version= {yes no}</code>	null
<code>history = days</code>	10
<code>ignore calendars = {yes no}</code>	no
<code>master = cpu</code>	null
<code>retain rerun job name = {yes no}</code>	no
<code>start = starttime</code>	0600

<code># comment</code>	Treat everything from the pound sign to the end-of-line as a comment.
<code>automatically grant logon as batch</code>	For Windows NT jobs only. If set to yes , the logon users for Windows NT jobs are automatically granted the right to "logon as batch." If set to no , or omitted, the right must be granted manually on a user or group basis. Note that the right cannot be granted automatically for users running jobs on a BDC, so you must grant the right manually in those cases.
<code>batchman schedule</code>	This is a production option that affects the operation of Maestro's Production Control process Batchman. Its setting determines the priority assigned to the schedules created for unscheduled jobs. Enter yes to have a priority of 10 assigned to these schedules. Enter no to have a priority of zero assigned to these schedules.
<code>carry job states</code>	<p>This is a pre-production option that affects the operation of the Stageman command. Its setting determines the jobs, by state, that are included in schedules that are carried forward. The parentheses enclosing the job states are required, but can be replaced by paired double or single quotation marks. The commas can be replaced by spaces. The valid job states are:</p> <pre>abend, abenp, add, done, exec, fail, hold, intro, pend, ready, rjob, sched, skel, succ, succp, susp, wait, waitd</pre> <p>Some examples:</p> <pre>carry job states=(abend,exec,hold,intro) carry job states="abend exec hold intro" carry job states='abend exec hold intro'</pre> <p>Or, an empty list:</p> <pre>carry job states=()</pre> <p>See Carry Forward Options on page 2-6 for more information.</p>
<code>carryforward</code>	This is a pre-production option that affects the operation of the stageman command. Its setting determines whether or not uncompleted schedules will be carried forward from the old to the new

Production Control file. Enter **yes** to have uncompleted schedules carried forward only if the Carry Forward option is enabled in the schedule definition. Enter **a11** to have all uncompleted schedules carried forward, regardless of the Carry Forward option. Enter **no** to completely disable the carry forward function.

The **stageman -carryforward** command line option is assigned the same values, and serves the same function as the **carryforward** Global Option. If it is used, it overrides the Global Option. See [stageman](#) on page 3-18 for more information.

See [Carry Forward Options](#) on page 2-6 for more information.

<code>company</code>	Your company's name (up to 40 characters). If the name contains spaces, place the entire name in quotation marks (").
<code>expanded version</code>	This option is set during installation by customize on UNIX, and Setup on Windows NT. If set to yes , expanded databases are used. If set to no , expanded databases are not used. Expanded databases permit the use of long object names-- for example, expanded job names can contain up to sixteen characters. The option is also set to yes when you run the dbexpand utility to convert from non-expanded to expanded databases. If the option does not exist, as in the case of a Maestro version earlier than 6.0, it is interpreted as no .
<code>history</code>	Enter the number of days for which job statistics will be retained. Older statistics are discarded on a first-in, first-out basis. This has no effect on job standard list files, which must be removed with the rmstdlist command described on page 10-29 .
<code>ignore calendars</code>	This is a pre-production option that affects the operation of the compiler command (see page 3-14). Its setting determines whether or not user calendars will be copied into the new Production Control file. Enter yes to prevent user calendars from being copied into the new Production Control file. This will conserve space in the file, but will not permit the use

of calendar names in date expressions. Enter **no** to have user calendars copied into the Production Control file. See *compiler* on page 3-14 for more information.

master

The Maestro name of the master domain manager.

retain rerun job name

This is a production option that affects the operation of Maestro's Production Control process Batchman. Its setting determines whether or not jobs that are rerun with the Conman Rerun command will retain their original job names. Enter **yes** to have rerun jobs always retain their original job names. Enter **no** to permit the **rerun from** name to be assigned to rerun jobs.

start

Enter the start time of Maestro's processing day in 24 hour format: *hhmm* (0000-2359). The default start time is 6:00 am, and the default launch time of the **final** schedule is 5:59 am. If you change this option, make sure you also change the launch time of the **final** schedule, which is usually set to one minute before the start time.

Global Options for MPE Slaves

In a Maestro network with a UNIX or NT system configured as domain managers and HP 3000 (MPE) systems configured as slaves, a set of Global Options can be defined on the UNIX/NT master domain manager to control Maestro's operation on the MPE slaves. These options take the place of parameters that would otherwise be set on the MPE systems using the Arranger program's CTP1 transaction. To incorporate these options, add them to your `globalopts` file. The syntax is described below. The entries are not case-sensitive. For a complete description of the MPE parameters, see *CTP1 Transaction* in the *Maestro for MPE User Guide*.

Global Option Syntax	Default Value
<code>rules mode = {yes no}</code>	no
<code>batchman schedule = {yes no}</code>	no
<code>all userjobs in userjobs schedule = {yes no}</code>	no
<code>set mpe job pri to zero = {yes no}</code>	no

`rules mode` Replaces CTP1- parameter 4 (Complete Control Mode). If set to `yes`, you must also set `batchman schedule` to `yes`.

`batchman schedule` Replaces CTP1- parameter 7 (Assign priority 10 to Batchman-created schedules). Note that this is also a valid option on UNIX and Windows NT. See [Global Options](#) on page 2-1.

`all userjobs in userjobs schedule` Replaces CTP1- parameter 8 (Place all userjobs in USERJOBS schedule). Set to `no`, if `rules mode` is set to `yes`.

`set mpe job pri to zero` Replaces CTP1- parameter 9 (Force MPE priority to 0 for all userjobs). Set to `no`, if `all userjobs in userjobs schedule` is set to `yes`.

Carry Forward Options

Schedules are carried forward by the **stageman** command during end-of-day processing. The carry forward process is affected by:

- The Carryforward schedule option (see [page 5-15](#)).
- The `carryforward` Global Option ([page 2-2](#)).
- The `stageman -carryforward` command line option (see [page 3-18](#)).
- The `carry job states` Global Option (see [page 2-2](#)).

The following table shows how the carry forward options work in combination.

Global Options	Carry Forward Operation
<code>carryforward=no</code>	No schedules are carried forward.
<code>carryforward=yes</code> <code>carry job states=(states)</code>	Schedules are carried forward only if they have jobs in the specified states, and they have the Carryforward option enabled. Only jobs in the specified states are carried forward with the schedules.
<code>carryforward=yes</code> <code>carry job states=()</code>	Schedules are carried forward only if they are uncompleted, and they have the Carryforward option enabled. All jobs are carried forward with the schedules.
<code>carryforward=all</code> <code>carry job states=(states)</code>	Schedules are carried forward if they have jobs in the specified states. Only jobs in the specified states are carried forward with the schedules.
<code>carryforward=all</code> <code>carry job states=()</code>	Schedules are carried forward if they are uncompleted. All jobs are carried forward with the schedules.

Carry Forward Notes

- Schedules in the ABEND, HOLD, READY, EXEC, and STUCK states are considered uncompleted.
- The `stageman -carryforward` command line option, if used, always overrides the `carryforward` Global Option. The default, if neither is specified, is `carryforward=yes`.
- The default entry is null for the `carry job states` Global Option. That is, if the list is empty or the option is absent, carry forward works as shown for `carry job states=()` above.

- Jobs and schedules that were cancelled, or whose Until times have expired, are never carried forward.
- The decision to carry forward a repetitive job (defined by the Every option) is based on the state of its most recent run.
- If a job is running when `jnextday` begins execution, and it is not in a Carryforward-enabled schedule, the job continues to run and is placed in the "userjobs" schedule for the new production day.

Global Options File Example

The following template file contains Tivoli's default settings:

```
maestrohome/config/globalopts
```

During the installation process, a working copy of the Global Options file is installed as:

```
maestrohome/mozart/globalopts
```

You can customize the working copy to suit your requirements. Following is a sample listing of a Global Options file.

```
# Maestro globalopts file on the Maestro master defines
attributes
# of the Maestro network.
#-----
company=                "Tivoli Systems"
master=                  main
start=                   0600
history=                  10
carryforward=            yes
ignore calendars        no
batchman schedule=      no
retain rerun job name=  no
#
#-----
# End of globalopts.
```

See [Working Directories](#) on page 1-13 for information about mounting requirements in a Maestro network.

Local Options

The Local Options are defined on each cpu, and apply only to that cpu.

Local Options File

Local Options are entered in the `localopts` file with a text editor. Changes can be made at any time, but do not take effect until Maestro is stopped and restarted. The syntax is described below. Entries are not case-sensitive.

Local Option Syntax	Default Value
<code># comment =</code>	
<code>bm check file = seconds</code>	120
<code>bm check status = seconds</code>	300
<code>bm check until = seconds</code>	300
<code>bm look = seconds</code>	30
<code>bm read = seconds</code>	15
<code>bm stats = {on off}</code>	off
<code>bm verbose = {on off}</code>	off
<code>jm job table size = size</code>	160
<code>jm look = seconds</code>	300
<code>jm nice = value</code>	0
<code>jm no root = {yes no}</code>	no
<code>jm read = seconds</code>	10
<code>merge stdlists = {yes no}</code>	yes
<code>mm read = seconds</code>	15
<code>mm response = seconds</code>	600
<code>mm retry link = seconds</code>	600
<code>mm sound off = {yes no}</code>	no
<code>mm unlink = seconds</code>	960
<code>nm ipvalidate = {none full}</code>	none
<code>nm mortal = {yes no}</code>	no
<code>nm port = tcpaddr</code>	31111
<code>nm read = seconds</code>	60
<code>nm retry = seconds</code>	800
<code>stdlist width = width</code>	80
<code>syslog local = facility</code>	-1

Local Option Syntax	Default Value
<code>thiscpu = <i>cpu</i></code>	<code>thiscpu</code>
<code>wr read = <i>seconds</i></code>	<code>600</code>
<code>wr unlink = <i>seconds</i></code>	<code>600</code>
<code>mozart directory = <i>mozart_share</i></code>	<code>none</code>
<code>unison network directory = <i>unison_share</i></code>	<code>none</code>
<code>parameters directory = <i>parms_share</i></code>	<code>none</code>

<code># <i>comment</i></code>	Treat everything from the pound sign to the end-of-line as a comment.
<code>bm check file</code>	Enter the minimum number of seconds BATCHMAN will wait before re-checking for the existence of a file that is used as a dependency.
<code>bm check status</code>	Enter the number of seconds Batchman will wait between checking the status of an internetwork dependency.
<code>bm check until</code>	Enter the maximum number of seconds Batchman will wait before reporting the expiration of an Until time for job or schedule. Decreasing the value below the default setting may unduly load the system. If it is set below the value of Local Option <code>bm read</code> , the value of <code>bm read</code> is used in its place.
<code>bm look</code>	Enter the minimum number of seconds Batchman will wait before scanning and updating its Production Control file.
<code>bm read</code>	Enter the maximum number of seconds Batchman will wait for a message in its message file.
<code>bm stats</code>	Enter <code>on</code> to have Batchman send its startup and shutdown statistics to its standard list file. Enter <code>off</code> to prevent Batchman statistics from being sent to its standard list file.
<code>bm verbose</code>	Enter <code>on</code> to have Batchman send all job status messages to its standard list file. Enter <code>off</code> to prevent the extended set of job status messages from being sent to the standard list file.
<code>jm job table size</code>	Enter the size, in number of entries, of the job table used by Jobman.

Local Options

<code>jm look</code>	Enter the minimum number of seconds Jobman will wait before looking for completed jobs, and performing general job management tasks.
<code>jm nice</code>	UNIX only. Enter the <code>nice</code> value to be applied to jobs launched by Jobman.
<code>jm no root</code>	UNIX only. Enter <code>yes</code> to prevent Jobman from launching root jobs. Enter <code>no</code> to allow Jobman to launch root jobs.
<code>jm read</code>	Enter the maximum number of seconds Jobman will wait for a message in its message file.
<code>merge stdlists</code>	Enter <code>yes</code> to have all of Maestro's control processes, except Netman, write their console messages to a single standard list file. The file is given the name <code>MAESTRO</code> . Enter <code>no</code> to have each process write to its own standard list file. See <i>Netman Local Options</i> on page 2-12.
<code>mm read</code>	Enter the rate, in seconds, at which Mailman checks its mailbox for messages. If omitted, the default is 15 seconds. Defining a lower value will cause Maestro to run faster at the expense of using more processor time.
<code>mm response</code>	Enter the maximum number of seconds Mailman will wait for a response before reporting that another cpu is not responding. The response time should not be less than 90 seconds.
<code>mm retry link</code>	Enter the maximum number of seconds Mailman will wait, after unlinking from a non-responding cpu, before it attempts to re-link to the cpu.
<code>mm sound off</code>	Determines how Mailman will respond to a <code>command tellop ?</code> command. Enter <code>yes</code> to have Mailman display information about every task it is performing. Enter <code>no</code> to have Mailman send only its own status.
<code>mm unlink</code>	Enter the maximum number of seconds Mailman will wait before unlinking from another cpu that is not responding. The wait time should not be less than the response time entered for the Local Option <code>mm response</code> .
<code>nm ipvalidate</code>	Enter <code>full</code> to enable IP address validation: if IP validation fails, the connection is not allowed. Enter <code>none</code> to allow connections when IP validation fails. For more information see <i>Network IP Address Validation</i> on page A-10. See also <i>Netman Local Options</i> on page 2-12.

<code>nm mortal</code>	Enter yes to have netman quit when all of its child processes have stopped. Enter no to have Netman keep running when its child processes have stopped. See <u><i>Netman Local Options</i></u> on page 2-12.
<code>nm port</code>	Enter the TCP port number that Netman responds to on this computer. This must match the TCP Address in the computer's Maestro cpu definition. See <u><i>Netman Local Options</i></u> on page 2-12.
<code>nm read</code>	Enter the maximum number of seconds Netman will wait for a connection request before checking its message queue for stop/start commands. See <u><i>Netman Local Options</i></u> on page 2-12.
<code>nm retry</code>	Enter the maximum number of seconds Netman will wait before retrying a connection that has failed. See <u><i>Netman Local Options</i></u> on page 2-12.
<code>stdlist width</code>	Defines the maximum width of Maestro's console messages. You can enter a column number in the range 1-255, and lines will be wrapped at that column or before, depending on the presence of imbedded carriage control characters. Enter a negative number, or zero, to ignore line width. See <u><i>Netman Local Options</i></u> on page 2-12. On UNIX, you should ignore line width if you enable system logging with <code>syslog local</code> (see <u><i>below</i></u>).
<code>syslog local</code>	For UNIX computers only. Enables or disables Maestro system logging. Enter -1 to turn off system logging for Maestro. Enter a number 0-7 to turn on system logging, and have Maestro use the corresponding local facility (LOCAL0-LOCAL7) for its messages. Enter any other number to turn on system logging, and have Maestro use the USER facility for its messages. For more information, see <u><i>Maestro Console Messages and Prompts</i></u> on page 2-12. See also <u><i>Netman Local Options</i></u> on page 2-12.
<code>thiscpu</code>	The Maestro name of this cpu.
<code>wr read</code>	Enter the number of seconds Writer will wait for an incoming message before checking for a termination request from Netman.
<code>wr unlink</code>	Enter the number of seconds Writer will wait before exiting if no incoming messages are received. The lower limit is 120, and the default is 600.

Netman Local Options

If Netman's home directory is not the same as Maestro's home directory, the following local options are moved from Maestro's `localopts` file to a separate `localopts` file in the Netman directory:

```
nm ipvalidate
nm mortal
nm port
nm read
nm retry
merge stdlists
stdlist width
syslog local
```

For more information about the Netman directory, refer to the topic *Product Groups* in section 1 of the *Tivoli Maestro Installation Guide*.

Decentralized Administration Options (Windows NT only)

If you installed Maestro using the procedure that permits decentralized administration of Maestro scheduling objects, use these options to define the shared directories on the master domain manager. For information about installation prerequisites, refer to *Decentralized Administration* and *Set Up for Decentralized Administration* in section 3 of the *Maestro Installation Guide*.

`mozart directory`

Enter the name of the master domain manager's shared `mozart` directory.

`unison network directory`

Enter the name of the master domain manager's shared `unison` directory.

`parameters directory`

Enter the name of the master domain manager's shared `maestrohome` directory.

If an option is not set or does not exist, the Maestro Composer program attempts to open the database files on the local computer.

Maestro Console Messages and Prompts

Maestro's control processes (Netman, Mailman, Batchman, Jobman, and Writer) write their status messages, referred to as *console messages*, to standard list files. Included in these messages are the prompts used as job and schedule dependencies. On UNIX, the messages can also be directed to the syslog

daemon (`syslogd`), and to a terminal running the Conman program. These features are described below.

sysloglocal Setting (UNIX only)

If you set `sysloglocal` in the Local Options file to a positive number, Maestro's control processes send their console and prompt messages to the syslog daemon. Setting it to `-1` turns this feature off. If you set it to a positive number to enable system logging, you must also set the Local Option `stdlistwidth` to zero, or a negative number.

Maestro's console messages correspond to the following syslog levels:

<code>LOG_ERR</code>	Error messages: control process abends, file system errors, etc.
<code>LOG_WARNING</code>	Warning messages: link errors, stuck schedules, etc.
<code>LOG_NOTICE</code>	Special messages: prompts, tellops, etc.
<code>LOG_INFO</code>	Informative messages: job launches, job and schedule state changes, etc.

Setting `sysloglocal` to a positive number defines the syslog facility used by Maestro. For example, setting it to 4 tells Maestro to use the local facility `LOCAL4`. After doing this, you must make the appropriate entries in the `/etc/syslog.conf` file, and reconfigure the syslog daemon. To use `LOCAL4`, and have Maestro's messages sent to the system console, enter the following line in `/etc/syslog.conf`:

```
local4    /dev/console
```

Or, to have Maestro's error messages sent to the `maestro` and `root` users, enter the following:

```
local4.err  maestro,root
```

Note that the *selector* and *action* fields must be separated by at least one tab. After modifying `/etc/syslog.conf`, you can reconfigure the syslog daemon by entering the following command:

```
kill -HUP `cat /etc/syslog.pid`
```

console Command

You can use the Console Manager's **console** command to set Maestro's message level, and to direct the messages to your terminal. The message level setting affects only Batchman and Mailman messages, which are the most numerous. It also sets the level of messages written to the standard list file(s) and the syslog daemon. The following command, for example, sets the level of Batchman and Mailman messages to 2, and causes the messages to be sent to your terminal:

```
console sess;level=2
```

Messages continue to be sent to your terminal until you either execute another Console command, or exit Conman. To stop sending messages to your terminal, you can enter the following Conman command:

```
console sys
```

For more information, see [Displaying Messages in the Console](#) on page 6-4 and [console](#) on page 9-40.

Local Options File Example

The following template file contains Tivoli's default settings:

```
maestrohome/config/localopts
```

During the installation process, a working copy of the Local Options file is installed as:

```
maestrohome/localopts
```

You can customize the working copy to suit your requirements. On the following page is a sample listing of a Local Options file. See [Working Directories](#) on page 1-13 for more information.

```
# maestro localopts file defines attributes of this cpu.
#-----
thiscpu                =sysl
merge stdlists        =yes
stdlistwidth          =80
sysloglocal           =-1
#-----
# Attributes of this cpu for Maestro batchman process:
bm check file         =120
bm check until        =300
bm look               =30
bm read               =15
bm stats              =off
bm verbose            =off
#-----
# Attributes of this cpu for Maestro jobman process:
jm job table size     =160
jm look               =300
jm nice               =0
jm no root            =no
jm read               =10
#-----
# Attributes of this cpu for Maestro mailman process:
mm response           =600
mm retrylink          =600
mm sound off          =no
mm unlink             =960
#-----
# Attributes of this cpu for Maestro netman process:
nm mortal             =no
nm port               =31111
nm read               =60
nm retry              =800
#-----
# Attributes of this cpu for Maestro writer process:
wr read               =600
wr unlink             =720
#-----
# Optional attributes of this cpu for remote database files
# mozart directory = d:\maestro\mozart
# parameters directory = d:\maestro
# unison network directory = d:\maestro\..\unison\network
#
#-----
# End of localopts.
```

Maestro Security

Maestro programs and commands determine a user's capabilities by comparing the user's name with the user definitions contained in the Security file. The following pages explain how to write your user definitions and manage the Security file.

A template file, named `maestrohome/config/Security`, is provided with the software. During installation, a copy of the template is placed in `maestrohome/Security`, and a compiled (operational) copy is installed as `../unison/Security`.

To create user definitions, edit the template file `maestrohome/Security` — do not modify the original template, `maestrohome/config/Security`. Then use the `makesec` command to compile and install a new operational Security file. After it is installed, you can create an editable copy of the operational file with the `dumpsec` command. The `makesec` and `dumpsec` commands are described later in this section. Changes to the Security file take effect when Maestro is stopped and restarted.

Network Security File Maintenance

Each UNIX and NT cpu in a Maestro network (domain managers, fault-tolerant agents and standard agents) has its own Security file. The files can be maintained independently on each cpu, or you can set up the Security file on the master domain manager and copy it to each domain manager, fault-tolerant agent, and standard agent. See [Working Directories](#) on page 1-13 for more information.

User Definitions

Following is the syntax for user definitions in the Security file.

```
[# comment]
user def-name user-attributes          [* comment]
begin
    object-type [object-attributes] access[=access[,...]]
    [object-type ... ]
[end]
[user ... ]
```

<code>[# *] comment</code>	Everything following a pound sign or an asterisk, and at least one space, to the end of line is treated as a comment. Comments are not copied into the operational Security file installed by <code>makesec</code> .
<code>user def-name</code>	The name of this definition (up to 36 alphanumeric characters starting with a letter).
<code>user-attributes</code>	One or more attributes that serve to qualify the users to whom the definition applies. See User-Attributes on page 2-20.
<code>begin</code>	A required keyword delimiting the user statement and the first object statement.
<code>object-type</code>	The type of object the user will be given permission to access. The object types are: <ul style="list-style-type: none"> <code>calendar</code> User calendars. <code>cpu</code> Cpu, domain, and cpu class. <code>file</code> Maestro master files. <code>job</code> Scheduled jobs. <code>parameter</code> User parameters. <code>prompt</code> Global (named) prompts. <code>resource</code> Scheduling resources. <code>schedule</code> Schedules. <code>userobj</code> User definition scheduling objects. <p>Omitting an object type prevents access to all objects of that type.</p>
<code>object-attributes</code>	One or more attributes that serve to qualify objects of the specified type. See Object-Attributes on page 2-22. If no attributes are specified, every occurrence of the object type qualifies.

`access[=access[, ...]]`

A list of access capabilities given to the user. If none are specified, no access is permitted. Entering `access=@` gives users all access capabilities. See [Access Capabilities](#) on page 2-27.

Order of User Qualification

In qualifying users to access Maestro objects, a user's actual attributes are compared to the user definitions in the order they are entered in the Security file. The first matching definition determines the user's capabilities. For this reason, it is important to order your user definitions from most specific to least specific. See [Sample Security File](#) on page 2-32 for more information.

Order of Object Qualification

Within a user definition, you can use multiple statements for a single object type to assign different access capabilities to different sets of objects. In such cases, the order of object statements is important. They must be ordered from most specific to least specific. For example:

```
#Incorrect:
job    name=@      access=display
job    name=ar@    access=@

#Correct:
job    name=ar@    access=@
job    name=@      access=display
```

See [Sample Security File](#) on page 2-32 for more information.

Variables

The following variables can be used in user definitions:

<code>\$jclgroup</code>	The group name of a job's script file.
<code>\$jclowner</code>	The user name of the owner of a job's script file.
<code>\$master</code>	The name of the master domain manager.
<code>\$owner</code>	The user name of the creator of a schedule and its job definitions.
<code>\$remotes</code>	The names of all standard agent cpus.
<code>\$slaves</code>	The names of all fault-tolerant agent cpus.

<code>\$thiscpu</code>	The name of the cpu on which the user is executing the Maestro command or running the Maestro program.
<code>\$user</code>	The name of the user executing the Maestro command or running the Maestro program.

The variables `$jclgroup` and `$jclowner` are verifiable only if the user is running a Maestro program on the cpu where the job's script file resides. If the program is being run on a different cpu, the user is denied access.

Case-Sensitivity

All keywords and variable names can be entered in either uppercase or lowercase.

Wildcards

Where noted in the syntax descriptions, the following wildcards are permitted:

- ? Replaces one alphabetic character.
- % Replaces one numeric character.
- @ Replaces zero or more alphanumeric characters.

User-Attributes

```
[user-attribute[+|~}user-attribute][...]]
```

user-attribute

An attribute that serves to qualify one or more users. The attributes are:

cpu=cpu[,...]

The cpus or cpu classes on which the users log in. Wildcards are permitted, and the following variables can be used: **\$master**, **\$remotes**, **\$slaves**, **\$thiscpu**. If omitted, all cpus qualify.

group=group[,...]

For UNIX users only. The groups of which the users are members. Wildcards are permitted. If omitted, all groups qualify. This is ignored for Windows NT users.

login=user[,...]

The user names. Wildcards are permitted. If omitted, all users qualify.

+ The 'and' function. The user must have this attribute.

~ The 'and not' function. The user must not have this attribute.

If no attributes are specified, the user definition applies to any user.

The Superuser

If there is no Security file, no users other than **root** can access Maestro objects, and the **root** user has unrestricted access to all objects and can execute all Maestro programs and commands. In the Security file for a network, you may wish to make a distinction between local **root** users and the **root** user on the master domain manager. For example, you can restrict local users to performing operations affecting only their login cpus, while permitting the master **root** user to perform operations that affect any cpu in the network. See [Sample Security File](#) on page 2-32 for more information.

Examples

1. All users whose names begin with "mis":

```
user example1 logon=mis@
```

2. All users in the `sys` group:

```
user example2 group=sys
```

3. All users in the `sys` group who are logged in on a fault-tolerant agent `cpu`:

```
user example3 group=sys + cpu=$slaves
```

Object-Attributes

```
object-type [object-attribute{+|~}object-attribute][...]
```

object-type The object types are:

calendar	User calendars.
cpu	Cpu, domain, and cpu class definitions.
file	Maestro master files.
job	Scheduled jobs.
parameter	User parameters.
prompt	Global (named) prompts.
resource	Scheduling resources.
schedule	Schedules.
userobj	User scheduling object definition.

object-attribute

An attribute that serves to qualify objects of the specified type. See descriptions of attributes below for each object type. If no attributes are specified, all occurrences of the object type qualifies.

- +** The 'and' function. The object must have this attribute.
- ~** The 'and not' function. The object must not have this attribute.

Calendar Attributes

name=calendar[,...] One or more calendar names. Wildcards are permitted. If omitted, all calendars qualify.

Examples

1. All calendars: **calendar** or: **calendar name=@**
2. The "monthend" calendar, and all calendars that have names beginning with "pay", except the "payroll" calendar:

```
calendar    name=monthend,pay@ ~ name=payroll
```

Cpu Attributes

`cpu=cpu[,...]`

One or more cpu, domain, or cpu class names. Wildcards are permitted, and the following variables can be used: `$master`, `$remotes`, `$slaves`, `$thiscpu`. If omitted, all cpus qualify.

Examples

1. The cpu on which the user is logged in:

```
cpu cpu=$thiscpu
```

2. All cpus:

```
cpu cpu=@ OR:cpu
```

3. All fault-tolerant agent and standard agent cpus:

```
cpu cpu=$slaves,$remotes
```

File Attributes

`name=file[,...]`

The names of Maestro master files. Wildcards are permitted. If omitted, all files qualify. The file names are:

<code>calendars</code>	Master calendar file.
<code>cpudata</code>	Master cpu file (also contains cpu classes and domains).
<code>jobs</code>	Master jobs file.
<code>mastsked</code>	Master schedule file.
<code>parameters</code>	Master parameter file.
<code>prodsked</code>	Production schedule file.
<code>prompts</code>	Master prompt file.
<code>resources</code>	Master resource file.
<code>security</code>	Security file.
<code>symphony</code>	Production Control file.

Examples

1. All master files:

```
file name=@
```

```
OR:
```

```
file
```

2. The master cpu file, calendar file, and Security file:

```
file name=cpus,calendars,security
```

Job Attributes

`cpu=cpu` The name of the cpu, or cpu class on which the job runs. Wildcards are permitted, and the following variables can be used: `$master`, `$remotes`, `$slaves`, `$thiscpu`. If omitted, all cpus qualify.

`jcl="path" | "cmd"` The path name of the job's script file, or the command (up to 255 characters). The path or command must be enclosed in quotes (""). Wildcards are permitted. If omitted, all job files and commands qualify.

`logon=user[,...]` The user names under which the jobs run. Wildcards are permitted, and the variables `$owner` and `$user` can be used. If omitted, all user names qualify.

`name=[sched.]job[,...]` The Maestro job names, optionally preceded by their schedule names. Wildcards are permitted. If omitted, all job names qualify.

Examples

1. All jobs:

```
job
```

2. All jobs in the **ap** and **ar** schedules:

```
job name=ap.,ar.@"
```

3. All jobs whose script files are in the path `maestrohome/mis` and begin with `jcl`:

```
job jcl="maestrohome/mis/jcl@"
```

4. All jobs that run on this cpu, except jobs that log in as **root**:

```
job cpu=$thiscpu ~ logon=root
```

5. All jobs that run on any cpu, where the job logs in as this user or the jcl file owner, but not **root**:

```
job cpu=@ + logon=$user,$jclowner ~ logon=root
```

Parameter Attributes

`cpu=cpu` The name of the cpu on which the parameters are defined. Wildcards are permitted, and the following variables can be used: `$master`, `$remotes`, `$slaves`, `$thiscpu`. If omitted, all cpus qualify.

`name=parameter[,...]` One or more parameter names. Wildcards are permitted. If omitted, all parameters qualify.

Examples

1. All parameters with names that begin with "r":

```
parameter name=r@
```

2. All parameters defined on this cpu with names that begin with "u":

```
parameter cpu=$thiscpu + name=u@
```

Prompt Attributes

`name=prompt[,...]` One or more prompt names. Wildcards are permitted. If omitted, all prompts qualify.

Examples

1. All prompts:

```
prompt OR: prompt name=@
```

2. All prompts with names beginning with "mis" or "test":

```
prompt name=mis@,test@
```

3. All prompts except those whose names begin with "sys":

```
prompt name=@ ~ name=sys@
```

Resource Attributes

`cpu=cpu` The name of the cpu or cpu class on which the resources are defined. Wildcards are permitted, and the following variables can be used: `$master`, `$remotes`, `$slaves`, `$thiscpu`. If omitted, all cpus qualify.

`name=resource[,...]` One or more resource names. Wildcards are permitted. If omitted, all resources qualify.

Examples

1. All resources: `resource`
2. All resources on fault-tolerant agent cpus with names beginning with "fox":

```
resource cpu=$slaves + name=fox@
```

Schedule Attributes

`cpu=cpu` The name of the cpu or cpu class on which the schedules run. Wildcards are permitted, and the following variables can be used: `$master`, `$remotes`, `$slaves`, `$thiscpu`. If omitted, all cpus qualify.

`name=sched[,...]` One or more schedule names. Wildcards are permitted. If omitted, all schedules qualify.

Examples

1. All schedules that run on this cpu:
2. All schedules whose names begin with "mis", that run on fault-tolerant agent cpus:

```
schedule cpu=$thiscpu
```

```
schedule name=mis@ + cpu=$slaves
```

User Object Attributes

`cpu=cpu` The name of the cpu on which the user is defined. Wildcards are permitted, and the following variables can be used: `$master`, `$remotes`, `$slaves`, `$thiscpu`. If omitted, all cpus qualify.

`logon=user[,...]` One or more `user def-names` from the user definition. Wildcards are permitted and the following variables can be used: `$owner`, `$user`. If omitted, all users qualify.

Examples

1. All Users:
`userobj`
2. All Users defined on this cpu:
`userobj cpu=$thiscpu`

Access Capabilities

Following is a list of the access keywords for each object type, and the capabilities given to users of the command line interface (CLI) and the graphical interface (GUI).

In Composer's GUI, menu items or buttons are dimmed for objects that the user cannot access. The Composer GUI checks security access before actions can take place, except in the case of **add** for jobs.

In Conman's GUI, security is checked when the command is executed, so menu items and buttons are not dimmed. If a user does not have access to an object, a warning or error is displayed when the action is attempted. In both Conman's and Composer's GUI, lists only display those items to which the user has access.

Object Type	Access Keyword	CLI User Capability	GUI User Capability
calendar	add	na	New Calendar and Save Calendar As
	delete	na	List of Calendars > Actions > Delete
	display	composer display, create	List of Calendars > Actions > Display
	modify	na (See file modify)	List of Calendars > Actions > Modify and Save Calendar As
	use	composer - use calendar in schedules	Schedule Definition > use calendars

Access Capabilities

Object Type	Access Keyword	CLI User Capability	GUI User Capability
cpu (Includes cpu classes and domains)	add	composer add, new	New CPU and Save CPU As New CPU Class and Save CPU Class As New Domain and Save Domain As
	console	conman console	Maestro Conman > View > Console
	delete	composer delete	List of CPUS > Actions > Delete List of CPU Classes > Actions > Delete List of Domains > Actions > Delete
	display	composer display, create	List of CPUS > Actions > Display List of CPU Classes > Actions > Display List of Domains > Actions > Display
	fence	conman fence	SHOWCPUS > Actions > Fence...
	limit	conman limit	SHOWCPUS > Actions > Limit...
	link	conman link	SHOWCPUS > Actions > Link
	modify	composer modify, replace	List of CPUS > Actions > Modify and Save CPU As List of CPU Classes > Actions > Modify and Save CPU Class As List of Domains > Actions > Modify and Save Domain As
	shutdown	conman shutdown	na
	start	conman start	SHOWCPUS > Actions > Start
stop	conman stop	SHOWCPUS > Actions > Stop	
unlink	conman unlink	SHOWCPUS > Actions > Unlink	

Object Type	Access Keyword	CLI User Capability	GUI User Capability
file	build	composer build	na
	delete	na	na
	display	Access Security with dumpsec	na
	modify	Access Security file with makesec; and composer modify calendars, parameters, prompts, and resources	na
job	add	composer add, new	Job Definition > File > Save and Save Job As
	adddep	conman adddep	SHOWJOBS > Actions > Add Dependency
	altpri	conman altpri	SHOWJOBS > Actions > Priority
	cancel	conman cancel	SHOWJOBS > Actions > Cancel Job
	confirm	conman confirm	SHOWJOBS > Actions > Confirm
	deldep	conman deldep	SHOWJOBS > Actions > Delete Dependency
	delete	composer delete	List of Jobs > Actions > Delete
	display	conman display composer display	List of Jobs and SHOWJOBS > Actions > Display
	kill	conman kill	SHOWJOBS > Actions > Kill
	modify	composer modify, replace	List of Jobs > Actions > Modify Job Definition > File > Save and Save Job As
	release	conman release	SHOWJOBS > Actions > Release
	reply	conman reply	SHOWPROMPTS > Actions > Reply
	rerun	conman rerun	SHOWJOBS > Actions > Rerun
	submit	conman submit (docommand, file, job).	Maestro Conman > Submit > Options and SHOWSCHEDULES > Actions > Submit
use	composer - use job in schedule	Schedule Definition > add jobs	

Access Capabilities

Object Type	Access Keyword	CLI User Capability	GUI User Capability
parameter	add	parms to add parameters	Parameter Definitions > Add
	delete	na	Definitions > Delete
	display	composer display, create and parms to display parameters	Parameter Definitions > display
	modify	parms to modify parameters (See also file modify.)	Parameter Definitions > Replace
prompt	add	na	Prompt Definitions > Add
	delete	na	Prompt Definitions > Delete
	display	composer display, create conman recall	Prompt Definitions > display
	modify	na(See file modify.)	Prompt Definitions > Replace
	reply	conman reply	SHOWPROMPTS > Actions > Reply
	use	composer - use prompt in schedule, and conman - add dependencies	Schedule Definition and Job Dependency > add global prompts
resource	add	na	Resource Definitions > Add
	delete	na	Resource Definitions > Delete
	display	composer display, create	Resource Definitions > list
	modify	na (See file modify.)	Resource Definitions > Replace
	resource	conman resource	SHOWRESOURCES > Actions > Change Units
	use	composer - use resource in schedule conman - add dependencies	Schedule Definition and Job Dependency > add resources

Object Type	Access Keyword	CLI User Capability	GUI User Capability
schedule	add	composer add, new	New Schedule
	adddep	conman adddep	SHOWSCHEDULES > Actions > Add Dependency
	altpri	conman altpri	SHOWSCHEDULES > Actions > Priority
	cancel	conman cancel	SHOWSCHEDULES > Actions > Cancel
	deldep	conman deldep	SHOWSCHEDULES > Actions > Delete Dependency
	delete	composer delete	Schedules > Actions > Delete
	display	composer display, create conman display	List of Schedules > Actions > Display and SHOWSCHEDULES
	limit	conman limit	SHOWSCHEDULES > Actions > Limit
	modify	composer modify, replace	List of Schedules > Actions > Modify
	release	conman release	SHOWSCHEDULES > Actions > Release
	reply	conman reply	SHOWPROMPTS > Actions > Reply
submit	conman submit sched	Maestro Conman > Submit > Schedule	
userobj	add	composer add, new	User Definitions > Add
	delete	composer delete	User Definitions > Delete
	display	composer display (password not displayed)	User Definitions > list
	modify	composer modify	User Definitions > Replace
	altpass	conman altpass	Change Password

Sample Security File

Following is a sample Security file. A discussion of the file follows the listing.

```
#####
#           Sample Security File
#####
# (1) APPLIES TO MAESTRO OR ROOT USERS LOGGED IN ON THE MASTER DOMAIN MANAGER.
user mastersm  cpu=$master + logon=maestro,root
begin
# OBJECT          ATTRIBUTES          ACCESS CAPABILITIES
# -----          -
job               access=@
schedule          access=@
resource          access=@
prompt            access=@
file              access=@
calendar          access=@
cpu               access=@
parameter         name=@ ~ name=r@          access=@
userobj           cpu=@ + logon=@          access=@
end
#####
# (2) APPLIES TO MAESTRO OR ROOT USERS LOGGED IN ON ANY CPU OTHER THAN
#       THE MASTER DOMAIN MANAGER.
user sm  logon=maestro,root
begin
# OBJECT          ATTRIBUTES          ACCESS CAPABILITIES
# -----          -
job               cpu=$thiscpu          access=@
schedule          cpu=$thiscpu          access=@
resource          cpu=$thiscpu          access=@
prompt            access=@
file              access=@
calendar          access=@
cpu               cpu=$thiscpu          access=@
parameter         cpu=$thiscpu ~ name=r@  access=@
end
#####
# (3) APPLIES TO USERS LOGGED INTO THE SYS GROUP ON THE MASTER DOMAIN
#       MANAGER.
user masterop  cpu=$master + group=sys
begin
# OBJECT          ATTRIBUTES          ACCESS CAPABILITIES
# -----          -
job               cpu=$thiscpu          access=@
                  + logon=$user
job               cpu=@              access=@
                  + logon=root
                  access=adddep,altpri,cancel,
                  confirm,deldep,release,
                  reply,rerun,submit,use
job               cpu=@
```

```

+ logon=$user,$jclowner
~ logon=root
access=add,adddep,altpri,
cancel,confirm,
deldep,release,reply,
rerun,submit,use
schedule cpu=$thiscpu access=@
schedule cpu=@ access=adddep,altpri,cancel,
deldep,limit,release,
submit
resource access=add,display,
resource,use
prompt access=add,display,reply,use
file access=build
calendar access=display,use
cpu cpu=@ access=@
parameter name=@ ~ name=r@ access=@
end
#####
# (4) APPLIES TO USERS LOGGED INTO THE SYS GROUP ON ANY CPU OTHER THAN
# THE MASTER DOMAIN MANAGER.
user op group=sys
begin
# OBJECT ATTRIBUTES ACCESS CAPABILITIES
# -----
job cpu=$thiscpu access=@
+ logon=$user
job cpu=$thiscpu
+ logon=root access=adddep,altpri,cancel,
confirm,deldep,release,
reply,rerun,submit,use
job cpu=$thiscpu
~ logon=root access=adddep,altpri,cancel,
confirm,deldep,release,
reply,rerun,submit,use
schedule cpu=$thiscpu access=@
resource access=add,display,resource,use
prompt access=add,display,reply,use
file access=build
calendar access=use
cpu cpu=$thiscpu access=console,fence,limit,
link,start,stop,unlink
parameter name=@ ~ name=r@ access=@
end
#####
# (5) APPLIES TO USERS LOGGED INTO THE MIS GROUP ON ANY CPU.
user misusers group=mis
begin
# OBJECT ATTRIBUTES ACCESS CAPABILITIES
# -----
job cpu=$thiscpu access=@
+ logon=$user
job cpu=$thiscpu
+ logon=$jclowner
~ logon=root access=submit,use

```

```

schedule          cpu=$thiscpu          access=add,submit,
                  modify,display
parameter         name=r@          access=@
parameter         name=@          access=display
end
#####
# (6) APPLIES TO ALL OTHER USERS LOGGED IN ON ANY CPU.
user default logon=@
begin
# OBJECT          ATTRIBUTES          ACCESS CAPABILITIES
# -----
job              cpu=$thiscpu
                + logon=$user          access=@
job              cpu=$thiscpu
                + logon=$jclowner
                ~ logon=root          access=submit,use
schedule        cpu=$thiscpu          access=add,submit,
                  modify,display
parameter         name=u@          access=@
parameter         name=@ ~ name=r@   access=display
end
#####

```

Discussion of Sample Security File

Note the order of definitions, from most to least specific. Because of the order, **maestro** and **root** users are matched first, followed by users in the **sys** group, and then users in the **mis** group. All other users are matched with the last definition, which is the least specific.

```

# (1) APPLIES TO MAESTRO OR ROOT USERS LOGGED IN ON THE MASTER DOMAIN MANAGER.
user mastersm cpu=$master + logon=maestro,root

```

This definition applies to **maestro** and **root** users logged in on the master domain manager. They are given unrestricted access to all objects, except parameters that have names beginning with "r". Access to the "r" parameters is given only to users in the **mis** group (see definition #5 below).

```

# (2) APPLIES TO MAESTRO OR ROOT USERS LOGGED IN ON ANY CPU OTHER THAN
#      THE MASTER DOMAIN MANAGER.
user sm logon=maestro,root

```

This definition applies to **maestro** and **root** users to whom definition #1 does not apply—that is, those who are logged in on any **cpu** other than the master domain manager. They are given unrestricted access to all objects on their login **cpu**. Note that prompts, files, and calendars are global in nature, and are not associated with a **cpu**.


```
# (3) APPLIES TO USERS LOGGED INTO THE SYS GROUP ON THE MASTER DOMAIN MANAGER.
user masterop cpu=$master + group=sys
```

This definition applies to users logged into the **sys** group on the master domain manager. They are given a unique set of access capabilities. Note that multiple object statements are used to give these users specific types of access to different sets of objects. For example, there are three `job` statements:

- The first `job` statement permits unrestricted access to jobs that run on the user's login `cpu` (`$thiscpu`), under the user's name (`$user`).
- The second `job` statement permits specific types of access to jobs that run on any `cpu`, and that run as **root**.
- The third `job` statement permits specific types of access to jobs that run on any `cpu`. The jobs must run under the user's name (`$user`), or under the name of the owner of the `jcl` file (`$jclowner`). Jobs that run as **root** are excluded.

```
# (4) APPLIES TO USERS LOGGED INTO THE SYS GROUP ON ANY CPU OTHER THAN
# THE MASTER DOMAIN MANAGER.
user op group=sys
```

This definition applies to **sys** group users to whom definition #3 does not apply—that is, those who are logged in on any `cpu` other than the master domain manager. They are given a set of access capabilities similar to those in definition #3. The exception is that access is restricted to objects on the user's login `cpu` (`$thiscpu`).

```
# (5) APPLIES TO USERS LOGGED INTO THE MIS GROUP ON ANY CPU.
user misusers group=mis
```

This definition applies to users logged into the **mis** group on any `cpu`. They are given a limited set of access capabilities. Because resources, prompts, files, calendars, and `cpus` are omitted, no access is permitted to these objects. These users are given unrestricted access to parameters with names that begin with "r", but are limited to displaying other parameters.

```
# (6) APPLIES TO ALL OTHER USERS LOGGED IN ON ANY CPU.
user default logon=@
```

This definition gives a set of default capabilities to users other than those covered by the preceding definitions. These users are given unrestricted access to parameters with names that begin with "u", but are limited to displaying other parameters. No access is permitted to parameters with names that begin with "r".

dumpsec

The `dumpsec` command will decompile the Security file and send the output to `stdout`.

Security: The user must have `display` access to the Security file.

```
dumpsec -v|-u
dumpsec secfile
```

`-v` Display command version information only.

`-u` Display command usage information only.

`secfile` The name of the file to dump.

Operation

If no options are specified, the operational Security file (`../unison/Security`) is dumped. To create an editable copy of a Security file, redirect the output of the command to another file (see [Examples](#) below).

Examples

1. Display the command version:
`dumpsec -v`
2. Dump the operational Security file to `stdout`:
`dumpsec`
3. Dump the operational Security file to a file named `mysec`:
`dumpsec > mysec`
4. Dump a Security file named `sectemp` to `stdout`:
`dumpsec sectemp`

makesec

The **makesec** command compiles user definitions and installs the Security file. Changes to the Security file take effect when Maestro is stopped and restarted.

Security: The user must have **modify** access to the Security file.

```
makesec -v|-u
makesec [-verify] infile
```

-v	Display command version information only.
-u	Display command usage information only.
-verify	Syntax check the user definitions in <i>infile</i> only—do not install the Security file. (Syntax checking is performed automatically when the Security file is installed.)
<i>infile</i>	The name of a file or fileset containing user definitions. A filename expansion pattern is permitted.

Operation

The command will syntax check and compile *infile*, and then install it as the operational Security file (`./unison/Security`). If the **-verify** option is specified, *infile* is checked for correct syntax, but it is not compiled or installed.

Examples

1. Display the command version:

```
makesec -v
```

2. Create an editable copy of the operational Security file in a file named **tempsec**; modify the user definitions with **vi**; then compile **tempsec** and replace the operational Security file:

```
dumpsec > tempsec
vi tempsec
... make modifications to tempsec ...
makesec tempsec
```

3. Compile user definitions from the fileset `userdef*`, and replace the operational Security file:

```
makesec userdef*
```

3

The Production Cycle

This section explains Maestro production. The first part discusses job execution and job environment. The second part discusses the Maestro production cycle and how to automate it.

Job Execution

Maestro's primary processing task is executing its schedules. A schedule is an outline of batch processing consisting of a list of jobs. Although the term *job* has various meanings in the scheduling environment, Maestro imposes a specific definition: a script file, or command, whose execution is controlled by Maestro. Throughout this manual, the terms *script file*, *job file*, and *jcl file* are used interchangeably.

Scheduled Jobs

Jobs that you wish to schedule for automatic execution by Maestro are defined independent of schedules. Defining a job involves supplying a unique Maestro job name, the path name of the script file, and the Logon name. The job is then placed in a schedule where dependencies that determine when the job and schedule will be launched can be specified.

A dependency is a condition that must be satisfied before a job or schedule will be launched. For example, you can specify that a job, `job2`, cannot be launched until another job, `job1`, successfully completes execution. Maestro always interprets an exit code of zero as successful completion. Any other exit code from `job1` means that it abended, which prevents `job2` from being launched. For more information on dependencies see section 5, *Composing Schedules*.

Defining Jobs within Schedules

Using Maestro's command line interface (CLI), jobs can be defined within schedules using the *autodoc* feature. For more information see [Documenting Jobs in Schedules with auto-doc](#) starting on page 8-65.

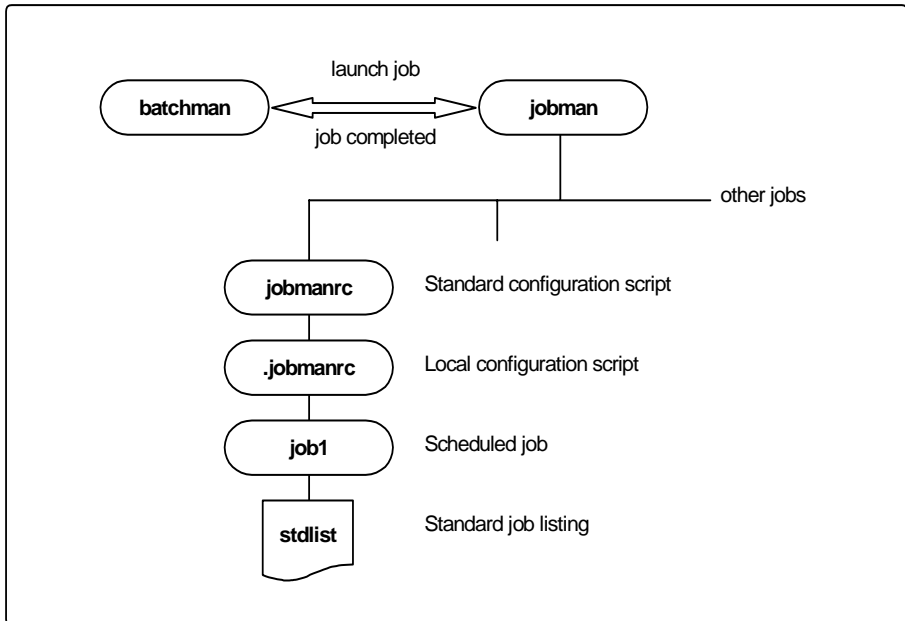


Figure 3-1: Launching Jobs on UNIX

Launching Jobs

Jobs are launched under the direction of the Production Control process Batchman. Batchman resolves all job dependencies to ensure the correct order of execution, and then issues a job launch message to the Jobman process.

Jobman begins by setting a group of environment variables, and then it executes the standard configuration script (*maestrohome/jobmanrc*). If the user is permitted to use a local configuration script, and the script exists as *\$HOME/.jobmanrc*, the local configuration script is also executed. The job itself is then executed either by the standard configuration script, or by the local configuration script.

Note: In the case of extended agent jobs, an access method script is executed instead of the standard configuration script. For more information about extended agents, see appendix A, *Maestro Networks* and appendix C, *Extended Agent Reference*.

Each of the processes launched by Jobman, including the configuration scripts and the jobs themselves, retain the user name recorded with the Logon of the job. In the case of submitted jobs (discussed later), the jobs retain the submitting user's name. To have the jobs execute with the user's environment, be sure to add the user's `.profile` environment to the local configuration script.

Jobman Environment Variables

The variables listed in the table 3-1 are set and exported by Jobman.

Table 3-1: Jobman Environment Variables

Variable Name	Value
HOME	The login user's home directory.
LOGNAME	For NT: %SYSTEMROOT%\SYSTEM32. For a UNIX system: /bin:/usr/bin.
PATH	Path for system commands: /bin:/usr/bin
TZ	The timezone.
UNISON_SHELL	The user's login shell.
UNISON_CPU	The name of this cpu.
UNISON_HOST	The name of the host cpu.
UNISON_JOB	The fully-qualified job name: <i>cpu#sched.job</i>
UNISON_JOBNUM	The job number (ppid).
UNISON_MASTER	The name of the master domain manager.
UNISON_RUN	Maestro's current production run number.
UNISON_SCHED	The schedule name.
UNISON_SCHED_DATE	Maestro's production date (<i>yymmdd</i>).
UNISON_SCHED_EPOCH	Maestro's production date expressed in epoch form.

Standard Configuration Script - jobmanrc

A standard configuration script template named `maestrohome/config/jobmanrc` is supplied with Maestro. It is installed automatically as `maestrohome/jobmanrc`. This script can be used by the system administrator to establish a desired environment before each job is executed. If you wish to alter the script, make your modifications in the working copy (`maestrohome/jobmanrc`), leaving the template file intact. The file contains configurable variables, and comments to help you understand the methodology. Table 3-2 describes the `jobmanrc` variables.

Table 3-2: Jobmanrc Variables

Variable Name	Value
<code>UNISON_JCL</code>	The path name of the job's script file.
<code>UNISON_STDLIST</code>	The path name of the job's standard list file.
<code>UNISON_EXIT</code>	(Settable) If set to yes , the job is terminated immediately if any command returns a non-zero exit code. If set to no , the job continues to execute if a command returns a non-zero exit code. Any other setting is interpreted as no .
<code>LOCAL_RC_OK</code>	(Settable) If set to yes , the user's local configuration script is executed (if it exists), passing <code>\$UNISON_JCL</code> as the first argument. The user may be allowed or denied this option. See Local Configuration Script - \$HOME/jobmanrc on page 3-5 for more information. If set to no , the presence of a local configuration script is ignored, and <code>\$UNISON_JCL</code> is executed. Any other setting is interpreted as no .
<code>MAIL_ON_ABEND</code>	(Settable) If set to yes , a message is mailed to the login user's mailbox if the job terminates with a non-zero exit code. This can also be set to one or more user names, separated by spaces, and 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: <pre>cpu#sched.job jcl-file failed with exit-code Please review standard-list-filename</pre>

Table 3-2: Jobmanrc Variables

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

Local Configuration Script - \$HOME/.jobmanrc

The local configuration script permits users to establish a desired environment for the execution of their own jobs. The script will be executed only under the following conditions:

1. The standard configuration script, `jobmanrc`, must be installed, and the environment variable `LOCAL_RC_OK` must be set to **yes** (see Table 3-2 above).
2. If the file `maestrohome/localrc.allow` exists, the user's name must appear in the file. If the allow file does not exist, the user's name must not appear in the file, `maestrohome/localrc.deny`. If neither of these files exists, the user is permitted to use a local configuration script.
3. The local configuration script must be installed in the user's home directory (`$HOME/.jobmanrc`), and it must have execute permission.

If you intend to use a local configuration script, it must, at a minimum, execute the job's script file (`$UNISON_JCL`). The Tivoli-supplied standard

configuration script, `jobmanrc`, executes your local configuration script as follows:

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

The value of `USE_SHELL` is set to the value of the `jobmanrc` `SHELL_TYPE` variable (see Table 3-2 [above](#)). `IS_COMMAND` is set to `yes` if the job was scheduled or submitted using the `docommand` construct. `EXECIT` is set to `exec` if the variable `USE_EXEC` is set to `yes` (see Table 3-2 [above](#)), otherwise it is null.

The following examples show ways of executing a job's script file, or command, in your local configuration script:

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

An example of a `.jobmanrc` that does processing based on the exit code of the user's job:

```
#!/bin/sh
#
PATH=maestrohome:maestrohome/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 "tellog This is a database error - page the dba"
elif [ $RETVAL -ge 100 ]
then
    conman "tellog Job aborted. Please page the admin"
fi
```

Local Options

Two Local Options (`jm nice` and `jm no root`) can be used to set a nice value for all jobs launched by Jobman, and to enable or disable the launching of root jobs. For information about these variables see [Local Options](#) starting on page 2-8.

Job Termination

The execution of a job is considered successful (SUCC state) if it returns an exit code of zero. A job's execution is considered abended (ABEND state) in the following cases:

1. The job calls `exit` with a non-zero exit code.
2. A command in the job returns a non-zero exit code causing the job to be terminated. This occurs only if the `-e` option is used, or if the `jobmanrc` variable `UNISON_EXIT` is set to `yes`. If the `+e` option is used, or if the `UNISON_EXIT` variable is set to `no`, the job is not terminated on a command failure. In this latter case, the job must explicitly call `exit` with a non-zero exit code to indicate that it abended.

As a general rule, other scripts that are executed within the body of the main job script should be invoked with the `-e` option, or they should explicitly call `exit` with a non-zero exit code to indicate an error.

Standard List Files

A standard list file is created automatically by Jobman for each job it launches. They can be viewed with Conman (see [page 6-51](#)). A standard list file contains header and trailer banners, echoed commands, and the `stdout` and `stderr` output of the job. For example:

```
=====
= JOB                : UX1#MAIL.SENDMAIL
= USER              : xpress
= JCLFILE            : /users/xpress/sendxpmail
= Job Number        : 8027
= Fri Jun  5 12:17:10 1997
=====
...
...          <stdout, stderr and echoed commands>
...
=====
= Exit Status        : 0
= System Time (Seconds) : 1      Elapsed Time (Minutes) : 0
= User Time (Seconds)  : 0
= Fri Jun  5 12:17:12 1997
=====
```

Standard list files are located in the `maestro` user's home directory, and are named as follows:

```
maestrohome/stdlist/yyyy.mm.dd/Oppid[.num]
```

Where:

```
yyyy.mm.dd   Maestro's production date (year, month, day).
```

<i>ppid</i>	The process id (job number).
<i>num</i>	A random number added by Jobman, if necessary, to make the file name unique.

Standard list files are also created for Maestro's production processes. Following is a sample listing (on a UNIX system) of standard list files for March 17, 1997:

```
ls -l stdlist/1997.03.17
-rw-rw---- 1 maestro bin 28005 Mar 17 06:01
MAESTRO
-rw-r--r-- 1 maestro bin 616 Mar 17 05:21 NETMAN
-rw-r--r-- 1 maestro bin 1533 Mar 17 09:49 O3851
-rw-r--r-- 1 maestro bin 889 Mar 17 09:52 O3956
-rw-r--r-- 1 maestro bin 842 Mar 17 09:52 O3961
-rw-r--r-- 1 maestro bin 860 Mar 17 11:50 O3997
-rw-r--r-- 1 maestro bin 858 Mar 17 10:02 O4055
-rw-r--r-- 1 maestro bin 1533 Mar 17 11:19 O5702
```

A job's standard list file can be referenced within the job's script using the `jobmanrc` variable `UNISON_STDLIST` (see Table 3-2 [above](#)).

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

Viewing Remote Standard List Files

The contents of standard list files for jobs that run on other cpus can be viewed using `Conman` (see [Viewing a Job's Standard List File](#) on page 6-51) or with the `conman showjobs` command (described on [page 9-80](#)). Only standard list files for jobs that can be displayed are accessible. Not all Maestro cpus have access to all jobs. The general restrictions are:

- Domain managers have access to all jobs in their domains and subordinate domains.
- Fault-tolerant agents have access to their own jobs. Fault-tolerant agents defined with Full Status mode ON also have access to jobs in their domains.
- Standard agent cpus do not have access to any jobs.

The `morestdl` command (described on [page 10-23](#)) can also be used to display standard lists.

Printing Standard List Files Automatically

You can have the standard list files for jobs printed automatically by adding a few commands to your local configuration script. The following commands, for example, will save the job's return code, and start a background subshell to print the job's standard list file. The `trap` is required to prevent the subshell from terminating when the main shell terminates, and the delay (`sleep`) provides enough time for the trailing banner to be appended to the standard list file.

```
#!/bin/sh
PATH=maestrohome:maestrohome/bin:$PATH
export PATH
# set the trap for logout. This is "signal" 0, but not
# HUP which is signal 1.
trap '(sleep 100;lp $UNISON_STDLIST; cp $UNISON_STDLIST
$HOME) & ' 0
# the ampersand is necessary
/bin/sh -c "$UNISON_JCL"
```

Standard List Files for Interactive Jobs

The `stdin`, `stdout` and `stderr` for a Windows NT interactive job is directed to its associated window. This information is not written to the job's standard list file.

Submitted Jobs

Ad hoc jobs can be submitted by users with the Maestro `at` and `batch` commands. These commands are described in detail in section 10, *Utility Commands*. For each submitted job, `at` and `batch` record the submitting user's environment, create a script file, and add the job to an existing schedule. Although the jobs are not documented like scheduled jobs, they are launched and controlled in exactly the same manner as scheduled jobs.

The Production Cycle

Maestro's processing day begins at the time defined by the Global Option `start time` which is set by default to 6:00 a.m. (see *Global Options* starting on page 2-1). To automate the daily turnover, a schedule named `sfinal` is supplied by Tivoli. This schedule is selected for execution everyday. It runs a job named `Jnextday` that performs pre-production tasks for the upcoming day, and post-production tasks for the day just ended. The commands used to perform these tasks are described in this section.

To help in understanding the turnover process, you may find it helpful to print a copy of the `sfinal` schedule and the `Jnextday` job, and refer to them as you read this section (`maestrohome/config/sfinal` and `maestrohome/Jnextday`).

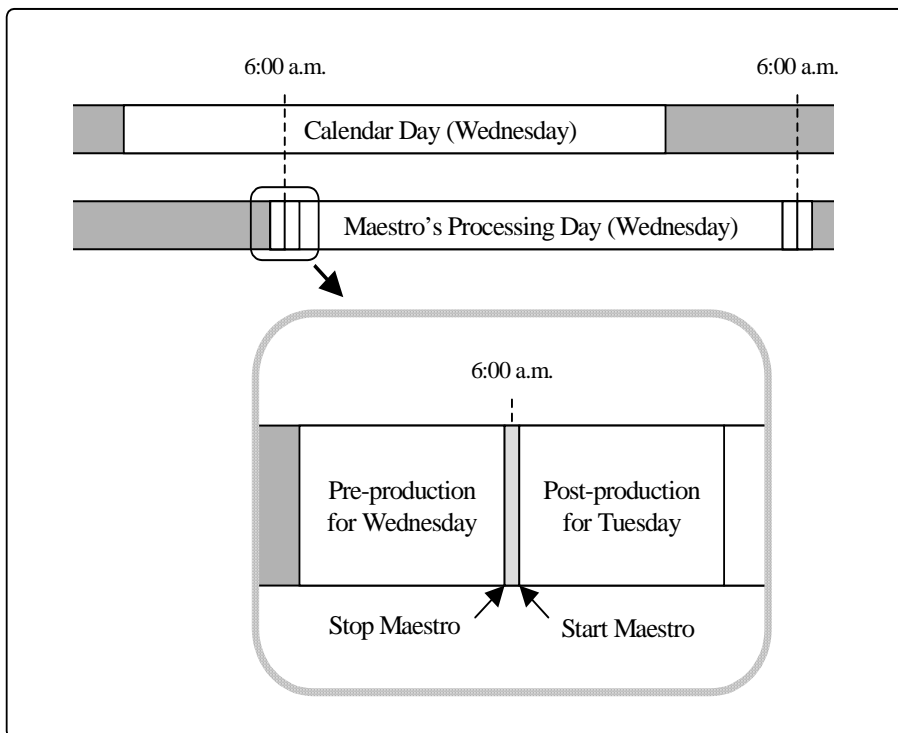


Figure 3-2: Production Cycle

Definitions

<code>mastsked</code> file	The Master Schedule file contains all of the schedules created with Composer. As new schedules are created, each is syntax-checked and added to <code>mastsked</code> .
<code>prodsked</code> file	The Production Schedule file contains the schedules selected for execution on a particular date. The file is built and loaded during the pre-production phase, and then its contents are merged into the Production Control File.
<code>symphony</code> file	The Production Control file contains the scheduling information needed by the Production Control process Batchman. The file is built and loaded during the pre-production phase. During the production phase, it is continually updated to indicate the current status of production processing—work completed, work in progress, and work to be done. To manage production processing, the contents of <code>symphony</code> can be altered and displayed with the console management program Conman.
Batchman process	The Production Control process is started at the beginning of each day to launch jobs in accordance with the information in the <code>symphony</code> file.

Pre-Production

Pre-production processing for an upcoming day involves the following steps. In a Maestro network, the steps are performed on the master domain manager.

1. Run **schedulr** to select the appropriate schedules for execution.
2. Run **compiler** to create an interim Production Control file.
3. Run **reptr** to print the pre-production (planning) reports.
4. Stop Maestro.
5. Run **stageman** to carry forward uncompleted schedules, log the Production Control file, and install the new Production Control file.
6. Start Maestro.

schedulr

Schedulr selects schedules for a specific date from the Master Schedule file (**mastsked**), and copies them to a new Production Schedule file (**prodsked**).

Security: You must have **build** access to the **prodsked** file.

```
schedulr -v|-u
schedulr [-date date ] [-scheds {in|-}] [-prodsked {out|-}]
          [-autodate  ]
```

- v** Display the command version and exit.
- u** Display command usage information and exit.
- date** Select schedules for a specific date. Entered as:
mm/dd/[yy]yy
- autodate** Select schedules for the current system date.
- scheds** Select the schedules named in the *in* file. The schedule names must appear in the file as [*cpu#*]*sched*, with one schedule per line. If a dash is entered instead of a file name, **schedulr** prompts for schedule names at **stdin**. If you omit this option, schedules are selected only for the specified date.
- prodsked** Direct the output of **schedulr** to the *out* file. If a dash is entered instead of a file name, the output is directed to **stdout**. If you omit this option, the output is written to a file named **prodsked** in the current directory.

Operation

If you omit the **-autodate**, and **-date** options, **schedulr** prompts for a date. If you respond to the prompt by hitting Return only, no date is assumed and no schedules are selected automatically.

Warning Message

30 **Error, in schedule: *name*, calendar *name* is not in the calendars database.**

When selecting schedules for a day's production, an undefined calendar was encountered in a schedule. The schedule is not selected for execution, unless by another calendar, day of the week, or specific date.

Examples

1. Select schedules for today's date, plus the schedules named in the file `myskeds`:

```
schemulr -autodate -schems myskeds
```
2. Select schedules for February 15, 1997, do not prompt for extra schedule names, and write the output to the file `myprodsked`:

```
schemulr -date 2/15/97 -prodsked myprodsked
```
3. Select schedules for February 15, 1997, and prompt for extra schedules:

```
schemulr -date 2/15/97 -schems -
```
4. Prompt for the schedule date, and extra schedules:

```
schemulr
Enter schedule date: 4/14/97
Enter a list of extra schedules
Schedule name: site1#sked2
Schedule name: <Return>
<list of schedules selected...>
End of Program
```

compiler

Compiler converts the Production Schedule file (`prodsked`) into an interim Production Control file (usually named `symnew`).

```
compiler -v|-u
compiler [-date date] [-input in] [-output out]
```

- v** Display the command version and exit.
- u** Display command usage information and exit.
- date** The production date to be recorded in the interim Production Control file. See *Operation* below.
- input** The name of the Production Schedule file. If this option is omitted, the default is `prodsked` in the current directory.
- output** The name of the interim Production Control file. If this option is omitted, the default is `symnew` in the current directory.

Operation

If you omit the `-date` option, `symnew` is given the same date as that recorded in the `prodsked` file. If there is no date in `prodsked`, the current system date is used. The date in `symnew` is the date the Production Control process will begin executing the Production Control file. The ability to enter different dates can be used to set up processing for past or future dates.

Warning Messages

The following messages are produced by **compiler** to indicate missing scheduling objects. The messages are normally found in the standard list file for the **Jnextday** job.

```
5 ... Undefined parameter in "schedule"; not replaced.
```

A parameter called for in a schedule does not exist in Maestro's master file. No substitution occurs and the parameter string itself is used.

```
102 ... Job name is not found in database. Added a dummy job in FAIL state.
```

A job named in a schedule does not exist in Maestro's master file. A dummy job of the same name is placed in the production schedule with a

priority of zero and a state of FAIL.

```
103 ... Prompt name not found. Added prompt name in symphony.
```

A prompt named in a schedule does not exist in Maestro's master file. A dummy prompt containing the following text is used instead:

```
Prompt name was not found in database. This is dummy text. Do you  
want to continue (Y/N).
```

```
104 ... Resource name for cpu name not found in database.  
Added resource name with 0 units.
```

A resource named in a schedule does not exist in Maestro's master file. A dummy resource with zero available units is used instead:

```
106 ... Cpu name does not exist in cpu database. Ignoring  
schedule name.
```

A schedule is defined to run on a cpu that does not exist. The schedule is ignored and not placed in the production schedule.

Examples

1. Compile `prodsked` into `symnew`:

```
compiler
```

2. Compile `prodsked` into `symnew`, and enter a production date of May 15, 1998:

```
compiler -date 5/15/98
```

3. Compile `mysked` into `mysym`:

```
compiler -input mysked -output mysym
```

reptr

Reptr is used to print pre- or post-production reports. Following a run of **compiler**, the pre-production reports should be selected, so that the planned processing for a new day can be reviewed. For samples of the reports, see section 7, [Reports](#).

```
reptr [-v|-u]
reptr -pre [-{summary|detail}] [symfile]
reptr -post [-{summary|detail}] [logfile]
```

<code>-v -V</code>	Display the command version and exit.
<code>-u -U</code>	Display command usage information and exit.
<code>-pre</code>	Print the pre-production reports (09A and/or 09B).
<code>-post</code>	Print the post-production reports (10A and/or 10B).
<code>-summary</code>	Print the summary reports (09A or 10A). If <code>-summary</code> and <code>-detail</code> are omitted, both sets of reports are printed.
<code>-detail</code>	Print the detail reports (09B or 10B). If <code>-summary</code> and <code>-detail</code> are omitted, both sets of reports are printed.
<i>symfile</i>	For pre-production reports, the name of the Production Control file from which reports will be printed. The default is <i>symnew</i> in the current directory.
<i>logfile</i>	For post-production reports, the name of the Production Control file or log file from which the reports will be printed. Note that log files are stored in the <i>shedlog</i> directory. The default is the current Production Control file (Symphony).

Operation

If you enter the command without options, both pre and post reports are printed. If you omit `-summary` and `-detail`, both sets are printed.

Examples

1. Print the pre-production detail report from the *symnew* file:

```
reptr -pre -detail
```

-
2. Print the pre-production summary report from the file `mysym`:

```
reptr -pre -summary mysym
```

3. Print the post-production summary report from the log file `M199703170935`:

```
reptr -post -summary schedlog/M199703170935
```

4. Print the post-production detail and summary reports from the log file `M$DATE`:

```
reptr -post schedlog/M$DATE
```

The `$DATE` variable contains the date-time stamp used by **stageman** to create the log file name. See **stageman** [Examples](#) on page 3-22 for more information.

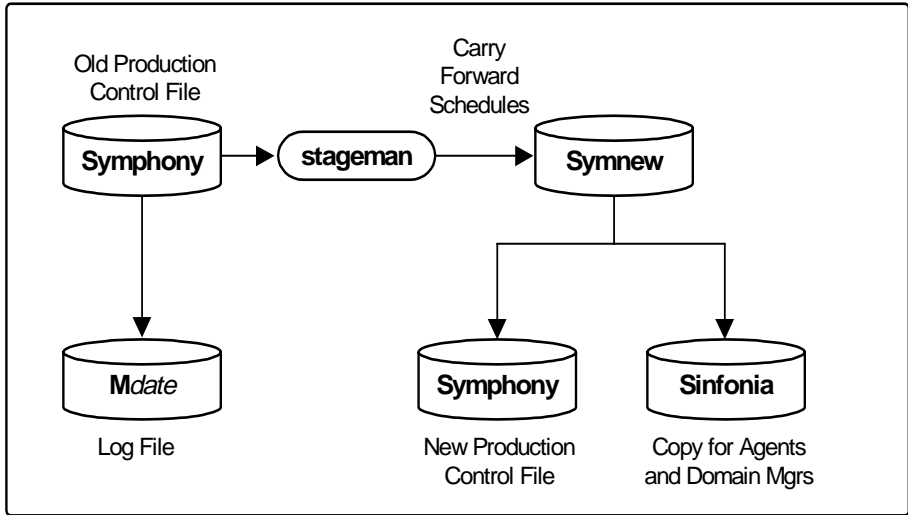


Figure 3-3: stageman Command

stageman

Stageman carries forward uncompleted schedules, logs the old Production Control file, and installs the new Production Control file. In a network, a copy of the Production Control file—called **sinfonia**—is also created for fault-tolerant agent cpus.

Security: You must have **build** access to the **symphony** file.

```

stageman -v|-u
stageman [-carryforward {no } ] [-log logfile ] [symnew]]
           {yes} [-nolog      ]
           {all}
  
```

- v Display the command version and exit.
- u Display command usage information and exit.
- carryforward Define the type of carry forward as follows:
 - no Do not carry forward any schedules.
 - yes Carry forward only those uncompleted schedules that are Carry Forward enabled.

<code>all</code>	Ignore Carry Forward enabling in schedules, and carry forward all uncompleted schedules.
<code>-log</code>	Log the old Production Control file, and give the log file this name. See Log File Names below for more information.
<code>-nolog</code>	Do not log the old Production Control file. See Log File Names below for more information.
<code>symnew</code>	The name of the file created by compiler . If omitted, the file <code>symnew</code> in the current directory is used.

Operation

If you omit the `-carryforward` option, the default for carry forward is determined by the `carryforward` Global Option. See [Carry Forward Options](#) starting on page 2-6 for more information.

Stageman also determines which script files can be deleted for jobs submitted with Maestro's `at` and `batch` commands (UNIX only). These are jobs that were not carried forward. The files are actually deleted by Batchman when it starts up for the new day.

If Batchman or Mailman are accessing the `symphony` file, **stageman** displays the message:

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

To continue, stop Batchman or Mailman, and rerun **stageman**. If **stageman** aborts for any reason, you must rerun both **compiler** and **stageman**.

Users running Conman at the time `symphony` is being switched are sent the message:

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

Some Conman commands executed during the switch may not execute properly because the target jobs or schedules were not carried forward.

Log File Names

Production Control log files are stored in the `schedlog` directory. The default naming convention used by **stageman**, when the `-log` and `-nolog` options are omitted, is as follows:

```
maestrohome/schedlog/Myyyyymmddhhtt
```

where: `yyyymmdd` The year, month and day.
 `hhtt` The hour and minute.

The same naming convention is coded into the `maestrohome/Jnextday` script supplied by Tivoli. If you wish, you can change the naming convention when you automate the production cycle using the `sfinal` schedule and the `Jnextday` job. For more information see [Automating the Production Cycle](#) starting on page 3-28.

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

Carry Forward Options

For a complete description of the options that control carry forward, refer to [Carry Forward Options](#) starting on page 2-6. The Carry Forward schedule option is specified in the Schedule Definition window's Options panel (see [page 5-15](#)).

Schedules Carried Forward

Schedules that are carried forward retain the Carry Forward option enabled, and therefore, may be carried forward again. If a non-SUCC schedule is carried forward, and it continues to terminate in a state other than SUCC, it may be carried forward indefinitely, unless its Until time expires or it is cancelled.

For Carry Forward to work properly in a network, the master domain manager's Production Control file must be updated with the latest schedule status from its agents and subordinate domain managers. This can be accomplished by executing a `conman "link @"` command on the master domain manager prior to executing `stageman`.

Schedule Names

Schedules that are carried forward are renamed as follows. If the global option `expanded version` is set to no:

`CFyjjjnn` where: `y` last digit of the year
 `jjj` Julian date
 `nn` sequence number (00-99, AA-ZZ)

If the global option `expanded version` is set to yes:

<code>CFyjjnxxxxxxxxxx</code>	where:	<code>y</code>	last digit of the year
		<code>jjj</code>	julian date
		<code>nn</code>	sequence number (00-99, AA-ZZ)
		<code>xx..xx</code>	random alpha string

For information about the global option `expanded version`, see [page 2-3](#).

The original schedule names are listed in the Dependencies column of a Console Manager Showschedules display.

Carry Forward Prompts

To retain continuity when carrying schedules forward, **stageman** creates special prompts in the new Production Control file to account for disconnected Follows dependencies. These prompts are issued after the new processing day begins, when Batchman checks to see if the job or schedule is ready to launch. They can be replied to with a `conman reply` command. The following is an example of a Carry Forward prompt in a `conman showprompts` display:

```
INACT 12 (SYS1#CF4123AA) follows SYS1#SKED3 satisfied?
```

This prompt indicates that a schedule from the previous day was carried forward as `CF4123AA`, and that it follows a schedule named `sked3` which was not carried forward. 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 schedule (<code>sked3</code>) 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 schedule (<code>sked3</code>) had completed successfully. The dependency is satisfied.

Examples

1. Carry forward all uncompleted schedules (regardless of the Carry Forward option's status), log the old `symphony` file, and create the new `symphony` file:

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

2. Carry forward uncompleted schedules as defined by the `carryforward` Global Option, do not log the old `symphony` file, and create a new Production Control file named `mysym`:

```
stageman -nolog mysym
```

Post-Production

Post-production processing for the preceding day involves the following steps. In a Maestro network, this is performed on the master domain manager.

1. Run **reptr** to print the post-production reports. See *reptr* on page 3-16.
2. Run **logman** to log job statistics.

logman Command

Logman is used to log job statistics from a Production Control log file.

```
logman -v|-u
logman [-smooth percent] [-minmax {elapsed|cpu}] logfile
```

-v	Display the command version and exit.
-u	Display command usage information and exit.
-smooth	Use a weighting factor that favors the most recent job run when calculating the normal (i.e., average) run time for a job. This is expressed as a percentage. For example, "-smooth 40" will apply a weighting factor of 40% to the most recent job run, and 60% to the existing average. The default is zero.
-minmax	Define how the minimum and maximum job run times are logged and reported.
elapsed	Base the minimum and maximum run times on elapsed time.
cpu	Base the minimum and maximum run times on cpu time.
	See <i>Elapsed Time vs. Cpu Time</i> on page 3-24 for more information.
<i>logfile</i>	The name of the Production Control file or log file from which job statistics will be extracted.

Operation

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

Elapsed Time vs. Cpu Time

Elapsed time, expressed in minutes, is greatly affected by system activity, and includes both the amount of time a job made use of the cpu and the intervals the job had to wait for other processes to release the cpu. In periods of high system activity, for example, a job may have a long elapsed time, and yet use no more cpu time than in periods of low system activity. On the other hand, cpu time, expressed in seconds, is a measure of the actual time a job made use of the cpu, and does not include the intervals when the job was waiting.

Maximum and minimum run times are listed in the following format on the Job Details Listing, Report 01 (see *Sample Reports* starting on page 7-12 for a sample of Report 01).

	Elapsed(mins)	CPU(secs)	
...	
Maximum	hh:mm:ss	seconds	(On run-date)
Minimum	hh:mm:ss	seconds	(On run-date)

If you run **logman** with the **-minmax elapsed** option, the maximum and minimum run times and dates are based solely on a job's elapsed time. 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. The cpu times, in this case, will not necessarily indicate their maximum and minimum extremes.

If you run **logman** with the **-minmax cpu** option, the maximum and minimum run times and dates are based solely on a job's cpu time. 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. The elapsed times, in this case, will not necessarily indicate their maximum and minimum extremes.

If you run **logman** without the **-minmax** option, 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.

Examples

1. Log job statistics from the log file `M199703170935`:

```
logman schedlog/M199703170935
```

2. Log job statistics from the log file `M$DATE` based on elapsed time, giving the most recent job runs a weight of 40% when calculating normal (average) run times:

```
logman -smooth 40 -minmax elapsed schedlog/M$DATE
```

The `$DATE` variable contains the date-time stamp used by **stageman** to create the log file name. See **stageman** [Examples](#) on page 3-22 for more information.

Production

The production phase begins when a Conman Start command is executed following pre-production processing. This starts the Production Control process (Batchman) which begins launching jobs in accordance with the information contained in the `symphony` file.

During the processing day, the production environment can be monitored and controlled with the Console Manager program. The graphical program is described in section 6, *Managing Production*, and the command line program is described in section 9, *Conman Command Line Reference*. In general, Console Manager commands either display or alter information in the `symphony` file.

Starting and Stopping In a Network

Communications between computers in a Maestro network is handled by the Netman process. Netman can be started using the **StartUp** utility, and can be stopped using the Console Manager's **shutdown** function. To automate the start up process, **StartUp** should be installed in the `/etc/rc` file on every computer to start Netman when a computer is rebooted.

Starting and Stopping for next day processing

At the end of the day, during pre-production processing for the next day, `link` and `stop` commands are executed on the master domain manager to link to and stop all cpus.

After the new **Symphony** file has been created on the master domain manager, a `start` is executed. This causes autolinked domain managers, fault-tolerant agents and standard agents to be initialized and started. Those that are not autolinked are initialized and started when a `link` is executed on the master domain manager. Domain managers, in turn, start their agents and subordinate domain managers.

New day processing is usually automated and handled by the `sfinal` schedule and the `Jnextday` job. See [Automating the Production Cycle](#) starting on page 3-28 for more information.

Stopping and Restarting during the day

Production on a cpu can be stopped and restarted during the day with `stop` and `start` commands. A `stop` command stops all production processes except **Netman**. A `start` command starts all production processes, including **Netman** if it is not already running.

If for any reason it is necessary to stop Netman on a cpu, you can, on that cpu, execute a `shutdown` command. `shutdown` stops the entire Maestro process tree. To restart following `shutdown`, you must either execute the `startup` command, or issue a Console Manager `start` command, locally on the cpus you wish to restart. `startup` will start **Netman** only. A `start` command will start the entire process tree.

For more information about the Console Manager refer to section 6, *Managing Production*. For information about command-line commands, refer to section 9, *Conman Command Line Reference*. For information about the `startup` command refer to page 10-32. For information about Maestro network operation refer to appendix A, *Maestro Networks*.

Automating the Production Cycle

Pre and post-production processing can be fully automated by scheduling a job, or a set of jobs, to run at the end of each day. A sample of such a job is provided as *maestrohome/Jnextday*. You may wish to print a copy of this script file to aid in understanding the processing steps. This job will:

1. Run **schedulr** to select the appropriate schedules for the new day.
2. Run **compiler** to create an interim Production Control file.
3. Run **reptr** to print the pre-production reports.
4. Stop Maestro processing.
5. Run **stageman** to carry forward uncompleted schedules, log the old Production Control file, and install the new file.
6. Start Maestro processing for the new day.
7. Run **reptr** to print the post-production reports from the most recent log file.
8. Run **logman** to log job statistics from the most recent log file.

The *Jnextday* job is included in a schedule named *final*, which is selected everyday for execution. The schedule is added during the initial set up phase—see *Starting Maestro for the First Time* on page 1-15.

You can create your own schedule and job(s) modeled after those supplied by Tivoli. If you choose to do so, consider the following:

- If you choose to change the way **stageman** generates Production Control log file names, remember that **reptr** and **logman** must use the same names.
- If you would like to print the pre-production reports in advance of a new day, you can split the *Jnextday* job into two jobs. The first job will execute **schedulr**, **compiler** and **reptr**. The second job will stop Maestro, execute **stageman**, start Maestro, and execute **reptr** and **logman**. The first job can then be scheduled to run at any time prior to the end of day, while the second job is scheduled to run just prior to the end of day.

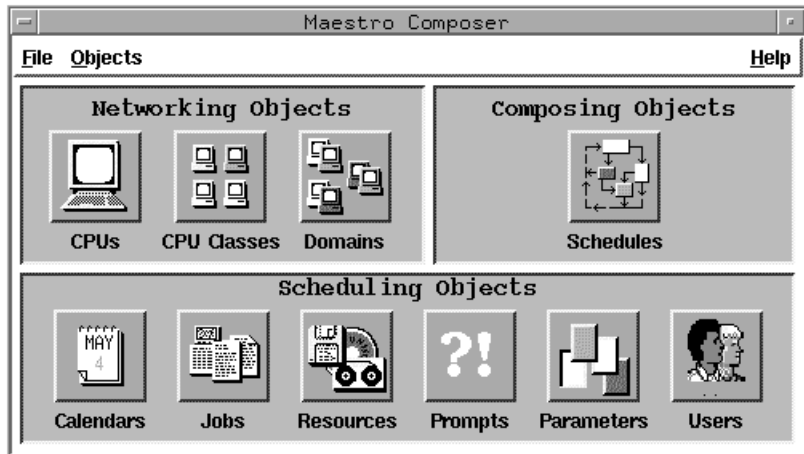
4

Composing Scheduling Objects

Scheduling objects—cpus, cpu classes, jobs, calendars, parameters, prompts, resources, and users—are managed with the Composer program. This section describes Composer basics, and explains how to define your objects. Scheduling objects are used when you write your schedules. Schedule definition and management are described in section 5, [Composing Schedules](#).

Composer Interface Basics

Composer is used to compose and schedule all of your production. The graphical user interface (GUI) is described in this section. For information about the command line interface (CLI) see section 8, [Composer Command Line Reference](#).



Running Composer

To run the Composer GUI program, log in as `maestro`, set your `DISPLAY` variable, and enter the following:

```
gmaestro [-backstore]
```

Where `-backstore` causes graphical information to be stored locally to improve performance when windows need to be redrawn. Note that some X servers may not provide adequate support for this feature. If your windows are only partially redrawn, or images are missing when using the option, discontinue using it.

On the Maestro Main Window, click the Composer button. The Maestro Composer window opens.

The Maestro Composer window can be opened directly from the command line by typing:

```
gcomposer [-backstore]
```

Using the Composer Main Window

Click an object icon or select the object's name from the Objects menu to open a List or Definition window for that object. Some objects—`cpus`, `cpu` classes, domains, calendars, jobs, and schedules—display a List window first; other objects—parameters, prompts, resources, and Users—display a Definition window with an integrated list of all currently defined objects. Each scheduling object is described in detail later in this section. Schedules and schedule dependencies are discussed in section 5, *Composing Schedules*.

To display the Composer program's version, select On Version... from the Help menu. The Version dialog opens showing Composer's Version and Revision. Click OK to close the dialog.

To exit Composer, select Quit Composer from the File menu.

Common Composer Elements

Common types of components found throughout Composer are described below. When necessary, differences are discussed elsewhere in the manual.

Status Bar and Error Messages

At the bottom of many Composer windows there is a status bar that displays error, warning, and informational messages. An audible beep accompanies any error or warning messages displayed.

CPU MISAGT saved to database.

If an invalid selection is made, an error message appears in the status bar. Error, warning and informational messages also appear in pop-up dialog boxes. On windows with an Add... button, if you attempt to add an invalid selection, the Add... button is disabled. Invalid characters typed in an entry box are rejected with a beep.

Reserved Words

Composer does not issue specific warnings if scheduling language keywords are used as names of schedules or scheduling objects. However, the use of keywords can result in errors. Avoid using these keywords when defining schedules and scheduling objects:

at	carryforward	confirmed	cpuname	day(s)
end	every	except	follows	go
hi	in	limit	maestro	needs
node	on	os	opens	order
priority	prompt	schedule	server	until

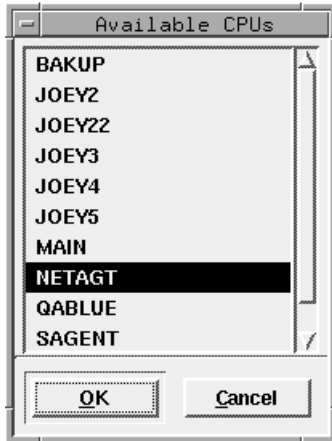
Security

The Composer GUI enforces Maestro security by disabling or omitting items on windows. Menu options and command buttons are enabled according to the user's security access. Lists contain only those items the user has proper security access to see. Also, warning and error messages are issued if a user attempts to perform an action he does not have security access for.

See [*Maestro Security*](#) starting on page 2-16 for complete security information.

Selecting Items from Available Dialogs

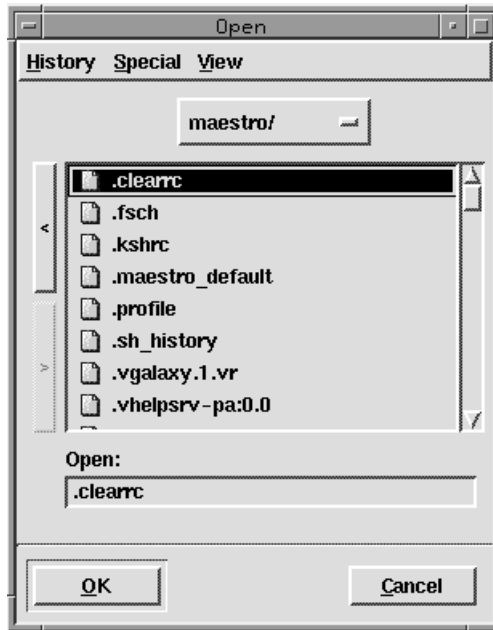
Available dialogs are used to select an item for an entry field and are generally opened from a definition window. The Available CPUs dialog is pictured below.



To select an item in an Available dialog, click on it in the list and then click OK to enter the selected item into the field of the corresponding window or dialog box. Double-clicking on a list item performs the same function as selecting an item and clicking OK. Click Cancel to close the Available dialog and make no selection.

Selecting a File with the Browser

When a file pathname needs to be selected, a file browser can be opened by clicking the Browse... button.



The dialog is the standard file browser for the system. For instructions on how to use the browser, refer to the system's on-line help or documentation.

Closing a Window Without Saving

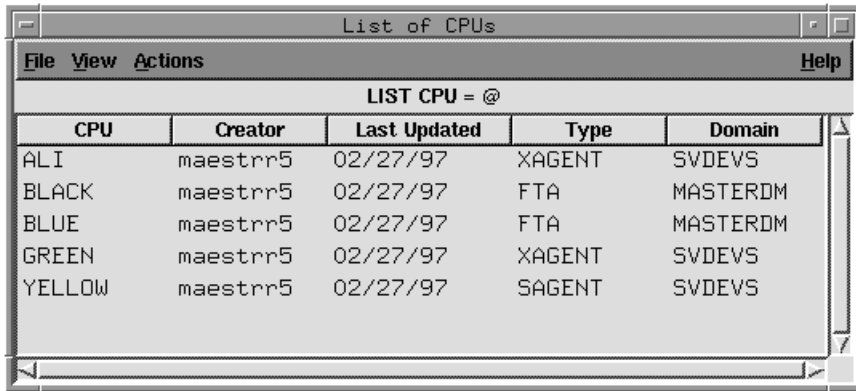
If you attempt to close a window and unsaved changes have been made, a warning appears indicating that your changes will be lost. Click OK to close the window and NOT save the changes. Click Cancel to return to the window where the unsaved changes exist.

Printing and Displaying Data

Data currently being edited cannot be displayed in Display windows until it is saved. Print commands print the currently displayed data, even if it is unsaved.

Maestro Cpus

Maestro cpu definitions are used to assign unique names to your systems, and define the communication links between them. A cpu definition is required for each system in a network. Cpu definitions are maintained with the Composer program on the master domain manager. Click the CPUs button or select CPUs... from the Objects menu on the Maestro Composer main window to open the List of CPUs window. The window displays existing cpus and provides menus for editing, creating, and viewing cpu definitions.



LIST CPU = @				
CPU	Creator	Last Updated	Type	Domain
ALI	maestr5	02/27/97	XAGENT	SVDEVS
BLACK	maestr5	02/27/97	FTA	MASTERDM
BLUE	maestr5	02/27/97	FTA	MASTERDM
GREEN	maestr5	02/27/97	XAGENT	SVDEVS
YELLOW	maestr5	02/27/97	SAGENT	SVDEVS

Cpus are displayed in the following format:

- CPU** The name of the cpu.
- Creator** The login of the creator of the cpu definition.
- Last Updated** The date the cpu definition was last edited.
- Type** The type of cpu: FTA (fault-tolerant agent), SAGENT (standard agent), or XAGENT (extended agent).
- Domain** The name of the cpu's domain.

Creating a New Cpu Definition

To create a new cpu definition, select New... from the Actions menu on the List of CPUs window. The New CPU dialog opens. A New CPU dialog can also be opened by selecting New... from the File menu on the CPU Definition window.



Enter the name of the cpu you want to define in the entry box. The name can contain up to sixteen alphanumeric, dash (-), and underscore (_) characters starting with a letter. Click OK to open a CPU Definition window for the new cpu. All of the fields in the CPU Definition are cleared or set to their default values and the cpu name is displayed in the title bar of the window. If the cpu name already exists, you are not allowed to continue. Click Cancel to close the New CPU dialog and create no new cpu. See [CPU Definition](#) on page 4-9 for information on completing the cpu definition.

Modifying a Cpu Definition

To modify a cpu definition, select the cpu in the List of CPUs window and then Modify... from the Actions menu. The CPU Definition window for the selected cpu opens permitting you to alter its characteristics. See [CPU Definition](#) on page 4-9 for information on modifying the cpu definition.

Displaying a Cpu Definition

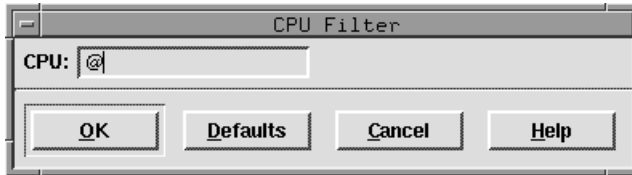
To display a cpu definition, select the cpu in the List of CPUs window and then Display... from the Actions menu. A display-only window for the selected cpu opens showing the cpu definition. See [CPU Definition](#) on page 4-9 for information on the cpu definition.

Deleting a Cpu Definition

To delete a cpu definition, select the cpu in the List of CPUs and then Delete... from the Actions menu. A dialog opens confirming that you want to delete the selected cpu. Click Yes to delete the cpu definition from the database, or click No to take no action.

Filtering the List of CPUs

To filter the cpus displayed in the List of CPUs window, select Filter... from the View menu. The CPU Filter dialog opens.



In the CPU text entry area, enter the string you want Composer to use to filter the list. Wildcards are permitted. Click Defaults to enter the "@" wildcard in the box, which results in a listing of all cpus. Click OK to filter the List of CPUs based on the selections and close the dialog. Click Cancel to close the Filter dialog and take no filter action.

Cpu Filter Example

The selections below filter the List of CPUs window to display only cpus that contain `site` in the first four characters of the name:



CPU Definition

The Cpu Definition and CPU Display windows are used to edit and display Maestro cpu definitions. The top Definition panel is the same for all agent types. The lower Maestro Options panel changes for different types of agents.

Note: For extended agents from Tivoli, be sure to refer to the documentation that accompanies the products.

The screenshot shows a dialog box titled "ADMIN CPU Definition". At the top, there is a menu bar with "File" and "Help". Below the menu bar is a title bar "ADMIN Definition". The main content area is divided into several sections:

- Node:** A text field containing "oregon.unison.com".
- TCP Address:** A text field containing "31111".
- Operating System:** A group of radio buttons with labels "UNIX", "MPE-iX", "MPE-V", "WNT", and "OTHER". "UNIX" is selected.
- Description:** A text field containing "A fault-tolerant agent."
- Domain:** A text field containing "REGION2" and a button labeled "Domains...".
- Maestro Options:** A section containing:
 - Type:** A group of radio buttons with labels "Fault Tolerant Agent", "Standard Agent", and "Extended Agent". "Fault Tolerant Agent" is selected.
 - CHECKED:** "AUTO Link", "Resolve Dependencies", and "Full Status".
 - UNCHECKED:** "Ignore".
 - Server:** A text field containing "2".

Definition Panel

- Node** Enter the node name or the IP address of the cpu. Fully-qualified domain names are accepted. This is a required field.
- TCP Address** Enter the TCP port number of Netman on this cpu. If omitted, the default is 31111. Tivoli recommends leaving this field at its default value unless directed to do otherwise. (The

port number of Netman on a cpu is defined by the Local Option `nm port`. See *Local Options* starting on page 2-8 for more information).

Operating System

Select the os type of this cpu: UNIX, MPE-IX, MPE-V, WNT, or OTHER.

Description

Enter a free-form description of the cpu (up to 40 characters).

Domain

Enter the name of the Maestro domain of the cpu. Click on the **Domains...** button to display a list of available domains from which to choose.

The default for the master, a fault-tolerant agent, or a standard agent is the master domain-- defined by the global option **master domain**. The default for a domain manager is the domain in which it is defined as the manager. The default for an extended agent is the domain of its host cpu.

Maestro Options

Type

Select the cpu type as one of the following:

Fault Tolerant Agent

Standard Agent

Extended Agent

Use Fault Tolerant Agent for domain managers, and backup domain managers.

AUTO Link

For a fault-tolerant agent or standard agent, select AUTO Link to have its communication link opened automatically when its domain manager is started. For a domain manager, select AUTO Link to have communication links opened automatically when its agents are started.

AUTO Link is useful primarily during the initial start up sequence at the beginning of each new day. At that time, the master domain manager creates a new production schedule, and distributes it to subordinate domain managers and agents that have their AUTO Link flags turned on. The other domain managers distribute the new production file to agents in their domains if their AUTO Link flags are turned on. The links for cpus that do not have the AUTO Link flag turned on, must be opened manually using the Console Manager's **Link** function.

-
- Ignore** Select this option only if you want Maestro to ignore this cpu definition. This option is useful if you want to pre-define schedules, jobs, and objects for a system that has not yet arrived.
- Resolve Dependencies**
For fault-tolerant agents only. Select this option to have the agent's Production Control process operate in Resolve All Dependencies Mode. In this mode, the agent tracks dependencies for its all jobs and schedules, including those running on other cpus. Note that **Full Status** must also be selected so that the agent is informed about activity on other cpus.
- If this option is not selected, the agent tracks dependencies for its own jobs and schedules only. This reduces cpu usage by limiting processing overhead.
- To keep the agent's Production Control file at the same level of detail as its domain manager select both **Full Status** and **Resolve Dependencies** options. Always select these options for backup domain managers. See **Full Status** [below](#).
- Full Status** For fault-tolerant agents only. Select this option to have the link from the domain manager operate in Full Status mode. In this mode, the agent is kept updated about the status of jobs and schedules running on other cpus in the network.
- If this option is not selected, the agent is only informed about the status of jobs and schedules on other cpus that affect its own jobs and schedules. This can improve operation by reducing network traffic.
- To keep the agent's Production Control file at the same level of detail as its domain manager select both **Full Status** and **Resolve Dependencies** options. Always select these options for backup domain managers. See **Resolve Dependencies** [above](#).
- Server** For fault-tolerant and standard agents only. Leave this field blank for domain managers. This identifies a server (**Mailman**) process on the domain manager that will send messages to the agent. This can be a letter or a number (A-Z or 0-9). The IDs are unique to each domain manager, so you can use the same IDs for agents in different domains without conflict. If more than 36 server ids are required in a domain, consider dividing it into two or more domains.

If a server ID is not specified, messages to a fault-tolerant or standard agent are handled by a single **Mailman** process on the domain manager. Entering a server ID causes the domain manager to create an additional **Mailman** process. The same server ID can be used for multiple agents. The use of servers reduces the time required to initialize agents, and generally improves the timeliness of messages. As a guide, additional servers should be defined to prevent a single **Mailman** from handling more than eight agents.

Host CPU

This option is required for standard and extended agent cpus, and is not used for other cpu types. Enter the name of the host cpu, or select one from the list provided by clicking the **CPUs...** button.

For a standard agent, the host must be its domain manager. For an extended agent, the host is the cpu where the access method resides, which can be any cpu type except another extended agent. See appendix A, *Maestro Networks* and appendix C, *Extended Agent Reference* for more information on extended agents.

Access

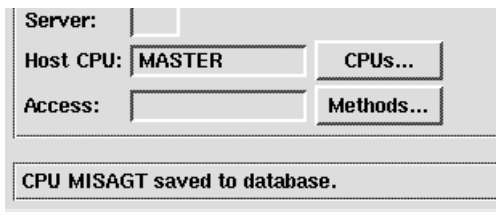
This option is required for extended and Network agent cpus, and is not used for other cpu types. Enter the name of the access method (up to eight characters starting with a letter), or select one from the list provided by clicking the **Methods...** button. The access method is used by the host to communicate with the extended agent.

For Local UNIX extended agents, enter `unixloc1`. For Remote UNIX extended agents, enter `unixrsh`. For Network agents, enter `netmth`.

When a cpu definition is saved, the existence of the access method (`maestrohome/maestro/methods/methodname`) on the local computer is checked. If the file does not exist, a warning is issued. See appendix A, *Maestro Networks* and appendix C, *Extended Agent Reference* for more information on extended agents. See appendix D, *Internetwork Dependencies* for more information on network agents.

Saving a CPU Definition

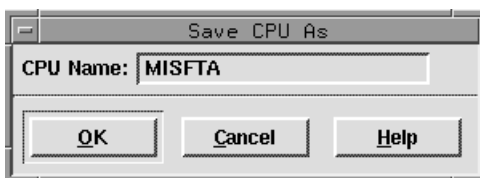
To save the current cpu definition, select Save from the File menu. When the cpu is saved a message appears at the bottom of the window and the window.



If you close the CPU Definition window without saving your changes, a dialog asks if you want to save the changes.

Renaming a CPU Definition

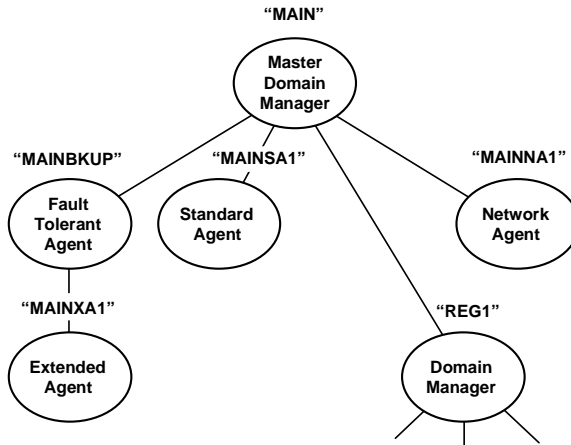
To save a cpu definition with a new name, select Save As... from the File menu. In the Save CPU As dialog, enter the new cpu name. It can contain up to sixteen alphanumeric, dash (-), and underscore (_) characters starting with a letter.



Click OK to save the cpu definition with the name. The CPU Definition window for that cpu name remains open so you can continue working. If the cpu name already exists you are alerted with an overwrite warning. Click Cancel to close the Save CPU As dialog without saving the definition.

Example Cpu Definitions for a Maestro Network

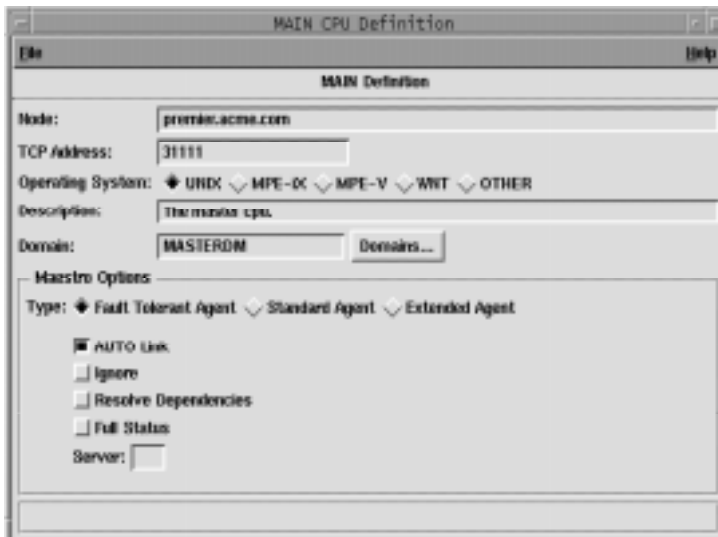
A sample Maestro network is shown below. CPU Definitions for each of the cpu types are also provided.



For more information about Maestro networks and cpu types, see appendix A, *Maestro Networks*.

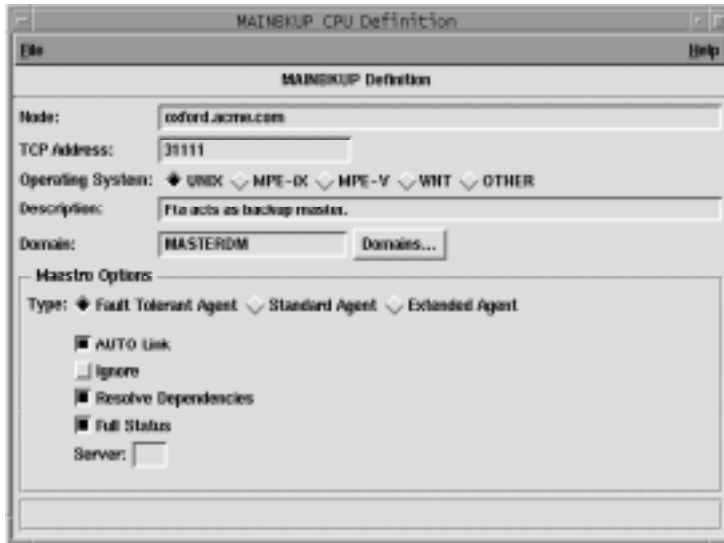
Master Domain Manager

Cpu MAIN is the master domain manager in the network.



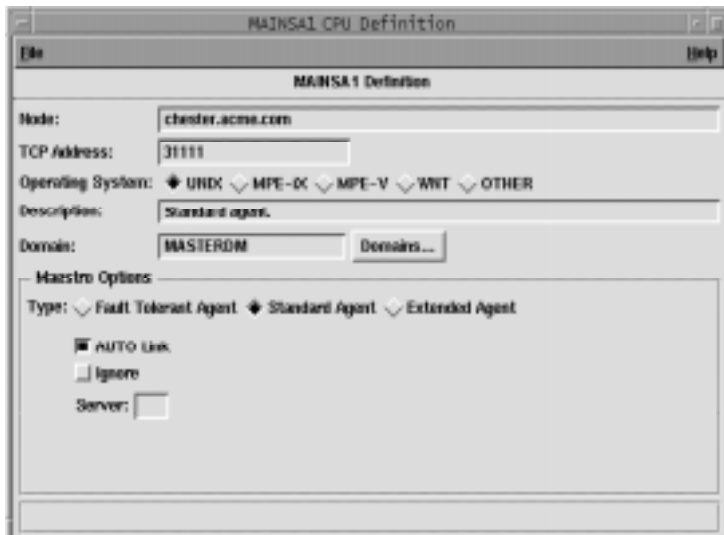
Fault-tolerant Agent and Backup Domain Manager

Cpu MAINBKUP is a fault-tolerant agent. Resolve Dependencies and Full Status are enabled, so MAINBKUP can act as a backup domain manager. For more information, see [Setting Up a Standby Domain Manager](#) on page A-14.



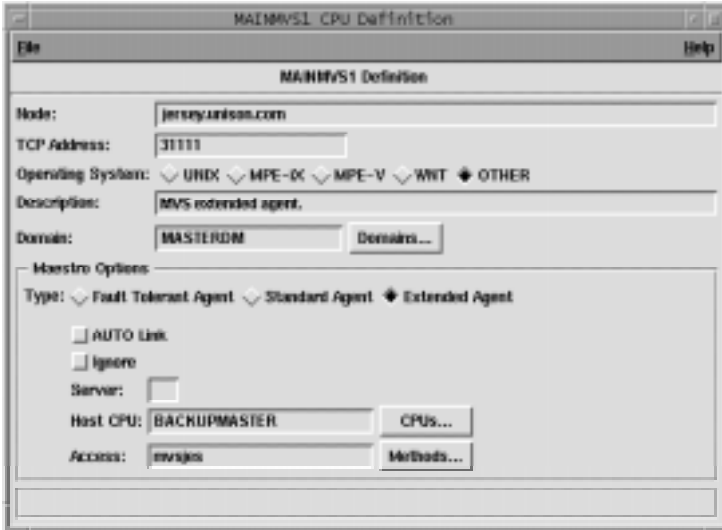
Standard Agent

Cpu MAINSA1 is a standard agent in the master domain (MASTERDM).



Extended Agent

Cpu MAINMVS1 is an extended agent. It's host is the fault-tolerant agent, MAINBKUP. MAINMVS1 uses an access method named `mvsjes`.

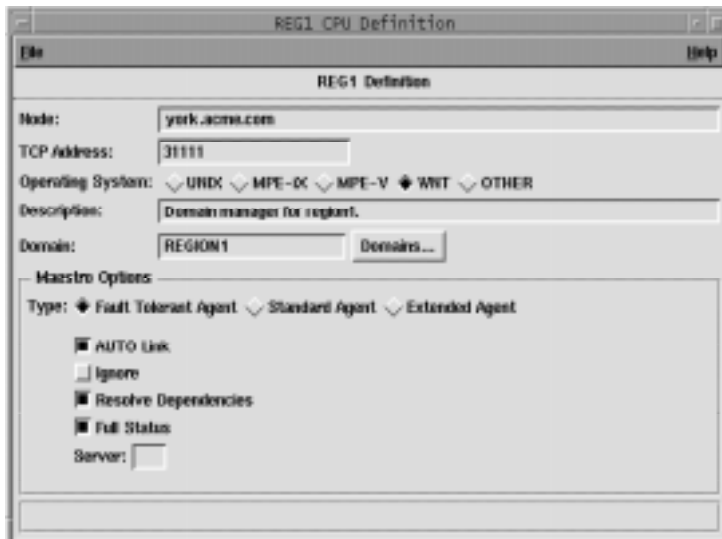


Network Agent

Cpu MAINNA1 is a network agent. It is defined as an extended agent, with a cpu definition similar to MAINMVS1 above. Its host can be any cpu in the same domain, except another extended agent. The access method is `netmth`, supplied with Maestro. For information about defining network agents, see appendix D, [*Internetwork Dependencies*](#).

Domain Manager

Cpu REG1 is the domain manager for the REGION1 domain.

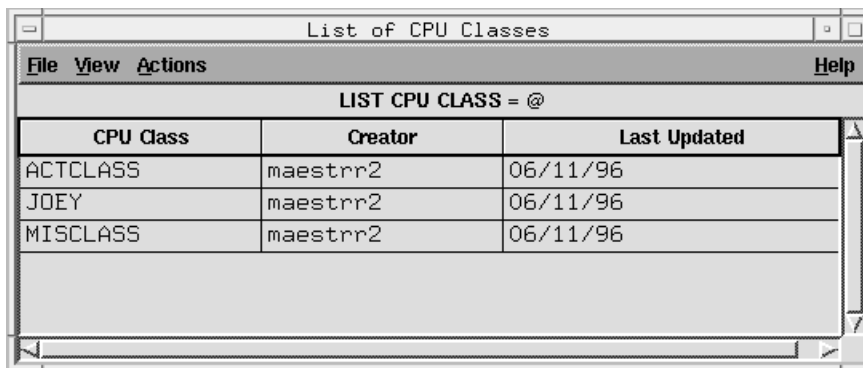


Cpu Classes

Cpu classes permit schedules to be replicated over multiple systems. Cpu class definitions contain a list of cpus. Maestro schedules can be assigned to run on a cpu class, which maps the schedule onto every cpu in the cpu class at production time. Maestro resources and jobs can also be defined for cpu classes. When scheduling a job or resource, the cpu class must match the cpu class of the schedule.

Note: If a job or resource is defined for both a particular cpu and for a cpu class, `compiler` will pick up the cpu information and use it in its processing. An override warning is issued indicating that the cpu job or resource will be used instead of the job or resource defined for the cpu class.

Cpu class definitions are maintained with the Composer program on the master domain manager. Click the CPU Classes button or select CPU Classes... from the Objects menu on the Maestro Composer window to open the List of CPU Classes window. The window displays existing cpu classes and provides menus for editing, creating, and viewing cpu class definitions.



The screenshot shows a window titled "List of CPU Classes" with a menu bar containing "File", "View", "Actions", and "Help". Below the menu bar is a title bar "LIST CPU CLASS = @" and a table with three columns: "CPU Class", "Creator", and "Last Updated". The table contains three rows of data.

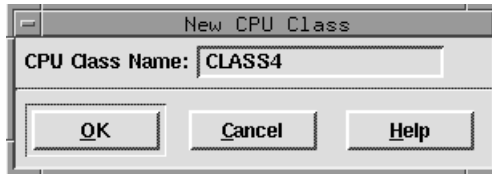
CPU Class	Creator	Last Updated
ACTCLASS	maestrr2	06/11/96
JOEY	maestrr2	06/11/96
MISCLASS	maestrr2	06/11/96

Existing cpu classes are displayed in the following format:

- CPU Class** The name of the cpu class.
- Creator** The login of the creator of the cpu class definition.
- Last Updated** The date the cpu class definition was last edited.

Creating a New Cpu Class Definition

To create a new cpu class definition, select New... from the Actions menu on the List of CPU Classes window. The New CPU Class dialog opens. A New CPU Class dialog can also be opened by selecting New... from the File menu on the CPU Class Definition window.



Enter the name of the cpu class you want to define in the entry box. It can contain up to sixteen alphanumeric, dash (-), and underscore (_) characters starting with a letter. Click OK to open a CPU Class Definition window for the new cpu. All of the fields in the CPU Class Definition are cleared or set to their default values and the cpu class name is displayed in the title bar of the window. If the cpu class name already exists, you are not allowed to continue. Click Cancel to close the dialog without creating a new cpu class. See [CPU Class Definition](#) on page 4-21 for information on completing the definition for the cpu class.

Modifying a Cpu Class Definition

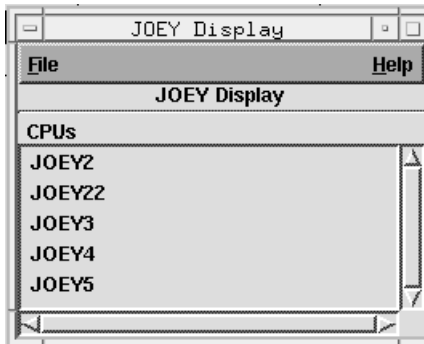
To modify a cpu class definition, select the cpu class in the List of CPU Classes window and then Modify... from the Actions menu. The CPU Class Definition window for the selected cpu class opens permitting you to alter its characteristics. See [CPU Class Definition](#) on page 4-21 for information on modifying the definition for the cpu class.

Deleting a Cpu Class Definition

To delete a cpu class definition, select the cpu class in the List of CPU Classes window and then choose Delete... from the Actions menu. A dialog opens confirming that you want to delete the selected cpu class. Click Yes to delete the cpu class definition from the database, or click No to take no action.

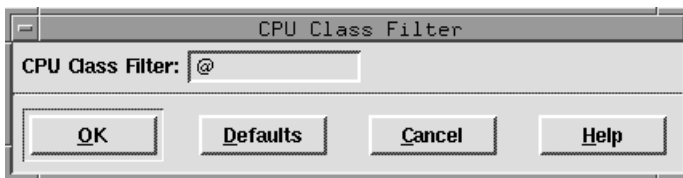
Displaying a Cpu Class Definition

To display a cpu class definition, select the cpu class in the List of CPU Classes window and then Display... from the Actions menu. A display-only window for the selected cpu class opens listing the cpus in the class.



Filtering the List of CPU Classes

To filter the cpu classes displayed in the List of CPU Classes window, select Filter... from the View menu. The CPU Class Filter dialog opens.



In the CPU Class text entry area, enter the string you want Composer to use to filter the list. Wildcards are permitted. Click Default to enter the "@" wildcard in the box, which results in a listing of all cpu classes. Click OK to filter the List of CPU Classes based on the selections and close the dialog. Click Cancel to close the dialog and take no filter action.

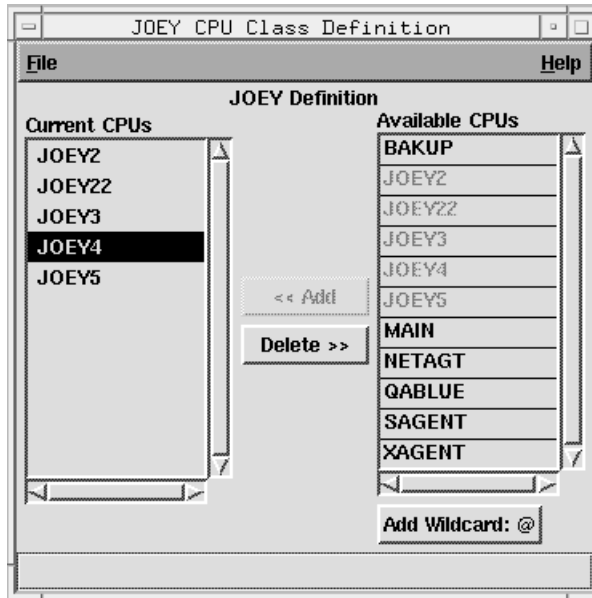
Cpu Class Filter Example

The selections below filter the List of CPU Classes window to display only cpu classes that contain **MIS** in the first three characters of the name:



CPU Class Definition

Use the CPU Class Definition window to define or modify cpu classes. For new cpu classes, the window is displayed with no cpus selected for the cpu class. For existing cpu classes, the current cpus for the class are listed. The window is split into two lists, logically divided by Add and Delete buttons.



The CPU Class Definition window contains two lists:

- Current CPUs** This list contains the cpus selected for the cpu class. A cpu can be listed more than once in a class, either explicitly or by wildcard. Multiple occurrences are reduced to one when the cpu class is used during Maestro production. Other cpu classes are not valid.
- Available CPUs** This list contains existing cpus that can be added to the Current CPUs list. You must have security access to a cpu to include it in a class.

Adding a cpu to the cpu class

Selecting one or more cpus in the Available CPUs list enables the Add button. Click Add to insert selected cpus into the Current CPUs list. Double-clicking on the desired cpu in the Available CPUs list will also add it to Current CPUs.

The @ wildcard, which specifies all defined cpus, can be entered into the list by clicking Add Wildcard:@.

Deleting a cpu from the cpu class

Selecting one or more cpus from the Current CPUs list enables the Delete button. Click Delete to remove selected cpus.

Saving a CPU Class Definition

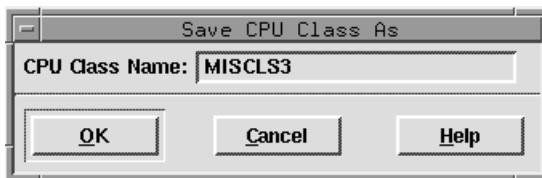
To save the current cpu class definition, select Save from the File menu. When the cpu class is saved a message appears at the bottom of the CPU Class Definition window and the window remains open so you can continue working.



If you close the CPU Class Definition window without saving any changes, a dialog asks if you want to save the changes.

Renaming a CPU Class Definition

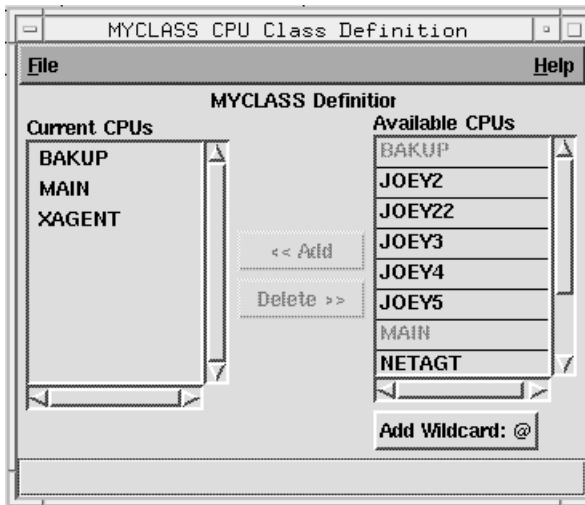
To save a cpu class definition with a new name, select Save As... from the File menu. The Save CPU Class As dialog opens.



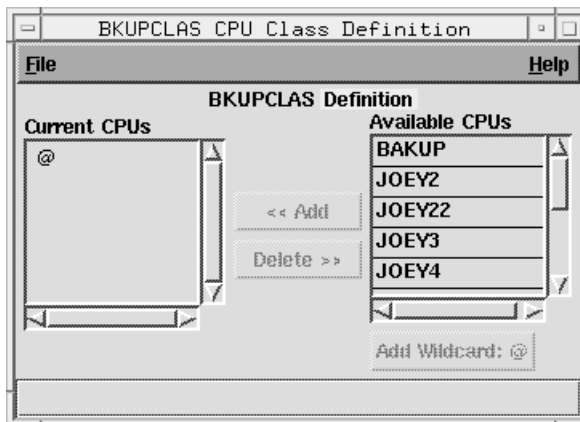
Enter the new cpu class name in the entry box. It can contain up to sixteen alphanumeric, dash (-), and underscore (_) characters starting with a letter in the entry box. Click OK to save the cpu class definition with the name. The CPU Class Definition window for that cpu class name remains open so you can continue working. If the cpu class name already exists you are alerted with an overwrite warning. Click Cancel to close the Save CPU Class As dialog and cancel the save as.

Example Cpu Class Definitions

1. A cpu class named MYCLASS that includes three cpus.



2. A cpu class named BKUPCLAS that includes all cpus.

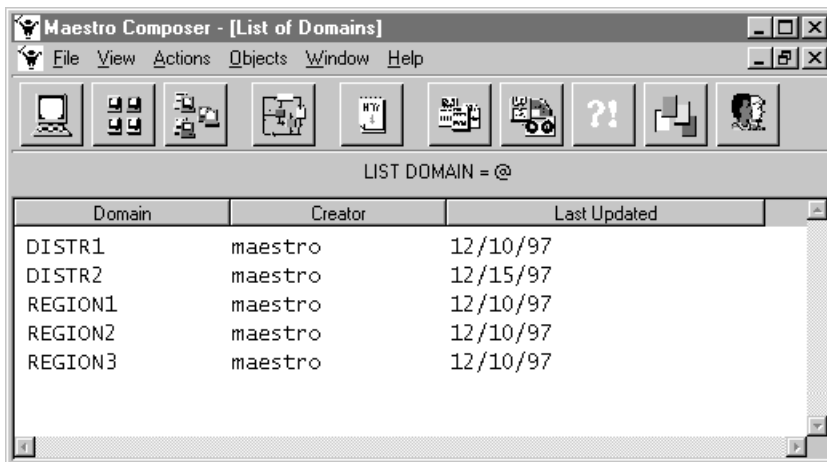


Domains

A Maestro domain is a group of cpus, consisting of one or more agents and a domain manager that acts as the management hub. With the exception of the master domain, all domains have a parent domain.

List of Domains

Domain definitions are maintained with the Composer program on the master domain manager. Click the **Domains** icon or select **Domains...** from the Objects menu on the Maestro Composer window to open the List of Domains window. The window displays existing domains and provides menus for editing, creating, and viewing domain definitions.



Existing domains are displayed in the following format:

- Domain** The name of the domain.
- Creator** The user name of the creator of the domain definition.
- Last Updated** The date the domain definition was last edited.

Displaying a Domain Definition

To display a domain definition, select it in the List of CPU Domains window and then choose **Display...** from the Actions menu. A display-only window for the selected domain is opened. See [DomainDefinition Window](#) on page 4-26 for information about the display.

Creating a New Domain Definition

To create a new domain definition, select **New...** from the Actions menu on the List of Domains window. The New Domain dialog opens. This can also be done by choosing **New...** from the File menu on the Domain Definition window.



Enter the name of the domain in the entry box. It can contain up to sixteen alphanumeric, dash (-), and underscore (_) characters starting with a letter. Click **OK** to open a Domain Definition window. If the domain name already exists, you are not allowed to continue. Click **Cancel** to close the dialog without creating a new domain. See [DomainDefinition Window](#) on page 4-26 for information on completing the domain definition.

Modifying a Domain Definition

To modify a domain definition, select it in the List of Domains window and then choose **Modify...** from the Actions menu. The Domain Definition window opens permitting you to alter its characteristics. See [DomainDefinition Window](#) on page 4-26 for information on modifying the domain definition.

Deleting a Domain Definition

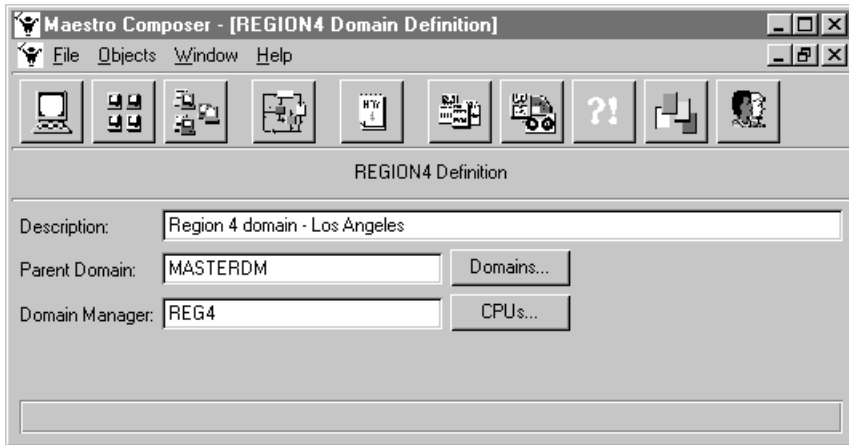
To delete a domain definition, select it in the List of Domains window and then choose **Delete...** from the Actions menu. A dialog opens confirming that you want to delete the domain. Click **Yes** to delete the domain definition from the database, or click **No** to take no action.

Filtering the List of Domains

To filter the domains displayed in the List of Domains window, select **Filter...** from the View menu. The Domain Filter dialog opens. In the Domain field, enter the string you want Composer to use to filter the list. Wildcards are permitted. Click **Default** to enter the "@" wildcard in the box, which results in a listing of all domains. Click **OK** to filter the list and close the dialog. Click **Cancel** to close the dialog and take no action.

DomainDefinition Window

Use the Domain Definition window to add or modify a domain.



The fields in the Domain Definition window are:

- Description** A free-form description of the domain (up to 40 characters).
- Parent Domain** The name of the domain immediately above this domain in the network hierarchy. Click the **Domains...** button for a list of domains.
- Domain Manager** The cpu name of this domain's domain manager. Click the **CPUs...** button for a list of cpus.

If you close the Domain Definition window without saving your changes, a dialog asks if you want to save the changes.

Saving a Domain Definition

To save the current domain definition, select **Save** from the File menu. When the domain is saved, a message appears at the bottom of the Domain Definition window.

Renaming a Domain

To save a domain definition with a new name, select **Save As...** from the File menu. In the Save Domain As dialog, enter a new domain name. It can contain up to sixteen alphanumeric, dash (-), and underscore (_) characters starting with a letter. Click **OK** to save the domain definition with the new

name. If the domain name already exists, you are alerted with an overwrite warning. Click **Cancel** to close the Save Domain As dialog without saving the domain definition.

Jobs

Although the term *job* has various meanings, Maestro imposes a specific definition: a script file or command, whose execution is controlled by Maestro. A job is an important Maestro scheduling object; without jobs, schedules have nothing to execute. A history is kept for every job that is run in a schedule.

Jobs are generally defined independently of schedules. When they are placed in schedules they can have dependencies assigned to them. However, jobs can also be defined in schedules using the command line interface (see *Documenting Jobs in Schedules with auto-doc* starting on page 8-65 for more information).

A qualified Maestro job name contains the Maestro cpu the job is assigned to and a job name; the two strings are delimited by a pound sign (#). For example a job named "job1" assigned to cpu "site1" has the qualified Maestro job name "site1#job1". The naming rules are:

1. Job name strings can contain up to forty alphanumeric, dash(-), and underscore () characters starting with a letter; cpu strings are limited to Maestro-defined cpu names (which have their own restrictions). Names cannot conflict with Composer scheduling language keywords. See *Reserved Words* on page 4-3.
2. The same job script can be documented more than once using different Maestro job names. This permits you to select different login names and recovery options for the same job script, based on the Maestro job name.
3. On each cpu, Maestro job names must be unique, but the same name can be used for jobs that run on different cpus. For example, the following job names are valid, because the jobs are assigned to different cpus:

```
site1#job2  
site2#job2
```
4. Note that the cpu name defines where the job's script file exists and where the job will be launched.
5. In a schedule, job name strings must be unique. For example, the jobs listed in 3 above cannot appear in the same schedule.

Click the Jobs button or select the Jobs... item from the Objects menu to open the List of Jobs window. The window displays existing jobs and provides controls to view, edit, and create job definitions.

The screenshot shows a window titled "List of Jobs" with a menu bar containing "File", "View", "Actions", and "Help". Below the menu bar is a title bar "LIST JOB = @#@". The main content is a table with four columns: "CPU", "Job", "Creator", and "Last Runtime". The table contains eight rows of job definitions, all created by "maestro".

CPU	Job	Creator	Last Runtime
MAIN	APJOB1	maestro	
MAIN	APJOB2	maestro	
MAIN	ARJOB1	maestro	
MAIN	ARJOB2	maestro	
MAIN	GLJOB1	maestro	
MAIN	GLJOB2	maestro	
MAIN	JNEXTDAY	maestro	
MAIN	PAY1	maestro	

Jobs are displayed in the following format:

- CPU** The name of the cpu or cpu class on which the job is defined.
- Job** The name of the job.
- Creator** The login of the creator of the job definition.
- Last Runtime** The date the job was last executed.

Creating a New Job

To create a new job, select New Job... from the Actions menu on the List of CPUs window. The New Job dialog opens. A New Job dialog can also be opened by selecting New... from the File menu on the Job Definition window, by selecting Job... from the Define menu on the Schedule Definition window, or by clicking Define Job... on the Schedule Definition window's Jobs panel.

The screenshot shows a dialog box titled "New Job". It has two input fields: "CPU:" with the value "MAIN" and a "CPUs..." button to its right; and "Job Name:" with the value "APAY1". At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

Enter the name of the cpu or cpu class on which the job runs in the CPU box. The name can be typed in or selected from the list opened by clicking the CPUs... button.

In the Job Name box, enter the name of the job. It can contain up to forty alphanumeric, dash (-), or underscore (_) characters starting with a letter. Click OK to open a Job Definition window for the new job. The fields in the Job Definition window are cleared or set to their defaults and the job name is displayed in the title bar of the window. If the job name already exists, no Job Definition window opens and you get an error. Click Cancel to close the dialog without creating a new job. See [Defining Jobs](#) on page 4-32 for information on completing the definition for the job.

Creating a New MPE Job

To create a new MPE job, select New MPE Job... from the Actions menu on the List of Jobs window. The New MPE Job dialog opens: it functions the same as the New Job dialog described above. A New MPE Job dialog can also be opened by selecting New... from the File menu on the MPE Job Definition window, by selecting MPE Job... from the Define menu on the Schedule Definition window, or by clicking Define MPE Job... on the Schedule Definition window's Jobs panel.

Modifying a Job

To modify a job definition, select the job in the List of Jobs window and then Modify... from the Actions menu. The Job Definition window for the selected job opens permitting you to alter its characteristics. See [Defining Jobs](#) on page 4-32 for information on modifying the definition for the job.

Displaying a Job

To display a job definition, select the job in the List of Jobs window and then Display... from the Actions menu. A display-only window for the selected job opens showing the job definition. See [Defining Jobs](#) on page 4-32 for information on the definition of the job.

Deleting a Job

To delete a job, select the job in the List of Jobs window and then Delete... from the Actions menu. A dialog opens confirming that you want to delete the selected job. Click Yes to delete the job from the database, or click No to take no action.

Filtering the List of Jobs

To filter the List of Jobs window, select Filter... from the View menu. The Job Filter dialog opens.



In the CPU and Job text entry areas, enter the strings you want to use to filter the list. Wildcards are permitted. Click Default to enter the "@" wildcard in the boxes, which results in a listing of all jobs. Click OK to filter the List of Jobs based on the selections. Click Cancel to take no filter action.

Job Filter Examples

1. The selections below filter the List of Jobs window to display only jobs that contain `SITE1` in the beginning the CPU string:

CPU: `SITE1@` Job: `@`

2. The selections below filter the List of Jobs window to display all jobs that contain `ACCT` in the beginning of the Job string:

CPU: `@` Job: `ACCT@`

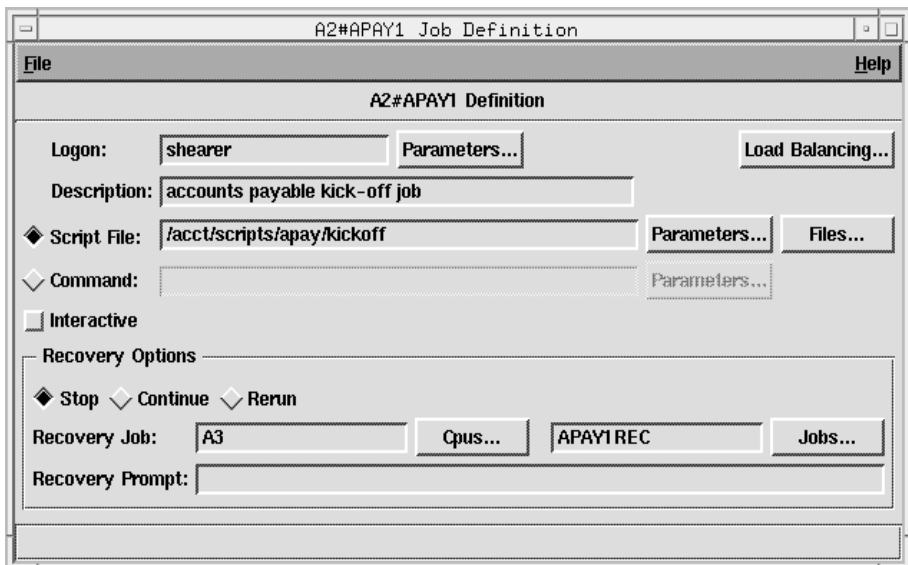
3. The selections below filter the List of Jobs window to display all jobs that contain `SITE` at the beginning of the CPU string and `ACCT` in the beginning of the Job string:

CPU: `SITE@` Job: `ACCT@`

Defining Jobs

The Job Definition window is used to define and edit jobs for execution in schedules. For new jobs, the window is opened with the fields cleared. For existing jobs, the fields contain the existing job information. The Job Display window is identical to the Job Definition window except that it cannot be edited. The MPE Job Definition window is very similar to the Job Definition window, any differences are described on [page 4-38](#).

The Job Definition window is split into two panels, the top definition panel and the lower options panel.



Definition Panel

The Logon and either the Script File or Command (but not both) entry fields are mandatory for defining a Maestro job.

Logon

The user name under which the job will run. It can contain up to 47 characters. If the name contains special characters it must be enclosed in quotes ("). This user must be able to log in on the computer on which the job will run.

For Windows NT jobs, the user must have a Maestro user definition. See [User Definitions](#) on page 4-54 for user requirements. Windows NT user names are in the form:

[domain\]username

	Parameters are permitted– see Using Parameters in Job Definitions below.
Description	Enter a free-form job description (up to 64 characters).
Script File	For an executable file, click this button and enter the path name and any options and arguments (up to 4095 characters). Click the Files... button to display a file selection dialog. Such a selection replaces any entry in the Script File field. Parameters are permitted– see Using Parameters in Job Definitions below. For Windows NT jobs, include the file extension (for example: .exe, .bat). UNC names are permitted. Do not use mapped drives. If the path name contains spaces, enter the name in another file and use that file's path name here.
Command	For a command, click this button and enter a valid command and any options and arguments (up to 4095 characters). A command is executed directly and, unlike Script File, the configuration script (<code>jobmanrc</code>) is not executed. Otherwise, the command is treated as a job, and all job rules apply. Parameters are permitted– see Using Parameters in Job Definitions below.
interactive	For Windows NT jobs, select this option to indicate that the job runs interactively on the Winfows NT desktop.
Load Balancing...	Click the button to open the Load Balancing dialog box. This option is only valid if you have Tivoli's Load Balancer application installed on the master domain manager, and you have Load Balancer configured as a Maestro extended agent. For more information on using Load Balancer with Maestro see the supplemental manual, <i>Extended Agent for Tivoli Load Balancer</i> .

Using Parameters in Job Definitions

Parameters have the following uses and limitations in job definitions:

- Parameters are allowed in the Logon, Script File, and Command fields.
- A parameter can be used as the entire string or a part of it.
- Multiple parameters are permitted in a single field.
- Enclose parameters in carets (^).
- Click the Parameters button to display a list of parameters, then select a parameter to have it inserted in the entry field.

For more information see [Parameters](#) on page 4-45.

Options Panel, Specifying Job Recovery

When you document a job, you can define a recovery option—Stop, Continue, or Rerun—and recovery actions—Recovery Job and Recovery Prompt. Recovery is performed automatically by Maestro's Production Control process if the job abends (terminates unsuccessfully—that is, with a non-zero exit code). The options panel has controls and entry fields to select recovery options for the job. The default is Stop with no Recovery Job and no Recovery Prompt.

If the job is a single command, be aware that some commands may return a non-zero exit code if they are successful. To avoid recovery actions in such cases, the Continue option should be specified with no Recovery Job and no Recovery Prompt. See [Recovery Options and Actions Table](#) on page 4-36 for more information on recovery options and their consequences.

Recovery Options Select one of the three options to use if the job abends.

Stop	Do not continue with the next job.
Continue	Continue with the next job.
Rerun	Rerun the job.

Recovery Job Enter the name of a recovery job to run if the original job abends. Recovery jobs are run only once for each abended instance of the original job.

The first box is used to specify the recovery job's cpu if it is different than the original job's cpu. If no cpu is specified, the original job's cpu is assigned. Not all jobs are eligible to have recovery jobs run on a different cpu. Follow these guidelines:

- The original job's and recovery job's cpu must have Maestro 4.5.5 or later installed.
- If either cpu is an extended agent, it must be hosted by a domain manager or a fault-tolerant agent running in Full Status mode.
- The recovery job's cpu must be in the same domain as the original job's cpu.
- If the recovery job's cpu is a fault-tolerant agent, it must be running in Full Status mode.

If you are specifying a recovery cpu, enter its Maestro cpu name, or click the CPUs... button opens and choose a cpu from the list. Note that all cpus in the network are displayed, whether or not they are valid for recovery jobs.

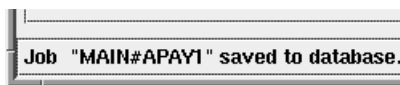
In the second box, enter the name of the recovery job. If you enter a job that is undefined, a warning is issued when you save the parent job. Clicking the Jobs... button opens a dialog from which to choose defined jobs. If a cpu selection has been made in the first box, the list is filtered to display only jobs defined for the selected cpu. If no cpu selection is made, all defined jobs are displayed.

Recovery Prompt Enter a recovery prompt (up to 64 characters) to issue if the job abends. The recovery prompt is a local prompt, responded to via the Conman program.

If the text begins with a colon (:), the prompt is issued, but no reply is necessary to continue processing. If the text begins with an exclamation point (!), the prompt is not issued, but a reply is necessary to proceed. Otherwise, the prompt is issued and a reply is necessary to continue. You can include backslash "n" (\n) within the text to cause a new line.

Saving a Job

To save the current job, select Save from the File menu. When the job is saved a message appears at the bottom of the window and the window remains open so you can continue working.



If you close the Job Definition window without saving any changes, a dialog asks if you want to save the changes.

Renaming a Job

To save the current job definition with a new name, select Save As... from the File menu. The Save Job As dialog opens. Enter a new CPU or Job Name (or both). Click OK to save the job definition. The Job Definition window for the new job name remains open so you can continue working. If the job name

already exists you are alerted with an overwrite warning. Click Cancel to close the dialog and cancel the save as.

Recovery Options and Actions Table

Combinations of recovery options and actions are summarized in the table below. The table is based on the following criteria from a schedule called sked1:

1. Schedule sked1 has two jobs, job1 and job2.
2. If selected for job1, the Recovery Job is jobr.
3. job2 is dependent on job1 and will not execute until job1 is complete.

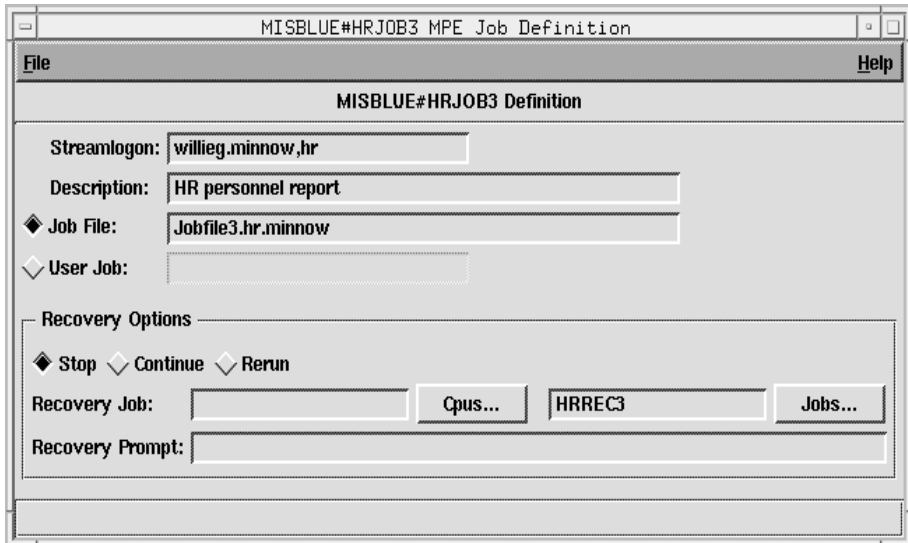
		Stop	Continue	Rerun
Recovery Prompt	no	Intervention required.	Launch job2.	Rerun job1. If job1 abends, issue Maestro prompt. If “yes” reply, repeat above. If job1 is successful, launch job2.
Recovery Job	no			
Recovery Prompt	yes	Issue recovery prompt. Intervention required.	Issue recovery prompt. If “yes” reply, launch job2.	Issue recovery prompt. If “yes” reply, rerun job1. If job1 abends, repeat above. If job1 is successful, launch job2.
Recovery Job	no			
Recovery Prompt	no	Launch jobr. If it abends, intervention required, or if successful, launch job2.	Launch jobr. Launch job2.	Launch jobr. If jobr abends, intervention is required. If jobr is successful, rerun job1. If job1 abends, issue Maestro prompt. If “yes” reply, repeat above. If job1 is successful, launch job2.
Recovery Job	yes			
Recovery Prompt	yes	Issue recovery prompt. If “yes” reply, launch jobr. If it abends, intervention required, or if successful, launch job2.	Issue recovery prompt. If “yes” reply, launch jobr. Launch job2.	Issue recovery prompt. If “yes” reply, launch jobr. If jobr abends, intervention is required. If jobr is successful, rerun job1. If job1 abends, repeat above. If job1 is successful, launch job2.
Recovery Job	yes			

Notes:

- "Intervention required" means that `job2` is not released from its dependency on `job1`, and therefore must be released by the operator.
- The Continue recovery option overrides the abend state, which may cause the schedule containing the abended job to be marked as successful. This will prevent the schedule from being carried forward to the next day.
- If you select the Rerun option without supplying a recovery prompt, Maestro will generate its own prompt.
- When a recovery job runs, it appears in Conman displays with the notation `>>recovery`. To reference the recovery job in a Conman command, you must use the name of the original job (`job1`, in the above scenario, not `jobr`). Recovery jobs are run only once per abend.

MPE Job Definition

The MPE Job Definition window is used to define and edit MPE jobs for execution on HP 3000 computers. The MPE Job Definition window is very similar to the Job Definition window (described in *Defining Jobs* on page 4-32). There are some differences in the Definition panel, which are described below.



Definition Panel

The Streamlogon and either the Job File or User Job (but not both) entry fields are mandatory for defining an MPE job. Parameters are invalid for MPE job definitions. For more information on MPE jobs and Maestro, refer to the *Maestro for MPE User Guide*.

Streamlogon Enter the logon under which the job will run in this format:

user.account[,group]

Description Enter a free-form description of the job (up to 64 characters).

Job File Click the radio button and enter a fully-qualified job file name in the format:

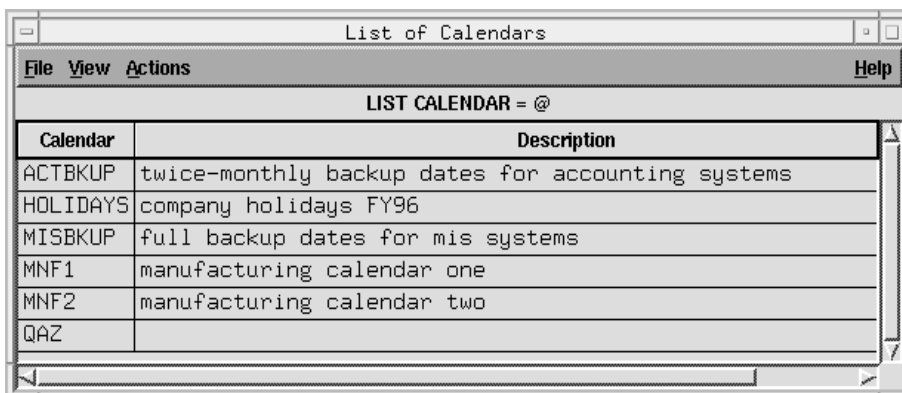
filename.group.account

User Job Click the radio button and enter the job name from the job's JOB card.

Calendars

Calendars are used to assign execution dates to schedules. You can create as many calendars as you wish to handle your scheduling needs. For example, calendars can contain lists of payday, month-end dates, or company holidays. Calendars provide an easy way to include and exclude dates for a schedule. Calendars can also be used in **datecalc** expressions (see [datecalc](#) on page 10-8 for more information).

Click the Calendars button or select the Calendars... item from the Objects menu on the Maestro Composer main window to open the List of Calendars window. This shows existing calendars and includes menus for editing, creating, and viewing calendars.



The screenshot shows a window titled "List of Calendars" with a menu bar containing "File", "View", "Actions", and "Help". Below the menu bar is a header "LIST CALENDAR = @" and a table with two columns: "Calendar" and "Description". The table contains the following data:

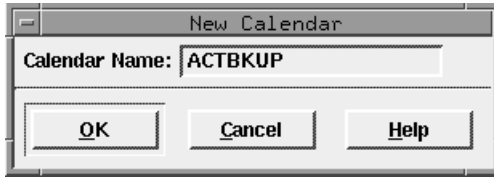
Calendar	Description
ACTBKUP	twice-monthly backup dates for accounting systems
HOLIDAYS	company holidays FY96
MISBKUP	full backup dates for mis systems
MNF1	manufacturing calendar one
MNF2	manufacturing calendar two
QAZ	

Calendars are displayed in the following format:

Calendar The name of the calendar.
Description A free-form description of the calendar.

Creating a New Calendar

To create a new calendar, select New... from the Actions menu on the List of Calendars window. The New Calendar dialog opens. A New Calendar dialog can also be opened by selecting New... from the File menu on the Calendar Definition window or by selecting Calendars... from the Define menu on the Schedule Definition window.



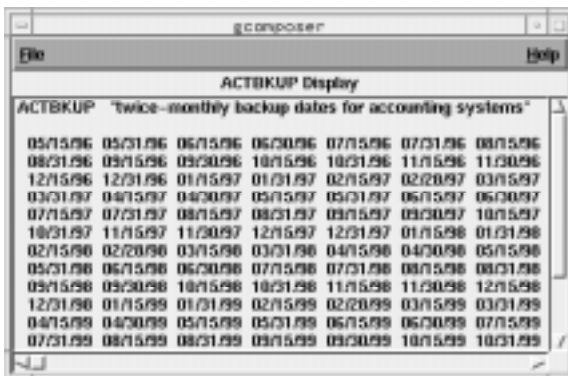
Enter the name of the calendar to be defined in the Calendar Name box. It can contain up to eight alphanumeric, dash (-), and underscore () characters starting with a letter. Click OK to open a Calendar Definition window for the new calendar. The Calendar Definition window has no dates selected and the calendar name is displayed in the title bar of the window. If the calendar name already exists, no Calendar Definition window opens and you get an error. Click Cancel to close the dialog without creating a new calendar. See [Calendar Definition](#) on page 4-42 for more information on completing the definition for the calendar.

Modifying a Calendar

To modify a calendar, select the calendar in the List of Calendars window and then Modify... from the Actions menu. The Calendar Definition window for the selected calendar opens permitting you to alter its characteristics. See [Calendar Definition](#) on page 4-42 for more information on modifying the definition of the calendar.

Displaying a Calendar

To display a calendar, select the calendar in the List of Calendars window and then Display... from the Actions menu. The Calendar Display window opens listing the dates in the selected calendar.

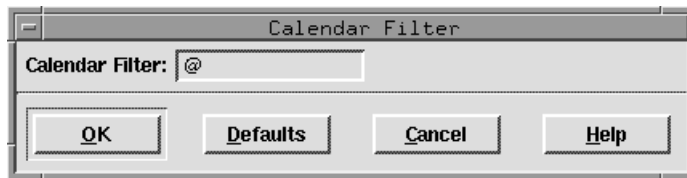


Deleting a Calendar

To delete a calendar, select the calendar in the List of Calendars window and then Delete... from the Actions menu. A dialog opens confirming you want to delete the selected calendar. Click Yes to delete the calendar, or click No to take no action.

Filtering the List of Calendars

To filter the List of Calendars window, select Filter... from the View menu. The Calendar Filter dialog opens. In the Calendar Filter field, enter the string you want to use to filter the list. Wildcards are permitted. Click Default to enter the "@" wildcard, which results in a listing of all calendars. Click OK to filter the List of Calendars, or click Cancel to take no filter action.



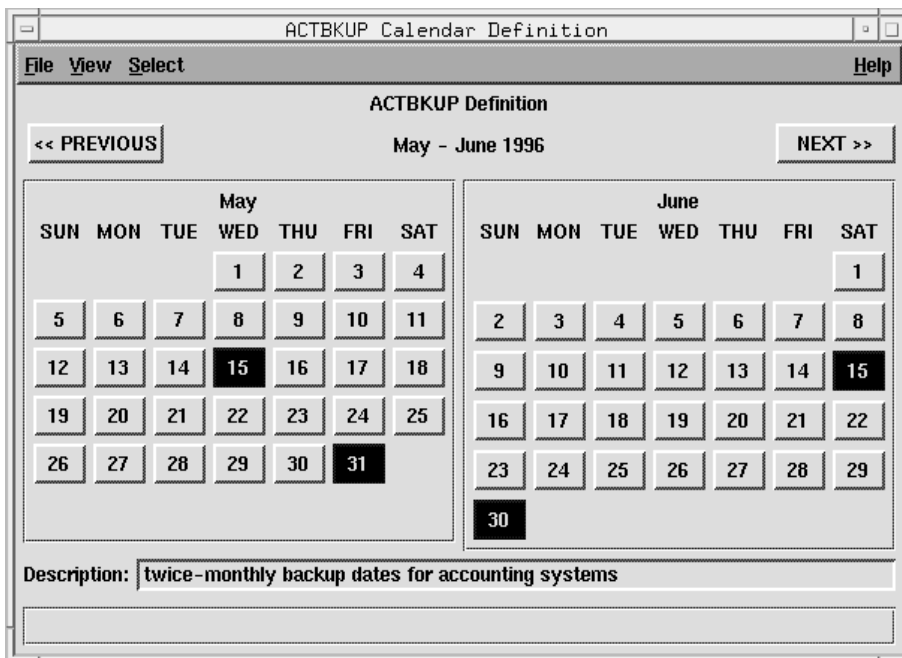
Calendar Filter Example

The selection below filters the List of Calendars to display only calendars that start with the string EOM:



Calendar Definition

The Calendar Definition window is used to define and modify calendars. The edit panel has navigation tools— Previous and Next— and displays a two-month calendar. You can enter a description of the calendar (up to 64 characters) in the Description box.



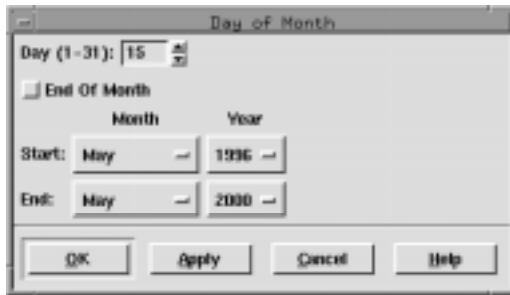
Selecting Dates for the Calendar

To select a day in a month, click on the corresponding numbered button. When the day is selected, the box changes color. To specify the color see [Choosing the Date Selection Color](#) on page 4-44.

Click the PREVIOUS button to move two months backward in the calendar, or click NEXT to move two months forward. From the current year, you can go back to January of the previous year and forward to December, 13 years in the future.

Selecting Days of the Month

To choose a day of every month in a specified range for the current calendar, select Day of Month... from the Select menu. The Day of Month dialog opens.



Enter a day of the month in the Day (1-31) box. The up/down arrows change the value in one day units. If a day is chosen that does not exist for some months in the range, no day is highlighted in those months.

Selecting End of Month chooses the last day of each month and overrides any entry in the Day box.

To set the beginning of a range, select a Month and Year adjacent to Start. Press the button in either of the boxes below Month and Year and a pop-up menu appears. The range for months is January through December, the range for years is the current year through nine years forward. Drag the mouse over a month or year to select it.

Using the buttons next to End, set the end of a range in the same manner as you did for the beginning.

Click OK or Apply to assign the selections to the calendar (OK also closes the dialog). The day of the month is highlighted in each month in the selected range in the Calendar Definition window. Click Cancel to close the dialog without making any day-of-month selection.

Saving a Calendar

To save the current calendar, select Save from the File menu. When the calendar is saved a message appears at the bottom of the Calendar Definition window and the window remains open so you can continue working.

If you close the Calendar Definition window without saving any changes, a dialog asks if you want to save the changes.

Renaming a Calendar

To save the current calendar definition with a new name, select **Save As...** from the **File** menu. The **Save Calendar As** dialog opens. Enter a new **Calendar Name**. Click **OK** to save the calendar. The **Calendar Definition** window for the new calendar name remains open so you can continue working. If the **Calendar Name** already exists you are alerted with an overwrite warning. Click **Cancel** to close the dialog and cancel the save as.

Choosing the Date Selection Color

To set a different color to highlight selection dates in the **Calendar Definition** window, select **Color...** from the **View** menu. The **Color Chooser** dialog opens. This is the standard **Color Chooser** for the system.

Defining a Holidays Calendar

When assigning execution dates to schedules via the **On/Except Panel** of the **Schedule Definition** window, there is an option button called **Workdays**. For this option to work properly, a `holidays` calendar must exist. The **Workdays** option selects every day, excluding **Saturday**, **Sunday**, and the dates in the `holidays` calendar. To create the `holidays` calendar, simply define a calendar named `holidays` with the desired dates selected.

Using Calendars in Schedules

Refer to *Selecting Schedule Execution Calendars* on page 5-11.

Parameters

Parameters are useful to substitute values into your schedules and job scripts. Since parameters are maintained centrally, all schedules and jobs are updated automatically when the value changes. For scheduling, a parameter can be used as a substitute for all or part of:

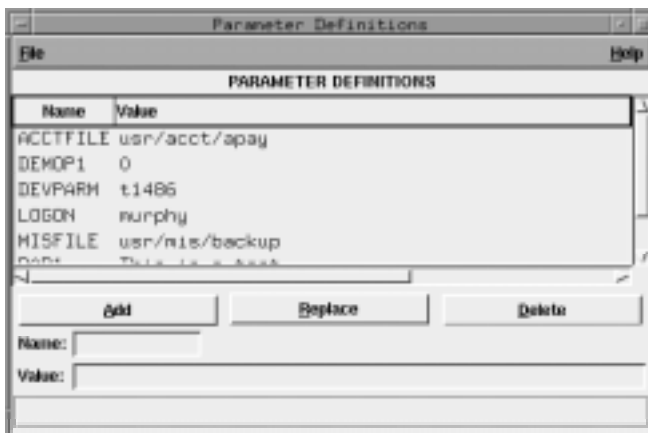
- File dependency path names.
- Text for local and global prompts.
- Logon, Command, and Script File names for Maestro jobs.

In these cases, the parameter must be defined on the master domain manager and its values is expanded during Maestro's pre-production processing. See [Using Parameters in Schedules, Jobs and Prompts](#) on page 4-47 for more information.

When using a parameter in a job script, the value is expanded at run time. In this case, the parameter must be defined on the cpu where it will be used. Parameters cannot be used for extended agent job scripting. See [Using Parameters in a Job Script: parms Command](#) on page 4-47 for more information.

Defining Parameters

Click the Parameters button or select the Parameters... item from the Objects menu on the Maestro Composer main window to open the Parameter Definitions window. The top of the window lists current parameters. Use the bottom part to add, modify, and delete parameter definitions.



Parameters are displayed in the following format:

Name The name of the parameter.
Value The value of the parameter.

The Parameter Definitions window can also be opened by selecting Parameters... from the Define menu on the Schedule Definition window.

Creating a New Parameter

To create a new parameter, type the name of the parameter in the Name box. It can contain up to eight alphanumeric characters, starting with a letter. Enter the parameter's value (up to 72 characters) in the Value box. The parameter Value can include backslash "n" (\n) to cause a new line. The Value cannot be another parameter. Click Add to add the parameter to the display list. If you enter a Name that already exists, the Add button is not enabled and the existing parameter is highlighted in the display list.

Modifying a Parameter

To modify a parameter, either select it in the display list or type its name into the Name box. Edit the text in the Value box, and click the Replace button.

Renaming a Parameter

To rename a parameter, select it in the display list, modify the Name, and click Add. If you change the Name to an existing parameter name, the Add button is not enabled and the existing parameter is highlighted in the list.

Deleting a Parameter

To delete a parameter, either select it in the display list or type its name in the Name box, and then click Delete.

Saving Parameters

To save the current list of parameters, select Save from the File menu. A message at the bottom of the window indicates completion.

Using Parameters in a Job Script: *parms* Command

Maestro parameters, among other things, are intended for use in job scripts. They are local to each cpu, and, unlike shell variables, they do not need to be exported. Parameters are accessible to all Maestro-launched jobs via the **parms** command. The following example illustrates the use of two parameters in a job script. Parameters used in job scripts are expanded at run time. Two parameters, *arpath* and *listdev*, are defined as:

Name	Value
ARPATH	/usr/lib/arfiles
LISTDEV	lj2

The parameters are then used in a job script as follows:

```
...
cd `parms arpath`
printer=`parms listdev`;export printer
lp -d$printer ardata
...
```

For a complete description of the *parms* command, see [parms](#) on page 10-25.

Using Parameters in Schedules, Jobs and Prompts

See the following references about how to use parameters in Maestro scheduling objects:

- For jobs see [Defining Jobs](#) starting on page 4-32.
- For Opens Files dependencies in schedules and jobs, see [Opens Files Panel: Selecting Schedule File Dependencies](#) starting on page 5-21 and [Opens Files Panel: Selecting Job File Dependencies](#) starting on page 5-41, respectively.
- For Global Prompt dependencies, see [Global Prompts](#) starting on page 4-48.
- For Local Prompt dependencies in jobs and schedules, see [Defining and Adding a Local Prompt to a Schedule](#) on page 5-20 and [Defining and Adding a Local Prompt to a Schedule](#) on page 5-20, respectively.

Parameters are accessible to all Maestro jobs, schedules, and prompts. Parameters used in scheduling are expanded during Maestro pre-production processing. If the value of a parameter is altered, the job definition will automatically take on the new value when pre-production processing occurs.

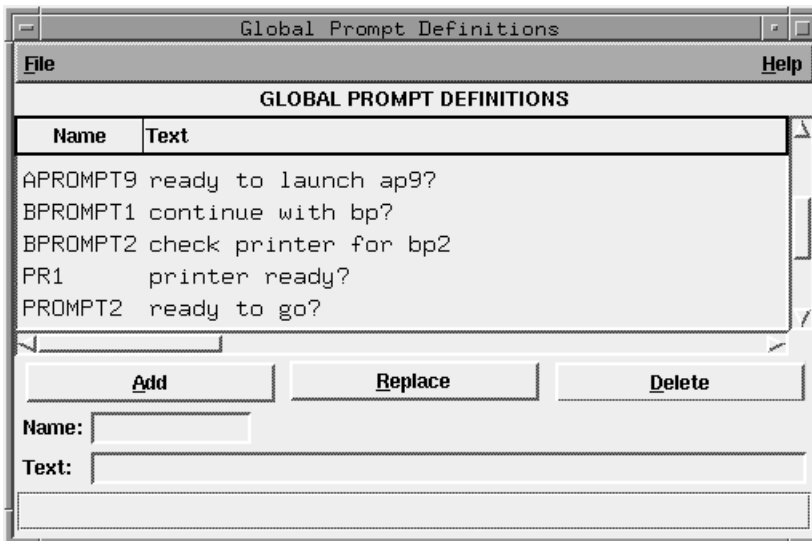
Global Prompts

Prompts can be used as dependencies for jobs and schedules. Prompts must be answered to affirmatively (via Conman) for dependent jobs and schedules to launch. For example, you can issue a prompt to make sure that a printer is on-line before a job that prints a report is run.

There are two kinds of prompts: global and local. Global prompts can be replied to by name. When a global prompt is issued, only one reply is necessary to release all jobs and schedules that use it as dependency. Local prompts, on the other hand, are linked to an individual job or schedule and are created as dependencies within individual schedule definitions. For more information on local prompts see *Defining and Adding a Local Prompt to a Schedule* on page 5-20 and *Defining and Adding a Local Prompt to a Schedule* on page 5-20.

Defining Global Prompts

Global prompts are maintained on the master domain manager with the Composer program. Click the Prompts button or select the Prompts... item from the Objects menu on the Maestro Composer main window to open the Global Prompt Definitions window. The top part of the window displays existing global prompts and the bottom part provides tools to add, modify, and delete prompt definitions.



Global prompts are displayed in the following format:

Name The name of the prompt.
Text The text of the prompt.

The Global Prompt Definitions window can also be opened by selecting Prompts... from the Define menu on the Schedule Definition window.

Creating a New Global Prompt

To create a new prompt, type the name of the prompt in the Name box. It can contain up to eight alphanumeric, dash (-), and underscore () characters starting with a letter. Enter the prompt text (up to 200 characters) in the Text box. If the text begins with a colon (:), the prompt is issued, but no reply is necessary to continue processing. If the text begins with an exclamation point (!), the prompt is not issued, but a reply is necessary to proceed. Otherwise, the prompt is issued and a reply is necessary to continue. You can include backslash "n" (\n) within the text to cause a new line.

Note: Maestro parameters can be used in the text string of the prompt. Multiple parameters are permitted. The parameter name must be enclosed in carets (^). For more information on parameters see [*Parameters*](#) on page 4-45.

Click Add to add the prompt to the display list. If you enter a Name that already exists, the Add button is not enabled and the existing prompt is highlighted in the display list.

Modifying a Global Prompt

To modify a prompt, either select the prompt in the display list or type its name into the Name box. Edit the text in the Text box, if any, and click the Replace button to replace the old prompt Text with the new Text.

Renaming a Global Prompt

To rename a prompt, select the prompt in the display list, modify the prompt Name, and click Add to add it to the display list. If you change the Name to an existing prompt name, the Add button is not enabled and the existing prompt is highlighted in the list.

Deleting a Global Prompt

To delete a prompt, either select the prompt in the display list or type its name into the Name box. Click Delete to remove the prompt from the display list.

Saving Global Prompts

To save the prompts in the current display list, select Save from the File menu. When the list is saved a message appears at the bottom of the Global Prompt Definitions window.

Using Global Prompts in Schedules and Jobs

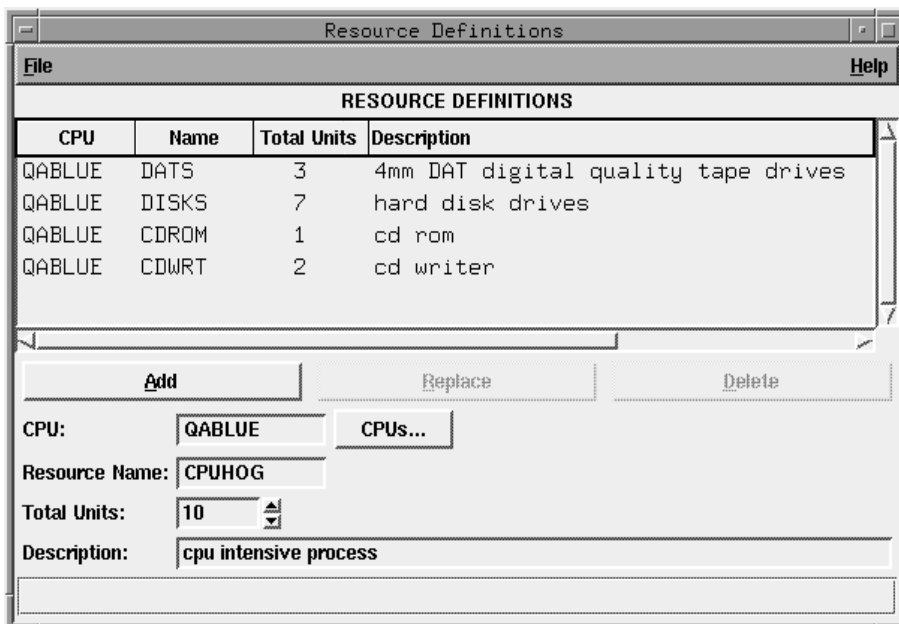
Refer to *Prompts Panel: Selecting Prompts for a Schedule* on page 5-19 and *Prompts Panel: Selecting Prompts for a Job* on page 5-39.

Resources

Maestro resources can represent either physical or logical resources on your system. Once defined, they can be used as dependencies for jobs and schedules that run on the cpu or cpu class for which the resource is defined.

Defining Resources

Resources are maintained on the master domain manager with the Composer program. Click the Resources button or select the Resources... item from the Objects menu on the Maestro Composer main window to open the Resource Definitions window. The top part of the window displays existing resources and the bottom part provides tools to add, modify, and delete resource definitions.



Resources are displayed in the following format:

CPU	The name of the cpu or cpu class to which the resource belongs.
Name	The name of the resource.
Total Units	The total available resource units.
Description	A free-form description of the resource.

The Resource Definitions window can also be opened by selecting Resources... from the Define menu on the Schedule Definition window.

Creating a New Resource

To create a new resource, enter the name of the Maestro cpu or cpu class on which the resource is to be defined in the CPU box. The cpu can be typed or selected from the list that is opened when you click the CPUs... button. In the Resource Name box, enter the name of the resource. It can contain up to eight alphanumeric, dash (-), and underscore (_) characters starting with a letter. The Resource Name must be unique for each cpu.

In the Total Units box, enter the number (0-1024) of total available resource units. The up/down arrows change the unit value by one. If no number is entered, a value of zero is assigned when the resource is added to list. This number can be modified during production (see [*Changing Available Units of a Resource*](#) on page 6-57).

Optionally, in the Description box, enter a free-form description of the resource (up to 64 characters).

Click Add to add the resource to the display list. If you enter a resource name that already exists, the Add button is not enabled and the existing resource is highlighted in the display list.

Changing the Total Units of a Resource

To modify a resource, either select the resource in the display list or enter its CPU and Resource Name into the appropriate boxes. Change the value, in the Total Units box and click the Replace button.

Renaming a Resource

To rename a resource, select the resource in the display list, modify the CPU or Resource Name (or both) and click Add to add it to the display list. If you change the CPU and Resource Name to an existing resource name, the Add button is not enabled and the existing resource is highlighted in the list.

Deleting a Resource

To delete a resource, either select the resource in the display list or type its CPU and Resource Name into the appropriate boxes. Click Delete to remove the resource from the display list.

Saving Resources

To save the resources in the current display list, select Save from the File menu. When the list is saved a message appears at the bottom of the Resource Definitions window.

Using Resources in Schedules and Jobs

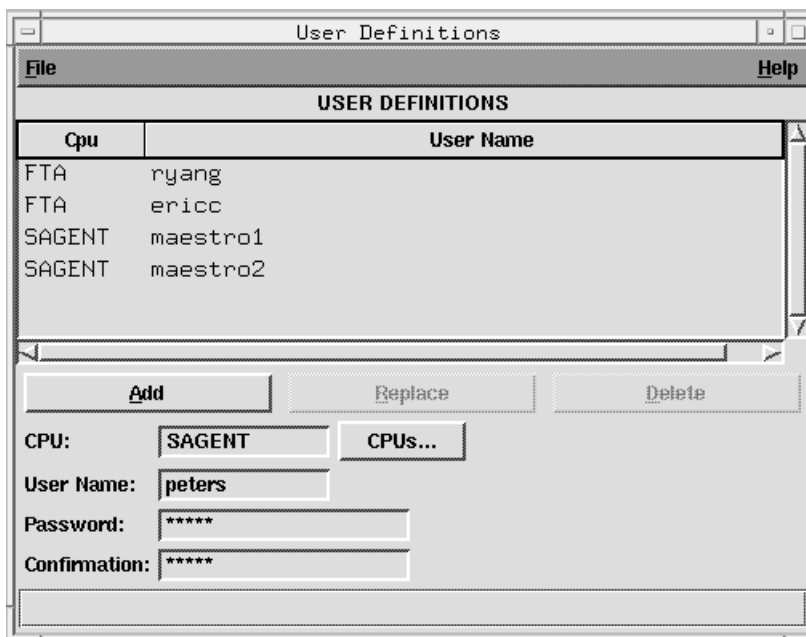
Refer to *Needs Resources Panel: Assigning Resources to a Schedule* on page 5-25 and *Needs Resources Panel: Assigning Resources to a Job* on page 5-44.

User Definitions

For Windows NT jobs only, the user name specified in the job Logon field must have a matching user definition. The definitions furnish the user passwords required by Maestro to launch jobs.

Defining Users

To define users, click the Users button or select the Users... item from the Objects menu on the Maestro Composer main window to open the User Definitions window. The window can also be opened by selecting Users... from the Define menu on the Schedule Definition window. The top part of the window displays existing Users. Use the bottom part to add, modify, and delete users.



User information is entered as follows:

Cpu The Maestro cpu on which the user is allowed to launch jobs. The pound sign is a required delimiter. If no cpu is specified the cpu on which **gcomposer** is running is used.

- User Name** The user name (up to eight alphanumeric, dash, and underscore characters starting with a letter). Note that user names are case-sensitive. The user must be able to log on to the computer on which it will have jobs launched by Maestro, and must have the right to “Log on as batch.”
- If the name is not unique in Windows NT, it is considered to be a local user, a domain user, or a trusted domain user, in that order.
- Password** The user’s password. Once a user definition is compiled, there is no way to read the password. Users with appropriate security privileges can modify or delete a user, but password information is never displayed. Leave this field blank if there is no password.
- Confirmation** Re-enter the user’s password for confirmation.

Creating a New User

To create a new User, enter the name of the Maestro cpu on which the User is to be defined in the CPU box. The cpu can be typed or selected from the list that is opened when you click the CPUs... button. In the User Name box, enter the account name of the user. If you need to include the domain name for a user, define the user with the command line **composer** program— refer to *User Definitions* on page 8-43.

In the Password box, enter the User’s password. Re-enter the password in the Confirmation box. For security purposes, the password is never displayed. You can leave the Password field blank if a password is not required.

Click Add to add the user to the display list. If you entered a user name that already exists, the Add button is not enabled and the existing user is highlighted in the display list.

NT Trusted Domain User

If it will be necessary for Maestro to launch jobs for a trusted domain user, special attention must be given to defining user accounts. Assuming Maestro is installed in Domain1 for user account **maestro**, and user account **sue** in Domain2 needs to have jobs launched by Maestro, the following must be true:

- There must be mutual trust between Domain1 and Domain2.
- In Domain1 on the computers where jobs will be 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."

Changing the Password of a User

To change the password of a user, either select the user in the display list or enter its CPU and User Name into the appropriate boxes. Change the password, in both the Password and Confirmation boxes and click the Replace button to replace the old password with the new one. You can leave the Password field blank to indicate no password for the user.

A user's password can also be temporarily changed for a production run. See [Changing a User's Password](#) on page 6-8 for more information.

Renaming a User

To rename a user, select the user in the display list, modify the CPU or User Name (or both) and click Add to add it to the display list. If you change the CPU and User Name to an existing user name, the Add button is not enabled and the existing user is highlighted in the list.

Deleting a User

To delete a user, either select the user in the display list or type its CPU and User Name into the appropriate boxes. Click Delete to remove the user from the display list.

Saving Users

To save the users in the current display list, select Save from the File menu. When the list is saved a message appears at the bottom of the User Definitions window.

5

Composing Schedules

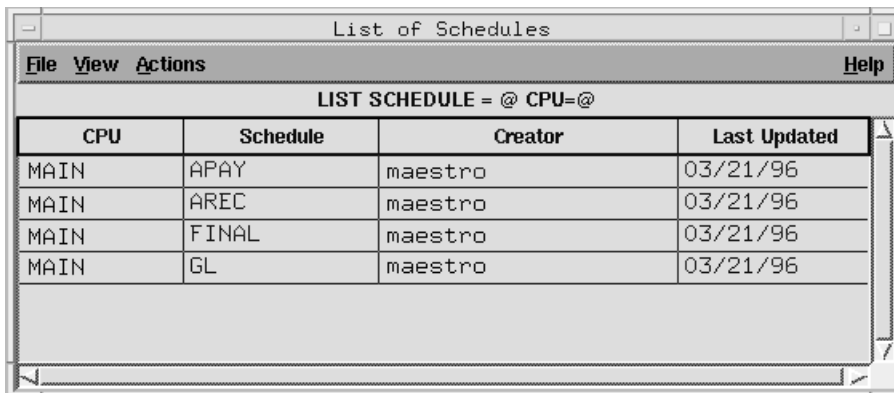
Maestro's primary processing task is executing its schedules. A schedule is an outline of batch processing consisting of a list of jobs. Each schedule is dated so that it can be selected automatically by Maestro for execution.

Schedules can specify that a job must run under a variety of events and conditions. For example, a schedule can specify that its jobs will not run until another schedule has completed execution of its own jobs, a certain file is accessible, a certain resource is available, or a prompt has been replied to. Schedule stipulations like these are called *dependencies*.

A dependency is a condition that must be satisfied before a job or schedule will be launched. All dependencies for schedules are selected using Composer's Schedule Definition window. This section describes how to create and edit schedules and their dependencies. A single job or schedule is permitted up to 40 dependencies.

Scheduling Basics

Click the Schedules button or select the Schedules... item from the Objects menu on the Maestro Composer main window to open the List of Schedules window. The window displays existing schedules and provides controls to view, edit, and create schedule definitions.



CPU	Schedule	Creator	Last Updated
MAIN	APAY	maestro	03/21/96
MAIN	AREC	maestro	03/21/96
MAIN	FINAL	maestro	03/21/96
MAIN	GL	maestro	03/21/96

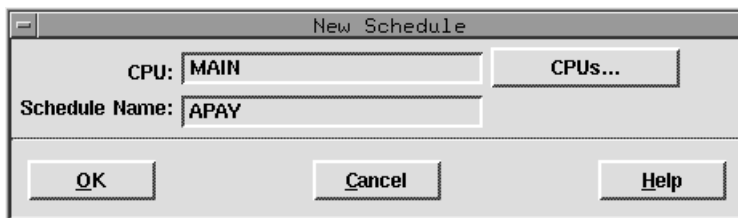
Schedules are displayed in the following format:

- CPU** The name of the cpu on which the schedule is defined.
- Schedule** The name of the schedule.
- Creator** The user name of the creator of the schedule definition.
- Last Updated** The date the schedule was last edited.

Creating a New Schedule

To create a new schedule, select New Schedule... from the Actions menu on the List of Schedules window. The New Schedule dialog opens. A New Schedule dialog can also be opened by selecting New... from the File menu on the Schedule Definition window.

A qualified schedule name comprises the cpu on which the schedule resides and a schedule name.



Enter the name of the cpu or cpu class on which the schedule is to be defined in the CPU box. The name can be typed in or selected from the Available CPUs list produced by clicking the CPUs... button. If no cpu or cpu class is selected, the cpu running Composer is used.

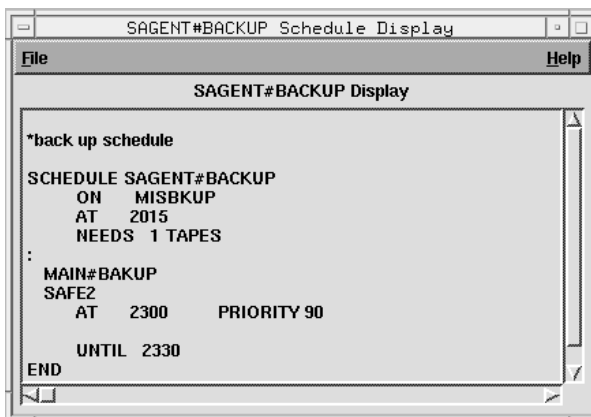
In the Schedule Name box, enter the name of the schedule (up to sixteen alphanumeric, dash (-), and underscore (_) characters starting with a letter). Click OK to open a Schedule Definition window for the new schedule. All of the fields in the Schedule Definition are cleared or set to their default values and the schedule name is displayed in the title bar of the window. (If the schedule name already exists, no Schedule Definition window opens and you get an error.) Click Cancel to close the dialog and create no new schedule. See [Selecting Schedule Dependencies](#) starting on page 5-6 for information on completing the definition of the schedule.

Modifying a Schedule

To modify a schedule definition, select the schedule in the List of Schedules window and then Modify... from the Actions menu. The Schedule Definition window for the selected schedule opens permitting you to alter its characteristics. See [Selecting Schedule Dependencies](#) on page 5-6 for information on the definition of the schedule.

Displaying a Schedule

To display a schedule definition, select the schedule in the List of Schedules window and then Display... from the Actions menu. The Schedule Display window for the selected schedule opens permitting you to view the schedule definition without the ability to modify it.



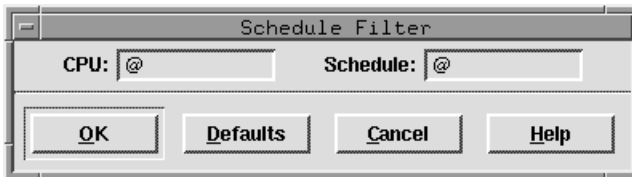
The schedule definition is presented in Maestro's scheduling language (see [The Scheduling Language](#) on page 8-50 for more information).

Deleting a Schedule

To delete a schedule, select it in the List of Schedules and select Delete.... from the Actions menu. A dialog opens confirming that you want to delete the schedule. Click Yes to delete it, or click No to take no action.

Filtering the List of Schedules

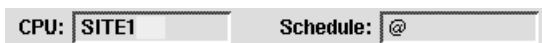
To filter the List of Schedules window, select Filter... from the View menu. The Schedule Filter dialog opens.



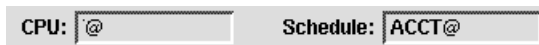
In the CPU and Schedule text entry areas, enter the strings you want Composer to use to filter the list. Wildcards are permitted. Click Defaults to enter the "@" wildcard in the boxes, which results in a listing of all schedules. Click OK to filter the List of Schedules based on the selections and close the dialog. Click Cancel to close the dialog and take no filter action.

Schedule Filter Examples

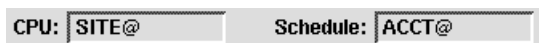
1. The selections below filter the List of Schedules window to display only schedules that contain `SITE1` as the CPU string:



2. The selections below filter the List of Schedules window to display all schedules that contain `ACCT` in the beginning of the Schedule string:

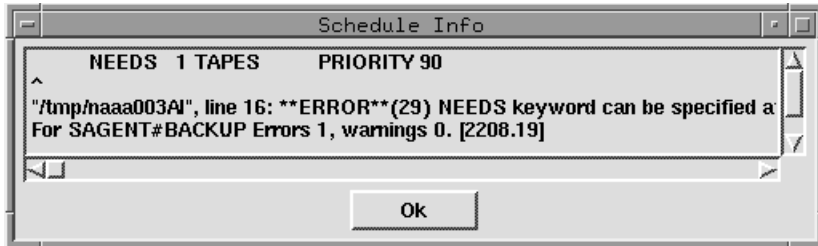


3. The selections below filter the List of Schedules window to display all schedules that contain `SITE` at the beginning of the CPU string and `ACCT` in the beginning of the Schedule string:



Debugging a Schedule

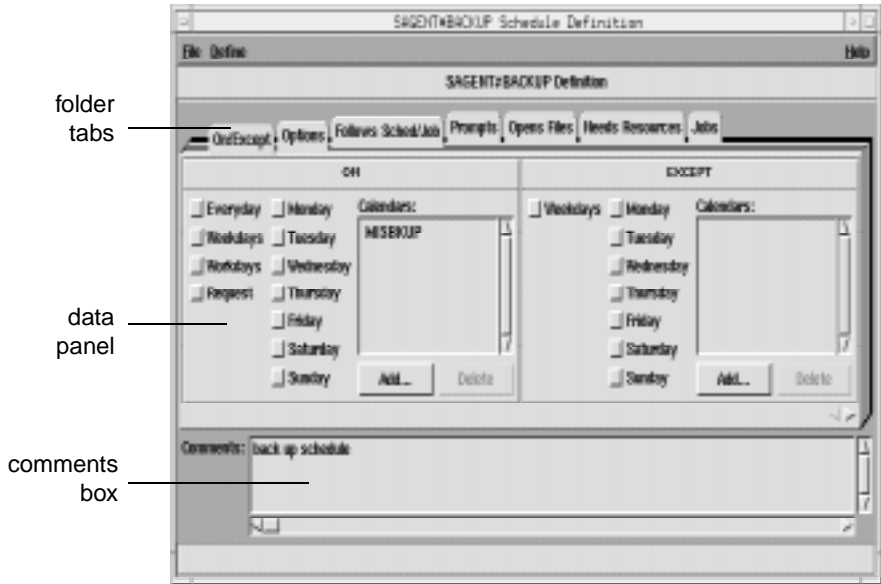
If you try to save or open a schedule that is not properly defined, the Schedule Info window opens. It displays error and warning messages that are useful in debugging a schedule. Note that schedules can be saved even if they generate warnings and errors.



Click OK to close the window.

Selecting Schedule Dependencies

The Schedule Definition window is used to compose schedules. It contains seven data panels selected one at a time from a set of tabs, and a Comments box for entering free-form comments and notes.



Tabs and Data Panels

The Schedule Definition window provides seven dependency categories, listed in the tabs at the top of the folders, for defining a schedule. Each tab has its own data panel:

On/Except Used to specify when the schedule will, or will not, execute.

Options Used to specify:

- the daily time frame within which the schedule will execute,
- the schedule's priority and job limit, and
- the state of the schedule's carry forward option.

Follows Sched/Job Used to specify jobs and schedules that must complete successfully before the schedule will launch.

Prompts	Used to specify prompts that must be issued before the schedule can be launched.
Opens Files	Used to specify files that must be available before the schedule can be launched.
Needs Resources	Used to specify resources that must be available before the schedule can be launched.
Jobs	Used to specify jobs to be included in the schedule. This panel contains the facility to specify dependencies for scheduled jobs.

Clicking on a new tab list changes the data panel. Each category data panel is described in detail under its heading in this section.

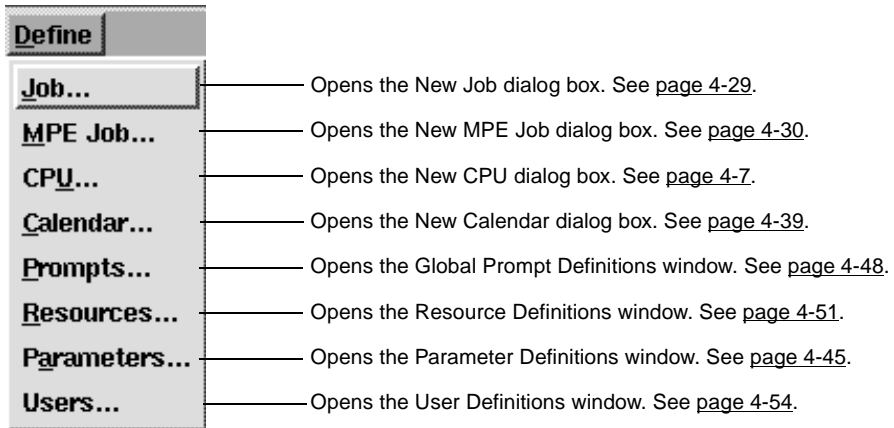
Comments Box

Use the Comments box to enter notes pertaining to the schedule (there is no character limit). There is only one comment box for each schedule; it does not change when the data panel changes.

Note: If comments were interspersed throughout a schedule definition created using the Maestro CLI and the schedule is displayed in the GUI, the schedule's comments are extracted, concatenated, and displayed together in the Comments box.

Defining Objects While Scheduling

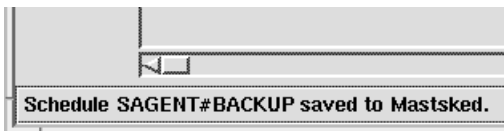
Normally, scheduling objects are defined prior to schedule definition, but the Define menu allows you to create new scheduling objects directly from the Schedule Definition window.



The menu items open the appropriate starting dialog boxes or windows to define the objects. See section 4, *Composing Scheduling Objects* for more information on object definition.

Saving a Schedule

To save the current schedule, select Save from the File menu. When the schedule is saved a message appears at the bottom of the Schedule Definition window and the window remains open so you can continue working.



If you close the Schedule Definition window without saving your changes, a dialog asks if you want to save the changes.

Order of Schedule Dependencies

When a schedule is saved, its dependencies are grouped and ordered as follows, regardless of the order they were selected:

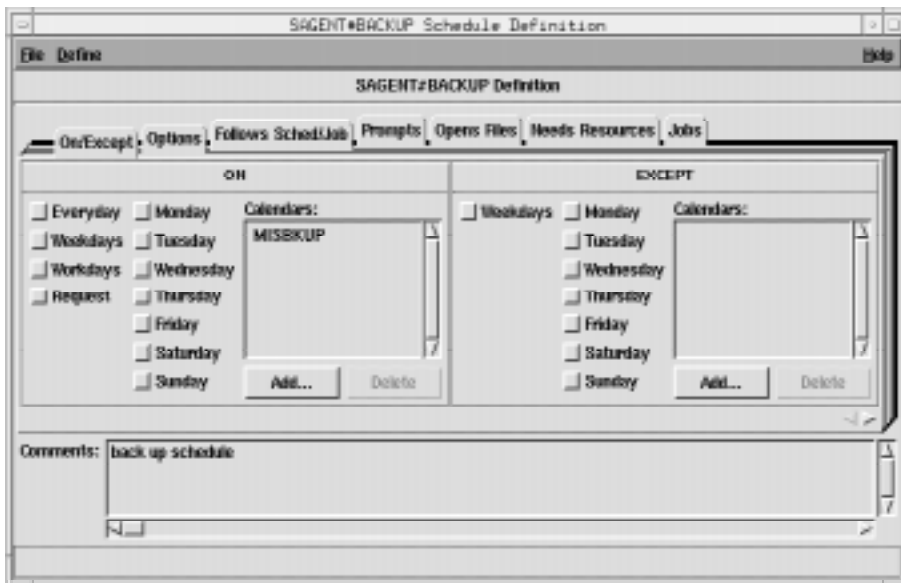
- 1 - Follows Sched/Job
- 2 - Prompts
- 3 - Opens
- 4 - Needs Resources

This is also the order in which the dependencies are resolved before a schedule is executed during production.

Note: For schedules created using the command line interface (CLI), an order is not imposed on dependencies, and they are resolved in the order they were written into the schedule. However, if a schedule is modified and saved using the graphical user interface (GUI), its dependencies are placed in the order shown above, and they are resolved in that order.

On/Except Panel: Schedule Execution Days

The On/Except panel is split into two sections. The ON section specifies when, or how often, a schedule is selected for execution by Maestro. The EXCEPT section specifies exceptions to the ON selections. If a day or date is selected in both the ON and EXCEPT panels, it logically defaults to the EXCEPT category. A schedule must have at least one ON selection.



Selecting Schedule Execution Days

Click the desired buttons to specify when the schedule will run.

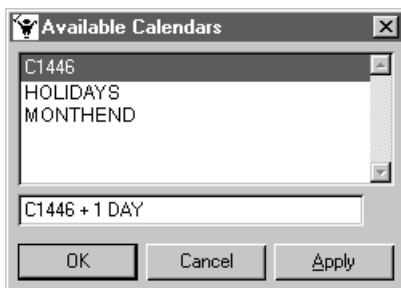
- | | |
|------------------------|--|
| Everyday | Every day. |
| Weekdays | Every day except Saturday and Sunday. |
| Workdays | Same as Weekdays, but it excludes the dates in the HOLIDAYS calendar. For this option to work properly, a HOLIDAYS calendar must exist. Selecting Workdays results in a warning when no HOLIDAYS calendar exists. However, Workdays can be selected, even if the HOLIDAYS calendar does not exist. |
| Request | On request only–no automatic selection. |
| Monday - Sunday | A day of the week. |

Selecting Schedule Execution Calendars

The Calendars list displays calendars on whose dates the schedule will execute.

Adding a Calendar to the List

To add a calendar to the list, click the Add... button. The Available Calendars dialog opens displaying a list of defined calendars.



Select a calendar in the list. In the entry field below the list, you can also add an offset expression in the form:

$$\{+|- \}n \{ \text{day}[s] | \text{weekday}[s] | \text{workday}[s] \}$$

where:

- day[s]** Includes every day of the week.
- weekday[s]** Includes every day except Saturday and Sunday.
- workday[s]** Same as **weekday[s]**, but excludes all dates that appear on a calendar named **holidays**, if it exists.

Click OK or Apply to add the calendar to the ON Calendars list– OK closes the dialog. Click Cancel to close the dialog and without selecting calendar.

Deleting a Calendar to the List

To delete a calendar from the ON Calendars list, select it in the list and click the Delete button.

Adding a New Calendar

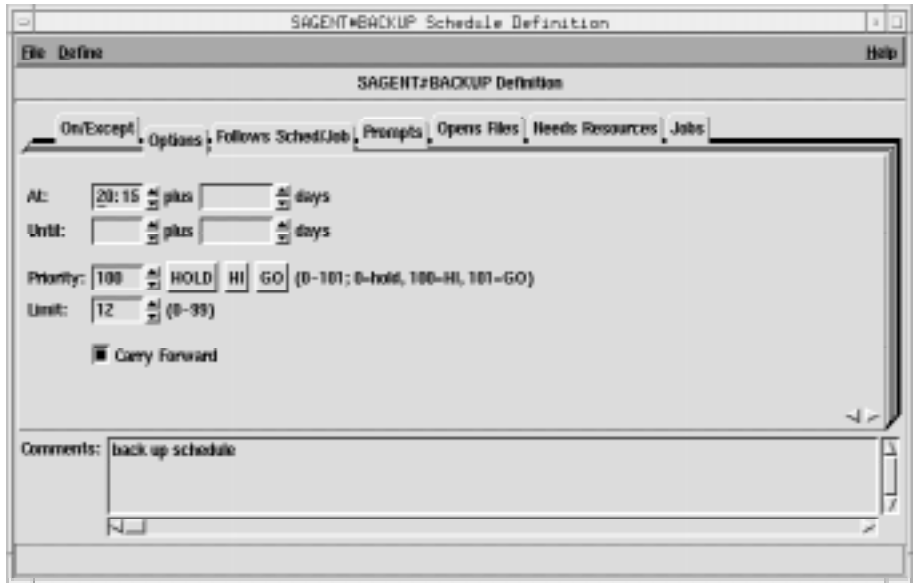
If you want to define a new calendar, select the Calendars... item from the Define menu of the Schedule Definition window. See [Creating a New Calendar](#) on page 4-39 for instructions on how to proceed.

Selecting Schedule Exclusion Days and Calendars

To define when the schedule will not run, use the EXCEPT section of the ON/EXCEPT panel. Use the same procedures described under *Selecting Schedule Execution Days* on page 5-10 and *Selecting Schedule Execution Calendars* on page 5-11.

Options Panel: Selecting Schedule Options

The Options panel specifies the time of day a schedule is to start and stop executing, it's priority, it's job limit, and whether or not it will be carried forward to the next day if it does not finish.



In the time fields, the arrows change the value in 15 minutes increments; in all other fields, the arrows change the value by one unit.

Selecting Schedule Execution Start Time

The At fields define the time of day the schedule is launched. Enter the time, in 24-hour format, in the box to the right of At. The range is 00:00-23:59.

You can enter an offset At time using the plus entry box. Enter an offset in days (0-99) from the scheduled launch time. The offset can be used at the schedule or job level, but not both. The offset days are Maestro processing days, not regular calendar days. When used at the schedule level, the offset is applied to all jobs in the schedule.

When using At and Until in the same schedule, you must make certain that the Until time is later than the At time. Using both At and Until creates a window within which the schedule is executed.

Selecting Schedule Execution Until Time

The Until fields define the time of day after which a schedule will not be launched, regardless of other dependencies. See [Selecting Schedule Execution Start Time](#) on page 5-13 for instructions on how to specify the time and offset, and for more information about using Until.

Until Time Notification

Maestro interprets the expiration of an Until time as a possible job or schedule event. If a job or schedule does not reach the SUCC or DONE state before its Until expires, Maestro writes a message to its standard list file and to the Maestro Console (see [Displaying Messages in the Console](#) on page 6-4 for more information on the Console). Once an Until time event has occurred, Maestro does not recognize another one for the job or schedule unless its Until time is subsequently changed to a later time and the threshold is passed again. For information on managing production see section 6, [Managing Production](#). For information regarding Until time event configuration see `bm check until` on page 2-9.

Warning Messages

The Until time notification messages written to the Maestro console and Batchman's standard list file are:

```
201 Schedule cpu#sched UNTIL time hh:mm has occurred
202 Job cpu#sched.job UNTIL time hh:mm has occurred
```

If the `cpu` in `cpu#sched` is the local cpu, then `cpu#` is dropped from the message.

Selecting a Schedule's Priority

Use the Priority field to assign priorities to Maestro schedules and jobs. Given two schedules in production whose dependencies are satisfied, the one with the higher priority will launch first.

To define a priority, enter a number (0-101) or click HOLD, HI, or GO. HOLD is the same as zero, HI is 100, and GO is 101. Only integers are accepted. If HOLD, HI, or GO is selected, the corresponding value is automatically entered in the field. If no selection is made, 10 is assigned when the schedule is queued for execution.

HOLD, or zero, prevents a schedule from being launched. In the case of HI and GO, all jobs in the schedule are given HI or GO priority. HI and GO jobs are launched as soon as their dependencies are satisfied, overriding the cpu

job limit, but not overriding the schedule's job limit or Maestro's job fence. For more information see [Changing a Cpu's Fence](#) on page 6-27 and [Changing a Cpu's Job Limit](#) on page 6-26.

Selecting a Schedule's Limit

Use the Limit field to specify the maximum number of jobs that can be executing concurrently in the schedule. Enter a number in the range 0-99. A limit of zero prevents jobs from launching. If no limit is entered, the limit is set to 99.

Specifying a Schedule to Carry Forward to the Next Day

Maestro can carry a schedule forward into the next processing day if it has not completed successfully. Click the Carry Forward button to enable this feature. Schedules that are carried forward retain the Carry Forward option, and therefore, may be carried forward again. See the [Carry Forward Options](#) on page 2-6 for more information.

Offset Launch Time Examples

These examples assume Maestro's processing day begins at 6:00 a.m. They illustrate a method of scheduling in advance of a weekend. Pre-production processing is performed on Friday morning to select and prepare a production schedule for Friday, Saturday, and Sunday. Figure 5-1 shows the timing of schedules and jobs. For more information on Maestro pre-production processing see [Pre-Production](#) starting on page 3-11.

1. Launch schedule WKEND1 on Friday, no sooner than 6:30 p.m., and launch job WE1 no sooner than 3:00 a.m. Saturday morning:
 - For schedule WKEND1, in the On/Except panel, click Friday. In the Options panel, select an At time of 18:30.
 - For job WE1, in the Options panel, select an At time of 3:00.
2. Launch schedule wkend2 on Saturday, no sooner than 8:30 a.m., and launch job we2 no sooner than 2:00 p.m. Saturday afternoon:
 - For schedule WKEND2, in the On/Except panel, click Friday. In the Options panel, select an At time of 8:30 plus 1 day.
 - For job WE2, in the Options panel, select an At time of 14:00.

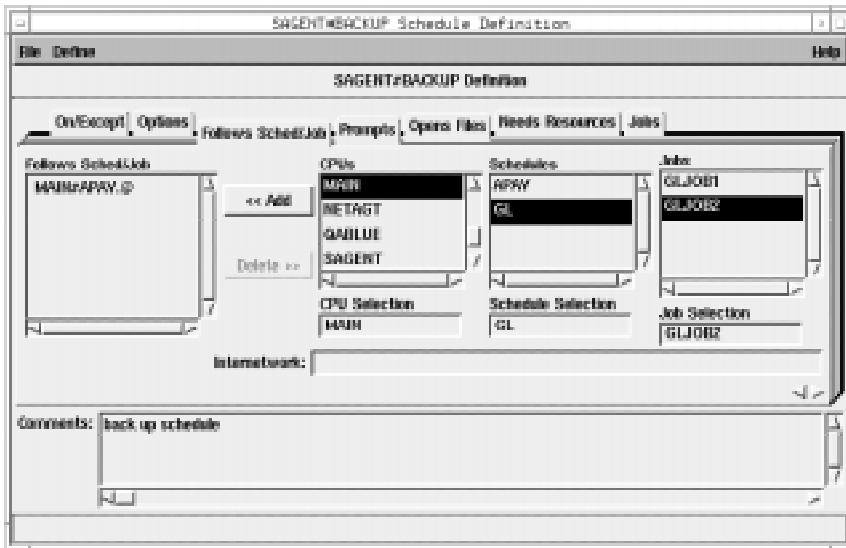
Options Panel: Selecting Schedule Options

Calendar Day	Maestro's Day	
Friday	Thursday	0000 0600 5:59 am Post-production processing for Thurs Pre-production processing for Fri
	Friday	1200 1800 6:30 pm Launch schedule WKEND1
Saturday	Friday	0000 3:00 am Launch job WE1
	Saturday	0600 8:30 am Launch schedule WKEND2 1200 1800 2:00 pm Launch job WE2
Sunday	Saturday	0000
	Sunday	0600 8:30 am Launch schedule WKEND3 1200 11:00 am Launch job WE3 1800
Monday	Sunday	0000
	Monday	0600 5:59 am Post-production processing for Fri, Sat, Sun Pre-production processing for Mon

3. Launch schedule wkend3 on Sunday, no sooner than 8:30 a.m., and launch job we3 no sooner than 11:00 a.m. Sunday morning:
 - For schedule WKEND3, in the On/Except panel, click Friday. In the Options panel, select an At time of 8:30 plus 2 days.
 - For job WE3, in the Options panel, select an At time of 11:00.
4. The same job timing established in the preceding examples can be achieved in a single schedule as follows:
 - For schedule WKEND1, in the On/Except panel, click Friday. In the Options panel, select an At time of 18:30.
 - For job WE1, in the Options panel, select an At time of 3:00.
 - For job WE2, in the Options panel, select an At time of 14:00 plus 1 day.
 - For job WE3, in the Options panel, select an At time of 11:00 plus 2 days.

Follows Sched/Job Panel: Select Processing for a Schedule to Follow

The Follows Sched/Job panel specifies the other schedules and jobs that must be completed before this schedule can be launched. The panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of jobs and schedules that this schedule must follow. The right side contains lists from which to choose available jobs and schedules.



If a “followed” job or schedule is not selected to run on the same day as a schedule that “follows” it, the dependency has no effect.

Reading the Follows List

The Follows Sched/Job list displays the schedules and jobs that must be completed successfully before this schedule can be launched. A list item consists of the qualified Maestro schedule or job name. That is, the cpu or cpu class, the schedule name, and the job name (if any). The cpu or cpu class and schedule are delimited by a pound sign (#) and the schedule and job are delimited by a period (.). For example,

`CPU1#SCHED2.APJOB1`

is a job named APJOB1, in the schedule named SCHED2, defined on the cpu or cpu class named CPU1. If the cpu or cpu class is the same as this schedule, then it is omitted in the Follows Sched/Job list.

Selecting and Adding Jobs and Schedules to Follow

The three lists on the right are used to select the followed jobs and schedules. The CPU Selection determines what is displayed in the Schedules list. The Schedule Selection determines what is displayed in the Jobs list. A cpu, schedule, or job can be selected by either selecting it from the appropriate list or by typing its name in the appropriate Selection box.

For Job Selection, the @ wildcard can be used by itself to indicate all jobs in a schedule.

When a job or schedule is in a selection box, the Add button is enabled. Click Add to move the selection to the Follows Sched/Job list. Double-clicking on the selected job or schedule in the lists also adds it to the Follows Schedules/Job list. (To select a schedule, the Job Selection box must be empty or have the @ wildcard entered.)

Specifying Internetwork Dependencies

Use the Internetwork entry box to specify remote jobs and schedules as Follows dependencies. Entries must be made in this format:

`net::net_dep`

`net` The name of the Maestro Network agent where the internetwork dependency is defined. The two colons (::) are a required delimiter.

`net_dep` The name of the internetwork dependency in the format:

`[cpu#] sched[. job]`

If no `cpu` is specified, the default is the Network agent's host.

The example below specifies job TASK1 in schedule SKED1 that runs on cpu SITE1. SITE1 is a cpu in a remote Maestro network that connects to the local Maestro network via the Network agent NETWAGNT.

Internetwork: `netwagnt::site1#sked1.task1`

When a value is entered in the Internetwork field, the Add button is enabled. Click Add to insert the selection to the Follows Sched/Job list. See appendix D for more information on internetwork dependencies and Network agents.

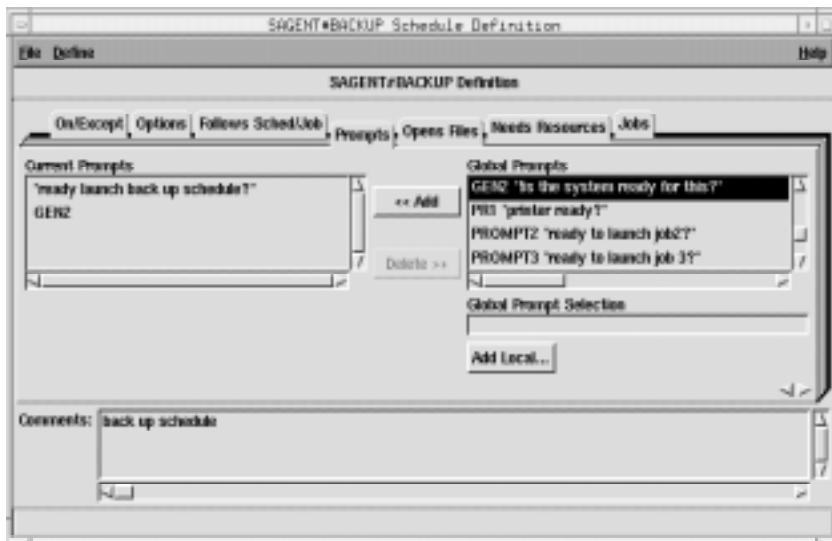
Deleting a Job or Schedule as a Follows Dependency

Selecting one or more items from the Follows Sched/Job list enables the Delete button. Click Delete to remove selected items.

Prompts Panel: Selecting Prompts for a Schedule

The Prompts panel specifies the prompts that must be issued or answered affirmatively before the schedule can be launched. Both global and local prompts can be added to the schedule definition from this panel. Global prompts can be replied to by name. When a global prompt is issued, only one reply is necessary to release all jobs and schedules that use it as a dependency. Local prompts are linked to an individual job or schedule and are created as dependencies within that individual job or schedule definition. For more information on global prompts, see [Global Prompts](#) on page 4-48.

The prompts panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of prompts that must be responded to. The right side contains a list from which to choose available prompts.



Reading the Current Prompts List

This list displays the prompts selected for the schedule. Global prompts are listed by name; local prompts are listed with the text of the prompt in double quotes.

Selecting and Adding a Global Prompt

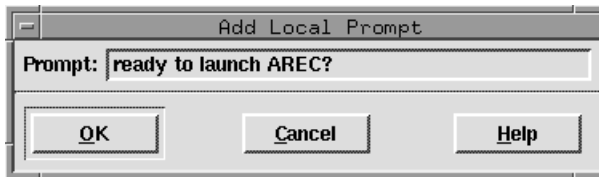
Select a global prompt either by typing its name in the Global Prompt Selection box or by selecting it from the Global Prompts list. The list displays the name of the prompt followed by the prompt text in double quotes.

If you want to define a new global prompt from this window, select Prompts... from the Define menu to open the Global Prompt Definitions window. See [Global Prompts](#) on page 4-48 for more instructions on defining prompts.

When a global prompt is selected, the Add button is enabled. Click Add to insert the prompt into the Current Prompts list. Double-clicking on the desired prompt in the Global Prompts list also adds it to Current Prompts.

Defining and Adding a Local Prompt to a Schedule

To define a local prompt, click Add Local... to open the Add Local Prompt dialog



Enter the desired prompt text in the Prompt entry box (up to 255 characters). If the text begins with a colon (:), the prompt is issued, but no reply is necessary to continue processing. If the text begins with an exclamation point (!), the prompt is not issued, but a reply is necessary to proceed. Otherwise, the prompt is issued and a reply is necessary to continue. You can include backslash "n" (\n) within the text to cause a new line.

Maestro parameters are permitted and must be enclosed in carets (^). For more information on parameters see [Parameters](#) on page 4-45.

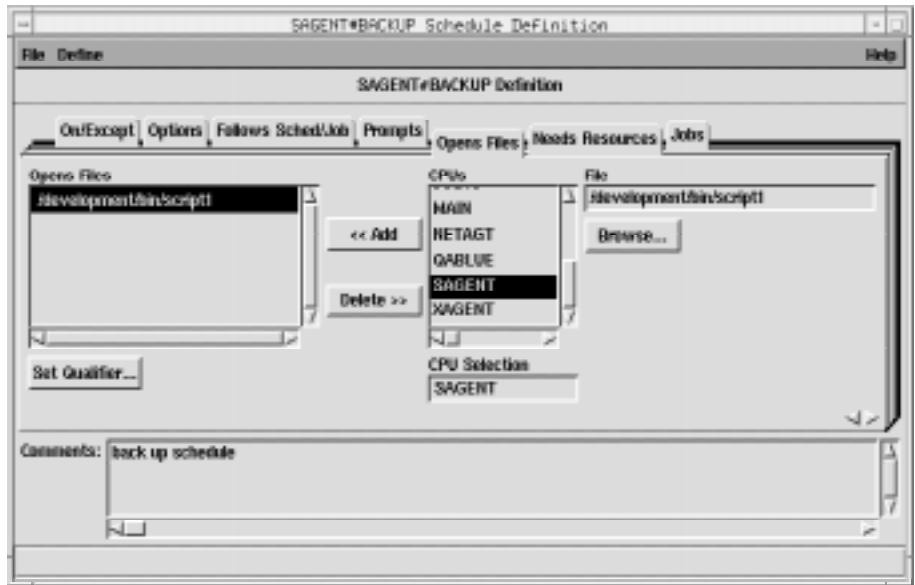
Click OK to add the prompt to the Current Prompts list. Click Cancel to define no local prompt and close the dialog.

Deleting Prompts From a Schedule

Selecting one or more prompts from the Current Prompts list enables the Delete button. Click Delete to remove the selected prompts.

Opens Files Panel: Selecting Schedule File Dependencies

The Opens Files panel specifies files that must exist before a schedule can be launched. File dependencies provide a convenient way to prevent jobs from failing due to unavailable files. However, this process does not provide an absolute guarantee: Maestro checks to see that the file is available for the type of access requested, but it does not reserve the file. It is possible, therefore, for another process to gain access to the file at any time, making it unavailable to the dependent job or schedule.



Note: On Windows NT, the access check for files is done by `maestro`, and on UNIX it is done by `root`.

The panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of file dependencies. The right side contains lists of available files from which to choose.

Reading the Opens Files List

The Opens File list contains the cpu and path names of the files on which the schedule is dependent. The cpu and path are delimited by a pound sign (#). Any test qualifiers are included in parenthesis after the file name. See [Specifying a Qualifier for a File Dependency](#) on page 5-23 for more information.

Note: The combination of path and qualifier cannot exceed 148 characters, within which the basename of the path cannot exceed 28 characters.

Selecting a File for a Schedule Dependency

Select the cpu or cpu class on which the desired file resides either by typing its name in the CPU Selection box or by selecting it from the CPUs list. If the current schedule is defined for a cpu class, then only that class is included in the CPUs list.

Select the path name of the file either by selecting it with the file selection dialog opened by clicking Browse... or by typing it directly. One or more Maestro parameters can be used and must be enclosed in carets (^). For more information on parameters see [Parameters](#) on page 4-45. For instructions on how to use a file selection box see [Selecting a File with the Browser](#) on page 4-5.

Note: The file browser only displays the files and directories for the cpu running Composer.

Adding a File Dependency to a Schedule

When a file pathname is in the File box, the Add button is enabled. Click Add to move the pathname to the Opens Files list. If the file is already in the list, it is not added again.

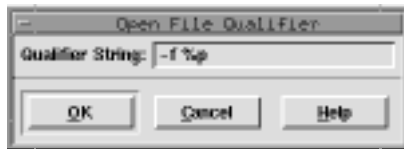
Note: When using a cpu class as the CPU Selection for the file dependency, if the file does not exist on some cpus in the class, then the schedules on those cpus cannot launch because the dependency cannot be resolved.

Deleting a File Dependency from a Schedule

When one or more file pathnames are selected in the Opens Files list, the Delete button is enabled. Click Delete to remove selected file pathnames.

Specifying a Qualifier for a File Dependency

To specify a test option for a file dependency, select the file in the Opens Files list and then click the Set Qualifier... button. The Open File Qualifier dialog opens.



Enter the file qualifier in the Qualifier String box (Maestro adds the parenthesis to the qualifier when the string is appended to the file in the Opens File list). Click OK to append the qualifier to the selected file pathname in the Opens Files list. Click Cancel to retain the previous qualifier.

Test Qualifiers - UNIX

For files on Maestro-UNIX cpus, any valid condition in the **test(1)** command can be used.

Test Qualifiers - Windows NT

The valid test qualifiers for files on Maestro-Windows NT cpus are listed below. The expression "%p" inserts the path name of the file.

- d %p True if the path name is a directory.
- e %p True if the file exists.
- f %p True if the file is an ordinary file.
- r %p True if the file is readable.
- s %p True if the file's size is not zero.
- w %p True if the file is writeable.

Security for test(1) Commands

On 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:

1. The Local Option `jm no root` must be set to `no`.
2. In the Security file, the user documenting the schedule, or adding the Opens Files dependency with a `comman adddep` command, must have `submit` access to a job with the following attributes:

```
name=cmdstest.fileeq
logon=root
jcl=the path name of the opens file
cpu=the cpu on which the opens file exists
```

Note that `cmdstest`, and `fileeq` do not actually exist.

File Test Examples

1. On a Maestro-Windows NT cpu, check to see that file `c:\usr\mis\data+33` is available for read access, and contains at least one record, before launching a dependent schedule or job:

```
c:\usr\mis\data+33(-s %p)
```
2. On a Maestro-Windows NT cpu, check to see if three directories- `\john`, `\mary`, and `\roger`- exist, before launching a dependent job or schedule:

```
d:\users (-d %p\john -a -d %p\mary -a -d %p\roger)
```
3. On a Maestro-UNIX cpu, check to see if `cron` has created its `FIFO` file before launching a dependent job or schedule:

```
/usr/lib/cron/FIFO (-p %p)
```
4. On a Maestro-Windows NT cpu, check to see that file `d:\users\john\execit1` on cpu `dev3` exists and is executable, before launching a dependent job or schedule:

```
dev3#d:\users\john\execit1(-x %p)
```
5. On a Maestro-UNIX cpu, check to see that file `/usr/tech/checker/data-file` on cpu `nyc` exists with a size greater than zero, and is writable, before launching a dependent job or schedule:

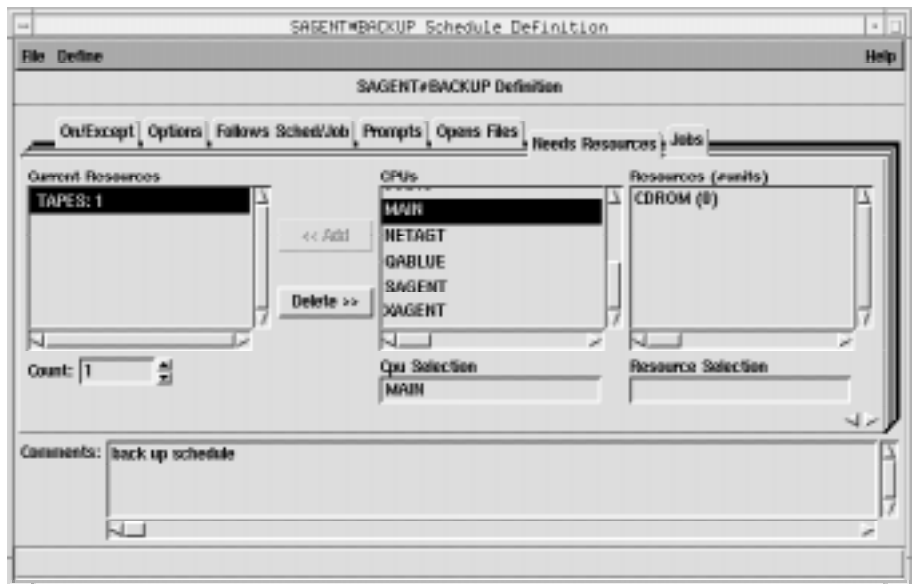
```
nyc#/usr/tech/checker/data-file(-s %p -a -w %p)
```


Needs Resources Panel: Assigning Resources to a Schedule

The Needs Resources panel specifies Maestro resources that must be available to the schedule before it can be launched. You can have a resource dependency at either the job or schedule level, but not both. The number of jobs and schedules using a resource at any one time cannot exceed 32.

Resources are allocated to jobs and schedules when they start executing and released when the job or schedule enters the SUCC or ABEND state. Units can be released early by executing the `release` command within a job script (see [release](#) on page 10-27).

The Needs Resources panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of resources for the schedule. The right side contains lists from which to choose available resources.



Reading the Current Resources List

The Current Resources list contains the resources that must be available for the schedule to launch. The resource name is preceded by the name of the cpu or cpu class on which the resource is defined and followed by the number of units of the resource required by the schedule. The cpu and resource are

delimited by a pound sign (#) and the resource and units allocated are delimited by a colon (:). For example,

```
FTACPU#TAPE:1
```

is one (1) unit of the resource named TAPE defined on the cpu or cpu class named FTACPU. If the resource is defined on the same cpu or cpu class as this schedule, then the cpu and delimiter are omitted in the list. If the resource is defined for a cpu class, the class name is omitted in the list.

Selecting a Resource

To select a resource, enter the name of the cpu or cpu class on which it is defined by either typing the name in the CPU Selection box or by selecting one from the CPUs list. The Resources list is filtered to display the resources for that cpu or cpu class. If the current schedule is defined for a cpu class, only that class is included in the list.

Complete the selection either by typing a resource name in the Resource Selection list or by selecting a resource from the Resources list. In the list, the name of the resource is followed by the total number of units in parenthesis. For example,

```
ARRAY(12)
```

is a resource named ARRAY with 12 total units available.

If you want to define a new resource from this window, select Resources... from the Define menu to open the Resource Definitions window. See [Resources](#) on page 4-51 for instructions on how to define resources.

Adding a Resource to a Schedule

When a resource name is in the Resource Selection box, the Add button is enabled. Click Add to add it to the Current Resources list. Double-clicking the resource name in the Resources list also adds it to the Current Resources list. If more than 32 units of the resource are needed, see [Changing the Number of Resource Units a Schedule Needs](#) on page 5-27.

Note: Adding a resource dependency to a job that runs on a fault-tolerant agent (FTA), or to a schedule that contains a job that runs on an FTA, may prevent the job from executing if the resource is not defined on

the FTA. To prevent this from happening, make certain that the cpu definition for the FTA has Full Status on, or, alternatively, make certain that the resource is defined on the FTA on which the job runs.

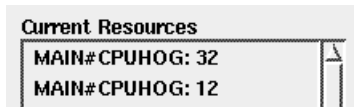
Deleting a Resource from a Schedule

Selecting one or more resources in the Current Resources list enables the Delete button. Click Delete to remove selected resources.

Changing the Number of Resource Units a Schedule Needs

To change the number of resource units a schedule needs, select the resource in the Current Resources list and enter a number in the entry box or use the up/down arrow buttons. The default count is one unit. If more than one item is selected in the Current Resources list, the Count field is disabled.

A maximum of 32 units is permitted per resource entry. If more than 32 units are needed, add the resource again. For example, if a schedule requires 44 units of MAIN#CPUHOG, the resource can be entered twice:



The total number of resources that can be assigned to a schedule is 1024.

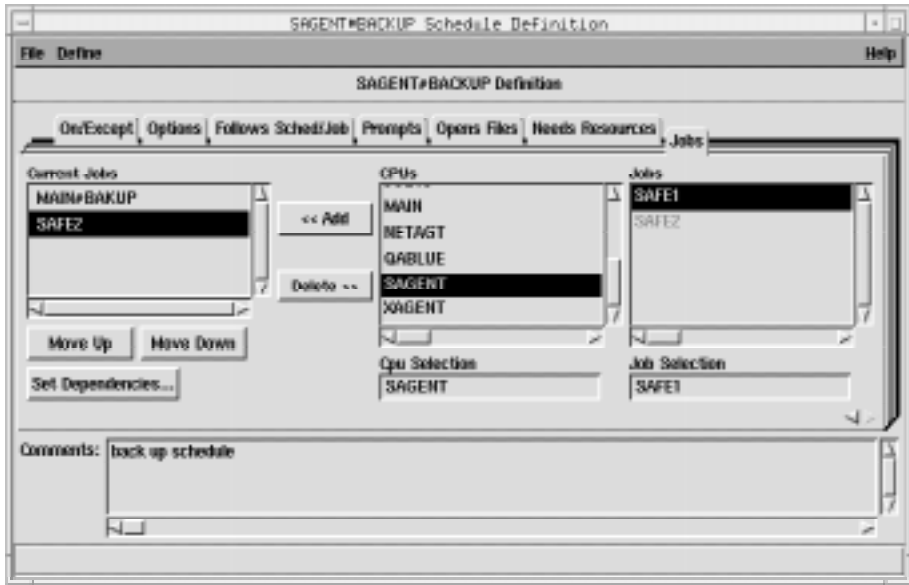
Examples of Controlling Production through Resources

Both examples assume that you have three schedules defined to run on the same day.

1. All three schedules are cpu-intensive and no more than two can run concurrently. To enforce this, first define a resource named, for example, CPUHOG with two total units. Then specify that each schedule needs one unit of CPUHOG before it can be launched. Since there are only two units of CPUHOG in total, only two schedules can run concurrently.
2. All three schedules are back-up related and each needs a tape drive to complete its task. There is only one tape drive to share. To ensure that each schedule has access to the tape drive, first define a resource named, for example, DRIVE with one total unit. Then specify that each schedule needs one unit of DRIVE before it can launch. The schedules will launch one at a time.

Jobs Panel: Placing Jobs in a Schedule

The Jobs panel specifies jobs that will execute in the current schedule. It also provides the facility to add dependencies to the jobs placed in the schedule. The panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of jobs that will execute in the schedule. The right side contains lists from which to choose available jobs.



Reading the Current Jobs List

The Current Jobs list contains the jobs that will execute in the schedule. The job name is preceded by the name of the cpu or cpu class on which the job is defined, delimited by a pound sign (#). If the job is defined on the same cpu as the current schedule, the cpu name and delimiter are omitted from the entry in the list. If the job is defined for a cpu class, then the class name is omitted from the list.

Selecting Jobs

To select a job to execute in the current schedule, enter the name of the cpu or cpu class on which the desired job is defined by either typing its name in the CPU Selection box or by selecting one from the CPUs list. The Jobs list is filtered to display the jobs for that cpu or cpu class. If the current schedule is defined for a cpu class, then only that class is included in the list.

Complete selection either by typing a job name in the Job Selection box or by selecting a job from the Jobs list. Only Maestro-defined jobs are permitted.

If you want to define a new job from this window, select Job... from the Define menu to open a New Job dialog box. (For a New MPE Job dialog box, select MPE Job... from the Define menu.) See [Creating a New Job](#) on page 4-29 for instructions on how to define jobs.

Adding a Job to a Schedule

When a valid job name is in the Job Selection box, the Add button is enabled. Click Add to append it to the Current Jobs list. Double-clicking on the job name in the list also appends it to Current Jobs. If the job is already in the list, it is not added again.

Deleting a Job from a Schedule

Selecting one or more jobs from the Current Jobs list enables the Delete button. Click Delete to remove selected jobs.

Changing the Order in the Current Jobs List

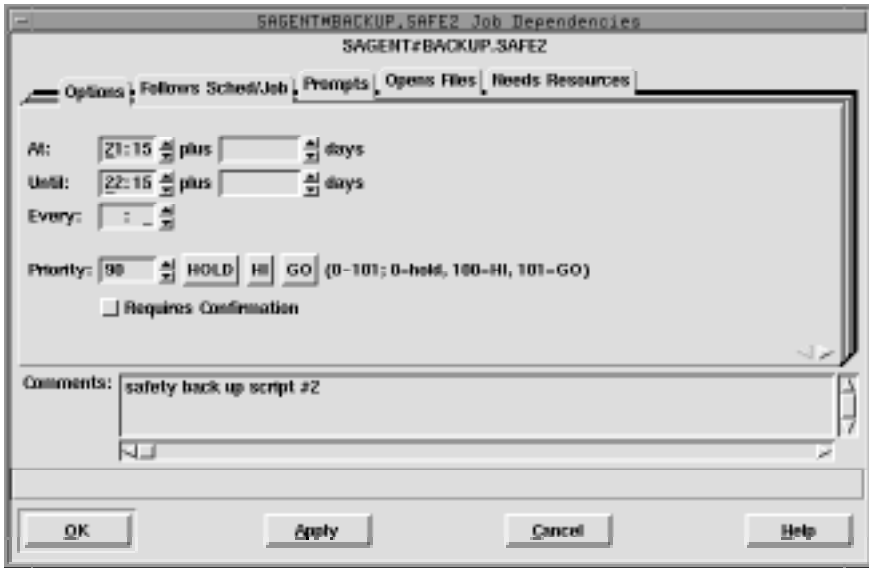
Using the Move Up/Move Down buttons, you can arrange the order of the jobs in the Current Jobs list. To change the order, select a job in the list and click Move Up to raise the job in the list or click Move Down to lower the job in the list. The order that jobs appear in schedule does not determine their order of execution.

Setting Dependencies for a Scheduled Job

To set dependencies for a job, click on the job in the Current Jobs list and then click the Set Dependencies... button. The Job Dependencies window opens. See [Selecting Job Dependencies](#) on page 5-30 for more information.

Selecting Job Dependencies

Dependencies for a job can only be specified once a job is added to a schedule. The Job Dependencies window is used to define the dependencies. It is accessed from the Jobs panel of the Schedule Definition window (see [Adding a Job to a Schedule](#) on page 5-29). When the Job Dependencies window is opened, the schedule and job name are displayed in the title bar.



Dependency Categories and Data Panels

The Job Dependencies window provides dependency categories on the tabs across the top of the window. The Job Dependencies window has five such categories with corresponding data panels:

- | | |
|--------------------------|--|
| Options | Used to specify: |
| | <ul style="list-style-type: none">• the daily time frame within which the job will execute,• how often the job runs,• the job's priority, and• if the job requires confirmation to reach a termination state. |
| Follows Sched/Job | Used to specify jobs and schedules that must complete successfully before the job will launch. |

Prompts	Used to specify prompts that must be issued before the job can be launched.
Opens Files	Used to specify files that must be available before the job can be launched.
Needs Resources	Used to specify resources that must be available before the job can be launched.

Clicking on a new tab changes the data panel. Each category data panel is described in detail under its heading in the sections that follow.

Applying Job Dependencies

Click OK or Apply to apply the current job dependencies to the job (OK also closes the Job Dependencies window). The job dependency information is not actually saved until Save is selected from the Schedule Definition window (see *Saving a Schedule* on page 5-8). Click Cancel to dismiss the window without applying changes made since Apply was last selected. If Cancel is selected and un-applied changes have been made, a warning prompt appears.

Comments Box

Use the Comments box to enter notes pertaining to the job dependencies (there is no character limit). There is a comment box for each job and it does not change when the data panel changes.

Note: If comments were interspersed throughout job dependencies created using the Maestro CLI and the job dependencies are displayed in the GUI, the comments are extracted, concatenated, and displayed together in the Comments box.

Order of Job Dependencies

When a schedule is saved, its job dependencies are grouped and ordered as follows, regardless of the order they were selected:

- 1 - Follows Sched/Job
- 2 - Prompts
- 3 - Opens
- 4 - Needs Resources

Selecting Job Dependencies

This is also the order in which the dependencies are resolved before a job is executed during production.

Note: For schedules created using the command line interface (CLI), an order is not imposed on job dependencies, and they are resolved in the order they were written into the schedule. However, if a schedule is modified and saved using the graphical user interface (GUI), its job dependencies are placed in the order shown above, and they are resolved in that order.

Options Panel: Selecting Job Options

The Options panel specifies the time of day a job is to start and stop executing, how often it is to execute, its priority, and whether or not it needs outside intervention to go to a successful state.



In the time fields, the arrows change the value in 15 minutes increments; in all other fields, the arrows change the value by one unit.

Selecting Job Execution Start Time

The At fields define the time of day the job is launched. Enter the time, in 24-hour format, in the box to the right of At. The range is 00:00-23:59.

You can enter an offset At time using the plus entry box. Enter an offset in days (0-99) from the scheduled launch time. The offset can be used at the schedule or job level, but not both. The offset days are Maestro processing days, not regular calendar days. When used at the schedule level, the offset is applied to all jobs in the schedule.

When using At and Until for the same job, make certain that the Until time is later than the At time. Using both At and Until creates a window within which the job is executed.

For an example of using At and offsets, see [Offset Launch Time Examples](#) on page 5-15.

Selecting Job Execution Until Time

The Until fields define the time of day after which a job will not be launched. See [Selecting Job Execution Start Time](#) above for instructions on how to specify the time and offset, and for more information about using Until.

Until Time Notification

Maestro interprets the expiration of an Until time as a possible job or schedule event. For more information, see [Selecting Schedule Execution Until Time](#) on page 5-14.

Specifying a Repeat Rate for a Job

The Every field defines the rate at which a job is to be launched repeatedly. Enter the repeat rate, in hours followed by minutes, by either typing it in or using the up/down arrows.

If an Every value is used, an attempt is made to launch the job repeatedly at the specified rate. The number of times the job actually executes depends on its dependencies, the level of system activity, and the availability of job slots. All job dependencies must be satisfied each time an attempt is made to launch the job.

A job must complete before another launch attempt is made. For example, if a job is scheduled to run every minute, but it takes seven minutes to run, the job launches every seven minutes, not every minute.

If an At time is specified, the repetition rate is synchronized with that time. If no At time is specified, the repetition rate is synchronized with the initial launch time. If Maestro is started after the At time, all of the repetitions may not take place.

If you cancel a repetitive job, it will not be launched again. If you rerun a repetitive job, the next iteration of the job is run immediately. If you rerun a repetitive job that had been cancelled, the repetition rate is re-instated. If a repetitive job abends, the repetitions continue following the optional recovery action.

Selecting a Job's Priority

The Priority field assigns values to Maestro schedules and jobs to prioritize their relative importance during production. If you have two jobs, both of which have all their dependencies satisfied, the one with the higher priority will launch first.

In the Priority field, enter a number (0-101) or click HOLD, HI, or GO. HOLD is equivalent to a priority of zero, HI is equivalent to 100, and GO is equivalent to 101. Only an integer can be entered in the entry box. If HOLD, HI, or GO is selected, then the corresponding value is automatically entered in the box. If no selection is made, a value of 10 is assigned when the job is queued for execution.

HOLD, or a priority of zero, prevents the job from being launched. HI and GO jobs are launched as soon as their dependencies are satisfied. They override the cpu job limit, but they do not override the schedule's job limit or Maestro's job fence. For more information see [*Changing a Cpu's Fence*](#) on page 6-27 and [*Changing a Cpu's Job Limit*](#) on page 6-26.

Requiring a Job's Completion to be Confirmed

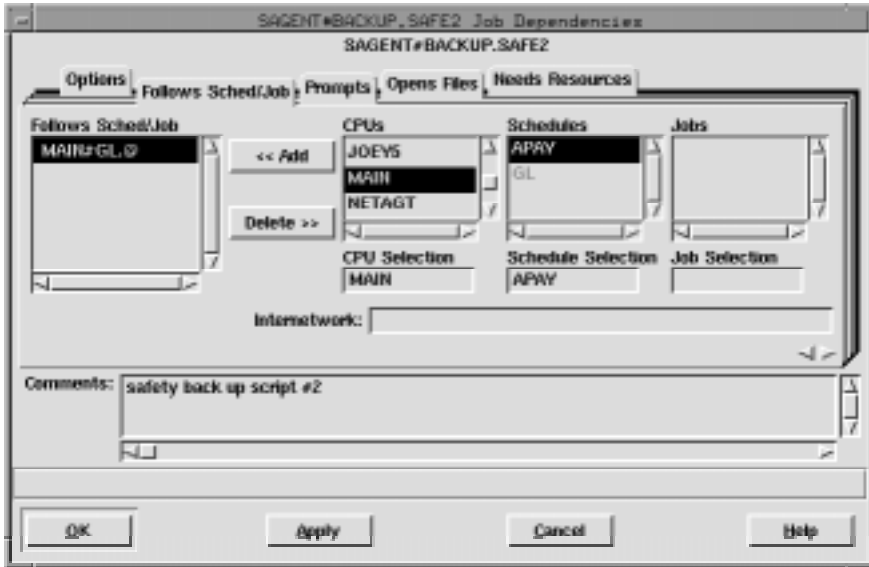
You can specify that the job's completion must be confirmed from Conman before it reaches a termination state. Click the Requires Confirmation button to enable this function.

When a job with Requires Confirmation enabled completes, it remains in the PEND state until confirmation is received. If confirmation is received before the job completes, its state is either SUCCP (successful confirmation received) or ABENDP (abend confirmation received). While a job is in the PEND, SUCCP, or ABENDP state, other jobs and schedules that are dependent on it are not released. Job states can be monitored using the Conman program.

Jobs that are in the PEND, SUCCP, or ABENDP states are not considered part of a schedule's job limit. A transition to one of these states reduces the number of executing jobs by one.

Follows Sched/Job Panel: Selecting Processing for a Job to Follow

The Follows Sched/Job panel specifies the other schedules and jobs that must be completed before this job can be launched. The panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of jobs and schedules that this job must follow. The right side contains lists from which to choose available jobs and schedules.



If a “followed” job or schedule is not selected to run on the same day as a job that “follows” it, the dependency has no effect.

Reading the Follows List

The Follows Sched/Job list displays the schedules and jobs that must be completed successfully before this job can be launched. A list item consists of the qualified Maestro schedule or job name. That is, the cpu or cpu class, the schedule name, and the job name (if any). The cpu or cpu class and schedule are delimited by a pound sign (#) and the schedule and job are delimited by a period (.). For example,

`CPU1#SCHED2.APJOB1`

is a job named APJOB1, in the schedule named SCHED2, defined on the cpu or cpu class named CPU1.

If the `cpu` or `cpu class` is the same as this schedule, then it is omitted in the Follows Sched/Job list. If a job in the same schedule as this job is added as a follows dependency, then the job will be displayed in the Follows Sched/Job list by job name only. The schedule is implied.

Selecting and Adding Jobs or Schedules to Follow

The three lists on the right are used to select the followed jobs and schedules. The CPU Selection determines what is displayed in the Schedules list. The Schedule Selection determines what is displayed in the Jobs list. A `cpu`, schedule, or job can be selected by either selecting it from the appropriate list or by typing its name in the appropriate Selection box.

For Job Selection, the `@` wildcard can be used by itself to indicate all jobs in a schedule.

When a job or schedule is in a selection box, the Add button is enabled. Click Add to insert the selection in the Follows Sched/Job list. Double-clicking on the selected job or schedule in the lists also adds it to the Follows Sched/Job list. (To select a schedule, the Job Selection box must be empty or have the `@` wildcard entered.)

Specifying Internetwork Dependencies

Use the Internetwork entry box to specify remote jobs and schedules as Follows dependencies. Entries must be made in this form:

`net::net_dep`

`net` The name of the Maestro Network agent where the internetwork dependency is defined. The two colons (`::`) are a required delimiter.

`net_dep` The name of the internetwork dependency in the format:

`[cpu#] sched[. job]`

If no `cpu` is specified, the default is the Network agent's host.

The example below specifies job `TASK1` in schedule `SKED1` that runs on `cpu SITE1`. `SITE1` is a `cpu` in a remote Maestro network that connects to the local Maestro network via the Network agent `NETWAGNT`.

Internetwork: `netwagnt::site1#sked1.task1`

When a value is entered in the Internetwork field, the Add button is enabled. Click Add to insert the selection to the Follows Sched/Job list. See appendix D for more information on internetwork dependencies and Network agents.

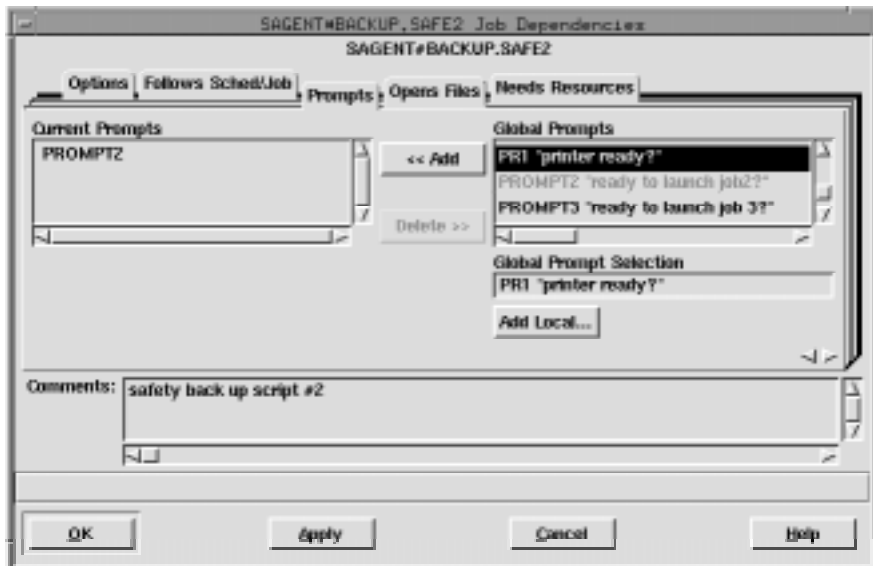
Deleting a Job or Schedule as a Follows Dependency

Selecting one or more items from the Follows Sched/Job list enables the Delete button. Click Delete to remove selected items.

Prompts Panel: Selecting Prompts for a Job

The Prompts panel specifies the prompts that must be issued or answered affirmatively before the job can be launched. Both global and local prompts can be added to the schedule definition from this panel. Global prompts can be replied to by name. When a global prompt is issued, only one reply is necessary to release all jobs and schedules that use it as a dependency. Local prompts are linked to an individual job or schedule and are created as dependencies within that individual job or schedule definition. For more information on Global Prompts, see *Global Prompts* on page 4-48.

The prompts panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of prompts that must be responded to. The right side contains lists from which to choose available prompts.



Reading the Current Prompts List

The Current Prompts list displays the prompts selected for the job. Global prompts are listed by name; local prompts are listed with the text of the prompt in double quotes.

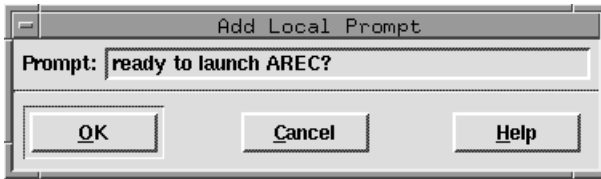
Selecting and Adding a Global Prompt

Select a global prompt either by typing its name in the Global Prompt Selection box or by selecting it from the Global Prompts list. The list displays the name of the prompt followed by the prompt text in double quotes.

When a global prompt is selected, the Add button is enabled. Click Add to insert the prompt into the Current Prompts list. Double-clicking on the desired prompt in the Global Prompts list also adds it to Current Prompts.

Defining and Adding a Local Prompt to a Job

To define a local prompt, click Add Local... to open the Add Local Prompt dialog.



Enter the desired prompt text in the Text entry box (up to 255 characters). If the text begins with a colon (:), the prompt is issued, but no reply is necessary to continue processing. If the text begins with an exclamation point (!), the prompt is not issued, but a reply is necessary to proceed. Otherwise, the prompt is issued and a reply is necessary to continue. You can include backslash "n" (\n) within the text to cause a new line.

Maestro parameters are permitted and must be enclosed in carets (^). For more information on parameters see [Parameters](#) on page 4-45.

Click OK to add the prompt into the Current Prompts list. Click Cancel to define no local prompt and close the dialog.

Deleting Prompts From a Job

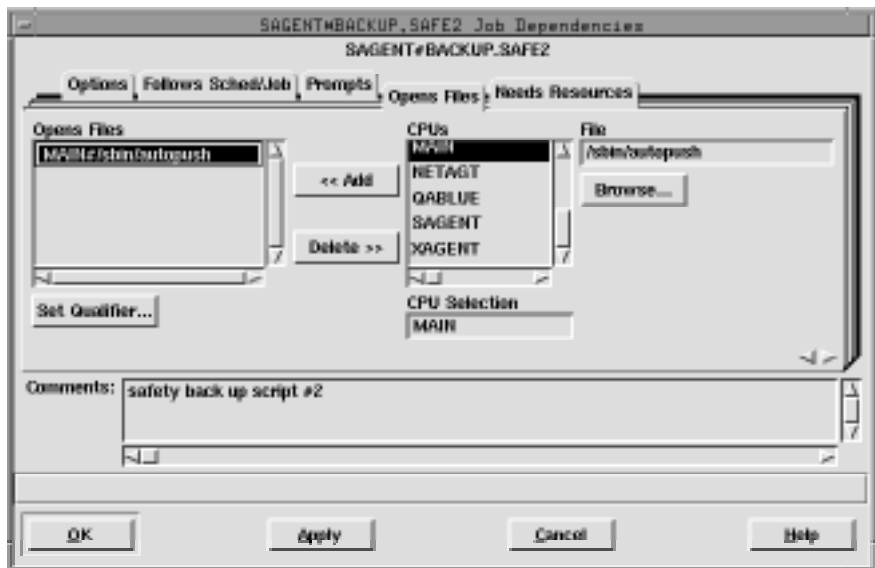
Selecting one or more prompts from the Current Prompts list enables the Delete button. Click Delete to remove the selected prompts.

Opens Files Panel: Selecting Job File Dependencies

The Opens Files panel specifies files that must exist before a job can be launched. File dependencies provide a convenient way to prevent jobs from failing due to unavailable files. However, this process does not provide an absolute guarantee: Maestro checks to see that the file is available for the type of access requested, but it does not reserve the file. It is possible, therefore, for another process to gain access to the file at any time, making it unavailable to the dependent job or schedule.

Note: On Windows NT, the access check for files is done by `maestro`, and on UNIX it is done by `root`.

The panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of file dependencies. The right side contains lists of available files from which to choose.



Reading the Opens Files List

The Opens File list contains the path names of the files upon which the job is dependent. Any test qualifiers are listed at the end of the file name in

parenthesis. If no Qualifier String is specified for a file residing on a Maestro cpu, the default is (-f %p).

Note: The combination of path and qualifier cannot exceed 148 characters, within which the basename of the path cannot exceed 28 characters.

Selecting a File for a Job Dependency

Select the cpu or cpu class on which the desired file resides either by typing its name in the CPU Selection box or by selecting it from the CPUs list. If the current schedule is defined for a cpu class, then only that class is included in the CPUs list.

Select the path name of the file either by selecting it with the file selection box or by typing it directly. One or more Maestro parameters can be used and must be enclosed in carets (^). For more information on parameters see [Parameters](#) on page 4-45. For instructions on how to use a file selection box see [Selecting a File with the Browser](#) on page 4-5.

Note: The file browser only displays the files and directories for the cpu running Composer.

Adding a File Dependency to a Job

When a file pathname is in the File box, the Add button is enabled. Click Add to move the pathname to the Opens Files list. If the file is already in the list, it is not added again.

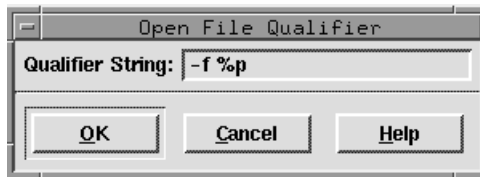
Note: When using a cpu class as the CPU Selection for the file dependency, if the file does not exist on some cpus in the class, then the jobs on those cpus cannot launch because the dependency cannot be resolved.

Deleting a File Dependency from a Job

When one or more file pathnames are selected in the Opens Files list, the Delete button is enabled. Click Delete to remove selected file pathnames.

Specifying a Test Qualifier for a File Dependency

To specify a test option for a file dependency, select the file in the Opens Files list and then click Set Qualifier... The Open File Qualifier dialog opens.



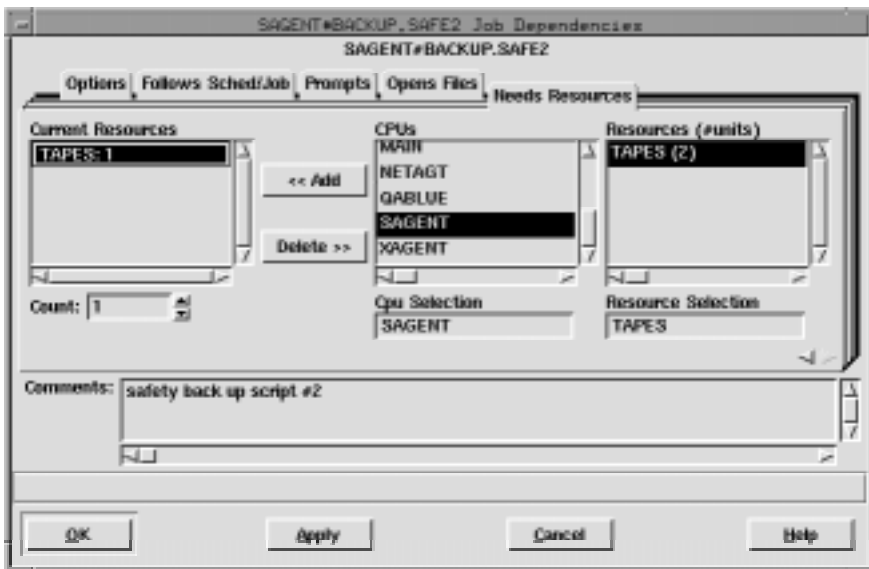
Enter the file qualifier in the Qualifier String box (Maestro adds the parenthesis to the qualifier when the string is appended to the file in the Opens File list). Click OK to append the qualifier to the selected file pathname in the Opens Files list. Click Cancel to retain the previous qualifier. For information about valid qualifiers, and examples see [*Specifying a Qualifier for a File Dependency*](#) on page 5-23.

Needs Resources Panel: Assigning Resources to a Job

The Needs Resources panel specifies Maestro resources that must be available to the job before it can be launched. You can have a resource dependency at either the job or schedule level, but not both. The number of jobs and schedules using a resource at any one time cannot exceed 32.

Resources are allocated to jobs and schedules when they start executing and released when the job or schedule enters the SUCC or ABEND state. Units can be released early by executing the `release` command within a job script (see [release](#) on page 10-27).

The Needs Resources panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of resources for the job. The right side contains lists from which to choose available resources.



Reading the Current Resources List

The Current Resources list contains the resources that must be available for the job to launch. The resource name is preceded by the name of the cpu or cpu class on which the resource is defined and followed by the number of units of the resource required by the job. The cpu and resource are delimited by a pound sign (#) and the resource and units allocated are delimited by a colon (:). For example,

```
MISCPU#TAPE:1
```

is one (1) unit of the resource named TAPE defined on the cpu or cpu class named MISCPU. If the resource is defined on the same cpu or cpu class as this schedule, then the cpu and delimiter are omitted in the list. If the resource is defined for a cpu class, the class name is omitted in the list.

Selecting a Resource

To select a resource, enter the name of the cpu or cpu class on which the desired resource is defined by either typing its name in the CPU Selection box or by selecting one from the CPUs list. The Resources list is filtered to display the resources for that cpu or cpu class. If the current schedule is defined for a cpu class, then only that class is included in the list.

Complete the selection either by typing a resource name in the Resource Selection list or by selecting a resource from the Resources list. In the list, the name of the resource is followed by the total number of available units in parenthesis. For example,

```
ARRAY ( 12 )
```

is a resource named ARRAY with 12 total units available.

Adding a Resource to a Job

When a resource name is in the Resource Selection box, the Add button is enabled. Click Add to append it to the Current Resources list. Double-clicking on the resource name in the Resources list also appends it to the Current Resources list. If the resource is already in the Current Resources list, it is not added again, unless more than 32 units of the resource are needed for the job. See [*Changing the Number of Resource Units a Job Needs*](#) on page 5-46.

Note: Adding a resource dependency to a job that runs on a fault-tolerant agent (FTA), or to a schedule that contains a job that runs on an FTA, may prevent the job from executing if the resource is not defined on the FTA. To prevent this from happening, make certain that the cpu definition for the FTA has Full Status on, or, alternatively, make certain that the resource is defined on the FTA on which the job runs.

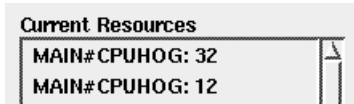
Deleting a Resource from a Job

Selecting one or more resources in the Current Resources list enables the Delete button. Click Delete to remove selected resources.

Changing the Number of Resource Units a Job Needs

To change the number of resource units the job needs, select the resource in the Current Resources list and enter a number in the entry box or use the up/down arrow buttons. The default count is one unit. If more than one item is selected in the Current Resources list, the Count field is disabled.

A maximum of 32 units is permitted per resource entry. However, if more than 32 units are needed, the resource can be added again. For example, if a job requires 44 units of MAIN#CPUHOG, the resource can be entered twice:



Current Resources	
MAIN#CPUHOG: 32	↑
MAIN#CPUHOG: 12	↑

The total number of resources that can be assigned to a job is 1024.

6

Managing Production

Maestro's production environment is managed with the Console Manager (**gconman** or **conman**). It is used to start and stop production processing, alter and display schedules and jobs, and control cpu linking in a network.

Console Manager Interface Basics

The Console Manager's graphical interface (GUI) is described in this section. See section 9, *Conman Command Line Reference* for information about the command line interface (CLI).

Running the Console Manager

To run Conman, log in, set your `DISPLAY` variable, and then enter the following:

```
gmaestro [-backstore]
```

or:

```
gconman [-backstore]
```

Where `-backstore` causes graphical information to be stored locally to improve performance when windows need to be redrawn. Note that some X servers may not provide adequate support for this feature. If your windows are only partially redrawn, or images are missing when using the option, discontinue using it.

If you run **gmaestro**, on the Maestro Main Window, click the Console Manager button to open the Maestro Console Manager window opens.



Using the Console Manager Main Window

To exit Console Manager, select Quit from the File menu. Other actions related to the main window are described below.

Displaying Console Manager's Version

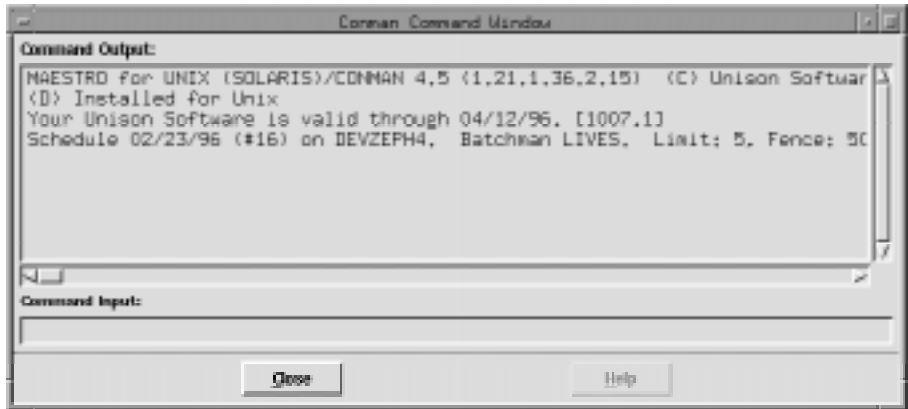
To display the Console Manager's version, select About... from the Help menu. To close the Version dialog, click OK.

Opening a SHOW Window for an Object

Click an object icon or select an object from the Objects menu to open the SHOW window for that object. There are SHOW windows for six Maestro objects: cpus, schedules, jobs, resource, prompts, and files. See [*SHOW Window Basics*](#) on page 6-21 for more information.

Issuing Command Line Commands

Maestro command line commands can be issued using the Command Window. Select Command Window... from the View menu to open the Command Window.

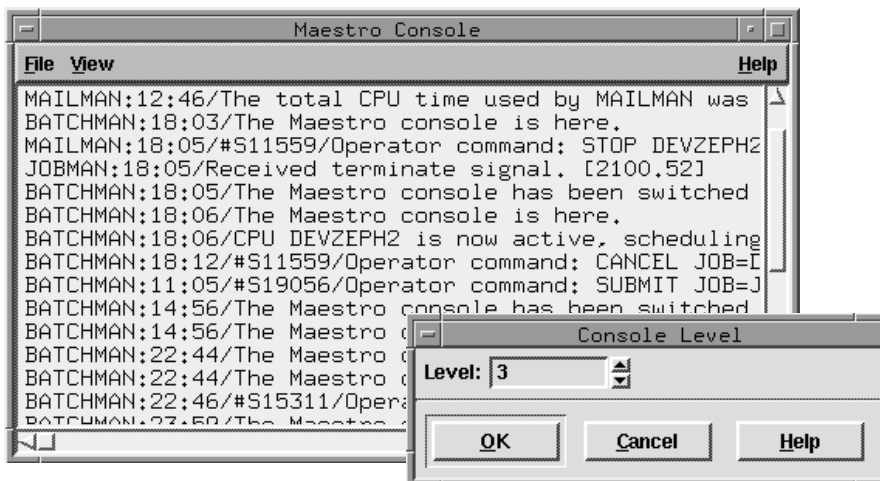


The Command Window has a display area for Console Manager output, labelled Command Output, and a text entry area for Console Manager commands, labelled Command Input. The output area echoes all commands and displays some status messages issued during the current run of Console Manager. The input area accepts Conman command line commands. Press the Return key after entering commands in Command Input to submit them to Conman. If the command has bad syntax or is invalid, a message appears in the output area. Click Close to close the window.

Enabling Allow Command Edit on SHOW windows opens the Command Window and writes commands to the Command Input area. See [Command Line Editing of Menu Actions](#) on page 6-23 for more information on the Allow Command Edit feature. For information on commands refer to section 9, [Conman Command Line Reference](#).

Displaying Messages in the Console

The Maestro Console displays Maestro messages. To open the Console window, select Console... from the View menu.



The level of messages the console receives is configurable. The levels are:

- 0 No messages (the default on fault-tolerant agent cpus).
- 1 Exception messages, such as operator prompts, and job abends.
- 2 Level 1, plus schedule successful messages.
- 3 Level 2, plus job successful messages (the default on the master domain manager).
- 4 Level 3, plus job launched messages.

To change the Console level, select Level... from the View menu. The Console Level dialog box opens. Enter the console message level (0-4) in the entry area by either typing it in or by using the up/down arrows. Click OK to change the message level and close the dialog. Click Cancel to close the dialog and retain the existing message level.

Displaying Maestro's Production Status

To view Maestro's status, select Status... from the View menu. A Status dialog opens displaying Maestro's version, the state of the Batchman program, and Maestro's job limit and fence. Click Close to close the dialog. Validation status is displayed in the Command Window when it is first opened and in the Maestro Main window.

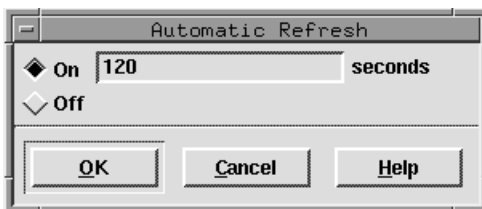
Refreshing All Console Manager Windows

Select Refresh All from the View menu on the Maestro Console Manager window to refresh the data in all open windows.

Note: When using Console Manager, you will enter commands to initiate certain actions, followed by commands to display the results. Depending on the level of system activity, the results of a previous command may not be immediately apparent. Therefore, it may be necessary to wait a moment to see results while the displays refresh. See [Setting the Refresh Rate](#) below for more information.

Setting the Refresh Rate

The refresh rate is the interval of time that Console Manager waits before updating displays. To open the Automatic Refresh dialog, select Auto Refresh... from the View menu .



To enable Auto Refresh, click the On button. To change the refresh rate, enter the desired interval of time (in seconds, the range is 15-43200) in the entry box. The default is 120 seconds. A warning appears if you enter an invalid number or no number. To disable Auto Refresh, click the Off button. If Off is selected, use Refresh All or Refresh Window from View menus to refresh the displays. Click OK to apply the new rate. Click Cancel to use the present rate.

Setting the refresh rate at a very short interval may affect performance. When displaying a large number of objects (for example, all the jobs in a large network), it may be necessary to increase the refresh time beyond the default of 120 seconds. This will insure that the entire display is written before the next refresh occurs. As an alternative, you can use a more restrictive filter to limit the number of objects, or turn off automatic refresh globally or for a specific display.

With Auto Refresh is On, a display may refresh immediately after you have highlighted an item. This may de-select the highlighted item. If this happens,

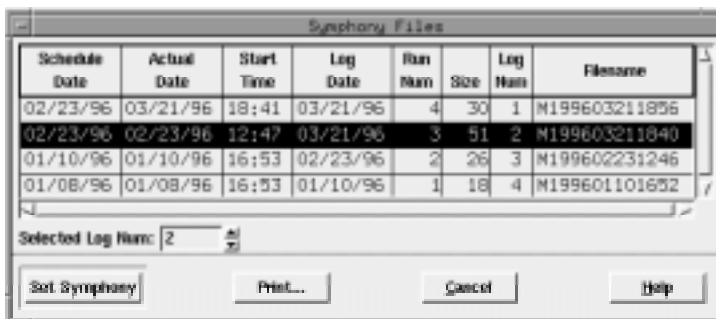
you must click on your selection a second time. Selecting the Stop Auto Refresh button from any window turns off Auto Refresh for that window.

Closing All Open Console Manager Windows

To close all open windows (except the Console Manager main window), select Close All from the View menu.

Changing the Symphony File

Console Manager windows display data from the currently selected Production Control (Symphony) file. When Console Manager is started, the most current Symphony file is automatically selected. To choose a different file, select Symphony... from the Options menu. The Symphony Files dialog opens displaying all available Symphony files.



Schedule Date	Actual Date	Start Time	Log Date	Run Num	Size	Log Num	Filename
02/23/96	03/21/96	18:41	03/21/96	4	30	1	M199603211856
02/23/96	02/23/96	12:47	03/21/96	3	51	2	M199603211840
01/10/96	01/10/96	16:53	02/23/96	2	26	3	M199602231246
01/08/96	01/08/96	16:53	01/10/96	1	18	4	M199601101652

Selected Log Num: 2

Sort Symphony Print... Cancel Help

The Symphony files are displayed in the following format:

Schedule Date The date used by the `schedule` command to select schedules for execution.

Actual Date The date Batchman began executing the Symphony file.

Start Time The time Batchman began executing the Symphony file.

Log Date The date the Symphony file was logged by the `stageman` command.

Run Num The run number assigned to the Symphony file. These are used internally for Maestro network synchronization.

Size The size of the log file in records.

Log Num The log number indicating the chronological order of log files. This number is used in the Selected Log Num field to select a specific log file.

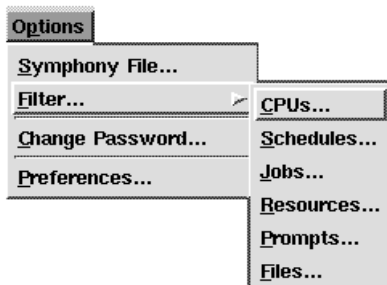
Filename The name of the log file assigned by the `stageman` command.

When you select a file in the list, the Log number appears in the Selected Log Num box. Enter a log number in the box and the corresponding file is selected in the list. The up/down arrow buttons increment/decrement the value by one.

Click Set Symphony to select the log file and close the dialog. All SHOW windows are refreshed and updated to reflect the information in the selected log file. If no value is specified, the current Symphony file is used. Click Cancel to close the dialog and disregard any selections.

Setting Object Filters

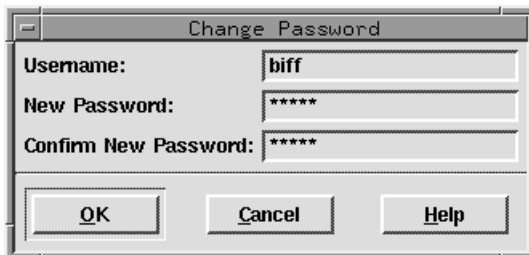
The data displayed in SHOW windows can be filtered using the object filters before any of the windows are opened. Once an object filter is set, the object's SHOW windows display data according to that filter. To open a filter dialog, select an object type from the cascading list displayed by selecting Filter... from the Options menu. The filter dialogs can also be opened from the object SHOW windows.



For detailed information on filters, see [Object Filters](#) on page 6-12.

Changing a User's Password

To change a user object's password, select Change Password... from the Options menu. The Change Password dialog opens.

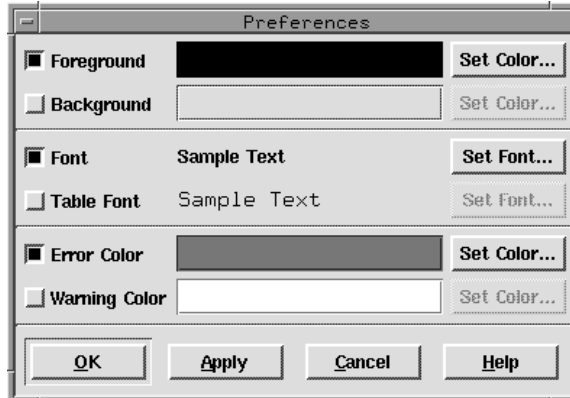


Enter the existing user's name in the Username field, the user's new password in the New Password field, and confirm the new password in the Confirm New Password field. Click OK to change the password and close the dialog. Click Cancel to close the dialog and retain the previous password.

Note: The Change Password action changes the password temporarily. The new password is stored only in the current Symphony file. To permanently change the password of a user object, use the Composer program. See *Defining Users* on page 4-54.

Configuring Preferences

This Preferences dialog permits you to set the colors and fonts for Console Manager displays. Select Preferences... from the Options menu to open the Preferences dialog.



The dialog is opened with the current preference settings displayed: colors are shown by a color block; fonts by “Sample Text”. The preferences are:

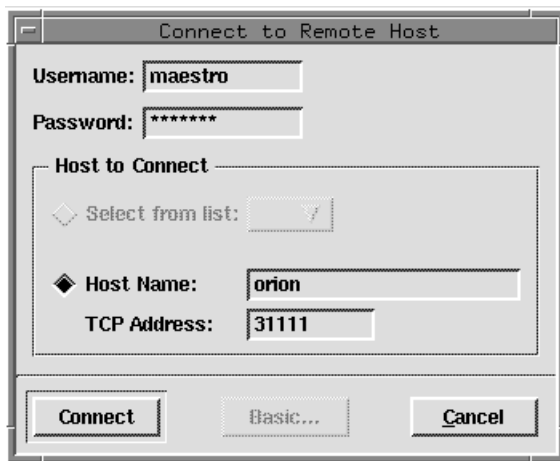
- | | |
|----------------------|--|
| Foreground | The color used to display text in displays. |
| Background | The color used for backgrounds in displays. |
| Font | The font used to display labels. |
| Table Font | The font used to display items in tables. The SHOWCPUS, SHOWSCHEDULES, and SHOWJOBS windows and the Dependent Objects dialogs have tables. |
| Error Color | The color used to highlight table cells that contain error conditions. The Error Color is used to indicate: <ul style="list-style-type: none"> • UNLINKED cpus. • Schedules in the ABEND and STUCK states. • Jobs in the ABEND, ABENP, and FAIL states. • Prompt dependencies in the NO state. • Expired Until times. |
| Warning Color | The color used to highlight table cells that contain warning conditions. The Warning color is used to indicate: <ul style="list-style-type: none"> • Opens dependencies in the NO state. • Jobs in the FENCE state. • Prompt dependencies in the ASKED or INACT state. • Needs Resources dependencies that are unavailable. |

To change a setting, click the button to the left of the setting label. (In the picture above, Set Color... is enabled for Foreground and Error Color, and Set Font... is enabled for Font.) Clicking Set Color... opens the Color Chooser dialog; clicking Set Font... opens the Font Chooser dialog. Choose a color or font from the appropriate dialog. To change a setting back to its default, deselect the button.

When you have completed configuring your preferences, click OK or Apply (OK also closes the dialog) to apply the settings. Click Cancel to close the dialog and retain the previous settings.

Connecting to a Remote Host

The Connect function is used to connect to a remote Console Manager (Conman) on another Maestro host. This is commonly used to connect to the master domain manager in another Maestro network. To open the Connect to Remote Host dialog, select Connect... from the File menu.



Username: Enter the user name that will be used to run the Console Manager on the remote Maestro host. This must be a valid user on the remote computer. If the remote computer is Windows NT, the user must be a member of its local Administrator group. The Maestro capabilities of the user are determined by the security set up on the remote Maestro host.

Password: Enter the user's password.

Select from list: Not used.

Host Name:	Click this button and enter the IP-resolvable host name of the remote Maestro cpu to which you wish to connect.
TCP Address:	Enter the TCP port number for connection to the remote Maestro cpu, usually 31111.
Connect	Click on this button to connect to the specified host and start the remote Console Manager.
Basic...	Not used.
Cancel	Click on this button to ignore your entries.

Valid Wildcards

The following wildcards can be used in Console Manager where permitted:

- @ Replaces one or more alphanumeric characters.
- ? Replaces one alphabetic character.
- % Replaces one numeric character.

Object Filters

Console Manager object filters are powerful tools that permit you to filter data in object SHOW windows. Configurable filter defaults permit you to customize your displays. There is a filter for each type of object SHOW window. Common attributes and details for each object filter are given below. Each filter is described in detail later in this section.

Common Filter Attributes

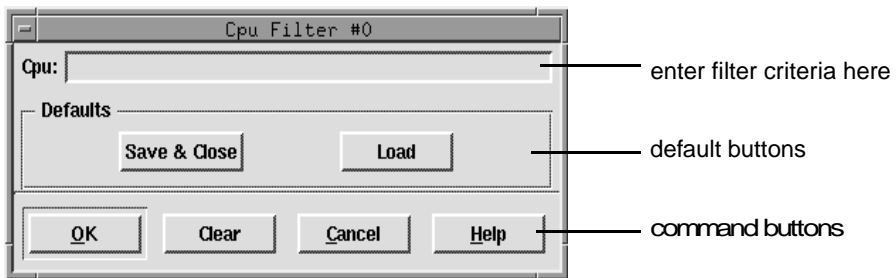
Each Filter dialog has common features and behavior.

Opening Object Filters

A Filter dialog can be opened in two ways:

- On the Console Manager window, select the Objects menu, then click Filter... to open a cascading menu. Select an object type from the menu to open the corresponding Filter dialog. Filters configured from here are applied to all subsequently opened SHOW windows. The title bar of a filter dialog opened from the Console Manager window has a zero (0) in it as in the picture below.
- On an object SHOW window, select Filter... from the View menu to open the corresponding object Filter dialog. Filter settings configured from here are applied to the parent SHOW window only. The number in the title bar of the filter dialog and SHOW number match.

The top portion of the Filter dialog is used to specify criteria for the filter. The bottom part provides buttons to save and load default settings and apply filters.



If no filter changes have occurred during a Console Manager session, default settings are used to open SHOW windows.

See [Saving a Filter Default](#) below for more information. Information on SHOW windows starts with [SHOW Window Basics](#) on page 6-21.

Saving a Filter Default

To save the current settings in any filter dialog as the default filter (and close the Filter dialog), click Save & Close. The filter setting is written to the appropriate configuration file, overwriting any previous filter default. Defaults are saved to the following files:

Cpu Filter	<code>maestrohome/.fcpu</code>
Schedule Filter	<code>maestrohome/.fsch</code>
Job Filter	<code>maestrohome/.fjob</code>
Prompt Filter	<code>maestrohome/.fprompt</code>
Resource Filter	<code>maestrohome/.fres</code>
File Filter	<code>maestrohome/.ffile</code>

Loading a Default Filter Setting

To load the current default settings for a filter, click Load. Entry fields and buttons will be set to the defaults.

Pre-filtering for a Console Manager Session

To set an object filter for an entire Console Manager session, open the Filter dialog from the Console Manager main window, set the filter, and click OK. The filter setting will be applied to any corresponding object SHOW window subsequently opened during that session.

Filtering a Single SHOW Window

To filter a specific object SHOW window, open the Filter dialog from the View menu on that SHOW window, set the filter, and click OK. Only that SHOW window will use that filter. This can be useful if you want to have two of the same kind of object SHOW windows open but you want them to display data based on different criteria.

Clearing All Fields

To blank all the fields in a Filter dialog, click Clear.

Retaining the Existing Filter

Click Cancel to close the Filter dialog and retain the existing filter.

Using Wildcards in Fields

Where permitted, wildcards can be used in filter entry fields. For valid wildcards, see [*Valid Wildcards*](#) on page 6-11.

Ad Hoc Scheduling

Using Console Manager's Submit dialogs, you can place jobs and schedules into the current production. You can submit schedules, jobs, script files, and commands. Schedules and jobs must already be documented by Maestro Composer to be submitted. Script files and commands do not have to be previously documented. Jobs, script files, and commands can be submitted into existing schedules or placed into the generic JOBS schedule.

Common Submit Dialog Attributes

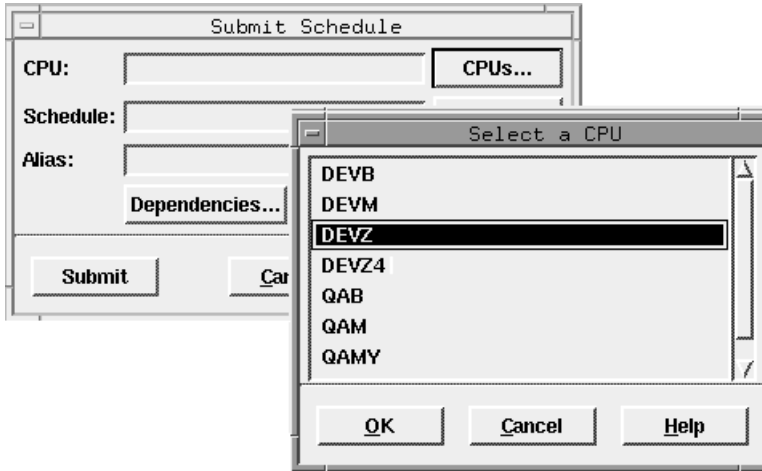
Submit dialogs have common features and behavior. Each submit dialog is described in detail later in this section.

Opening Submit Dialogs

A Submit dialog can be opened in two ways:

- From the Submit menu, select Schedule..., Job..., File..., or Command... to open the corresponding Submit dialog. For schedules, you are prompted to enter an existing schedule name. Jobs, script files, and commands submitted from the Console Manager window are placed in the generic JOBS schedule.
- Select a Submit option from the Actions menu on a SHOWSCHEDULES window. For Submit Schedule, the schedule selected in the display is submitted and no Submit dialog is opened. Jobs, script files, and commands are submitted into the schedule selected in the display. For more information on the SHOWSCHEDULES window, see [page 6-32](#).

The top portion of a Submit dialog has entry fields to specify what to submit. Some entry fields have associated buttons that open a list of available items for the entry field. For example, in the following Submit Schedule dialog, clicking the CPUs... button opens a Select a CPU dialog from which to choose valid cpus. The bottom part of the Submit dialog provides buttons to specify dependencies and submit the item.



Specifying Dependencies

To specify dependencies for the schedule, job, file, or command you want to submit, click the Dependencies... button. For schedules, the Add Schedule Dependencies window opens; for jobs, files, and commands, the Add Job Dependencies window opens. For instructions on how to use these windows, see [Adding Dependencies to Schedules](#) starting on page 6-64 and [Adding Dependencies to Jobs](#) starting on page 6-77.

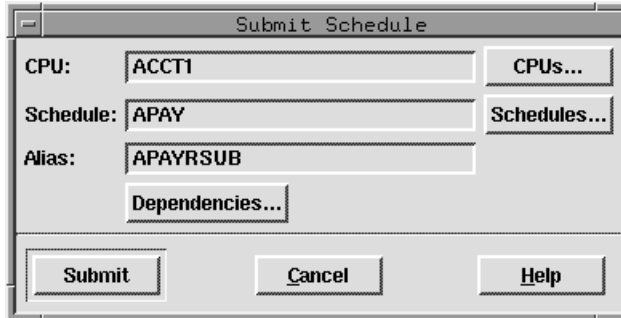
Submitting Production

After you have completed the Submit dialog and set any dependencies, click Submit to schedule the processing into the current production.

If you do not want to submit what you have specified, click Cancel.

Submitting Schedules

Use the Submit Schedule dialog to submit an existing schedule into the current production. A schedule is submitted with all of its defined dependencies and jobs. To submit a schedule, you must be running Console Manager on the master domain manager, or have access to the Maestro databases on the master domain manager.

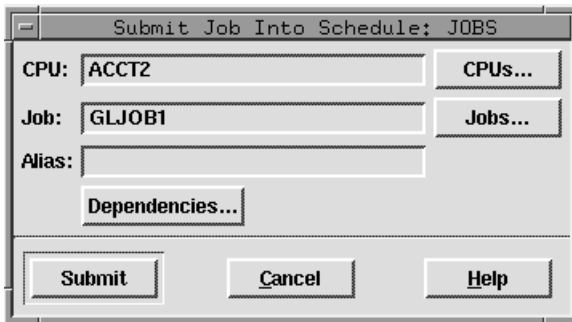


- CPU** Enter the name of the cpu or cpu class on which the schedule is defined. Wildcards are permitted, in which case, the schedule is launched on all qualifying cpus. If no entry is made, the default is the cpu running Console Manager. Click CPUs... to get a list of available cpus.
- Schedule** Enter the name of the schedule. The list opened from clicking Schedules... is filtered to display only those schedules defined for the CPU selection. Wildcards are permitted.
- Alias** If the schedule exists in the current production, enter a unique name (up to eight characters starting with a letter) to be assigned to the schedule. If no Alias is specified, Maestro generates one if necessary.

The command buttons and general Submit dialog behavior are discussed in *Common Submit Dialog Attributes* starting on page 6-15.

Submitting Jobs

Use the Submit Job dialog to submit a job into the current production. If the Submit Job dialog is opened from the Maestro Composer window, the job is inserted into the generic JOBS schedule (as shown in the picture below). If the Submit Job dialog is opened from the SHOWSCHEDULES window, the job is inserted into the schedule selected in the display and the schedule name appears in the title bar. To submit a job, you must be running Console Manager on the master domain manager, or have access to the Maestro databases on the master domain manager.

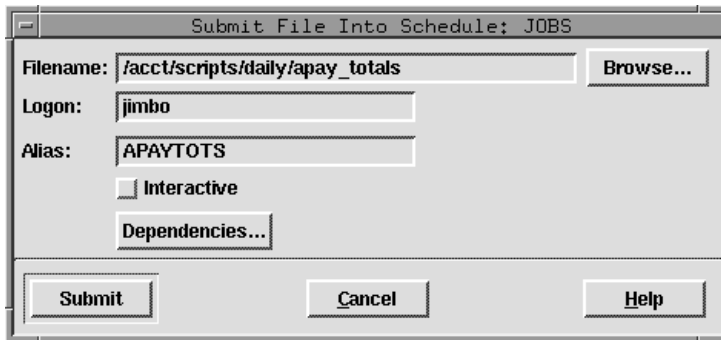


- CPU** Enter the name of the cpu on which the job is defined. Wildcards are permitted, in which case, the job is launched on all qualifying cpus. If no entry is made, the default is the cpu running Console Manager. You cannot submit a job to a cpu class.
- Job** Enter the name of the Maestro job. The list opened from clicking Jobs... is filtered to display only those jobs defined for the CPU selection. Wildcards are permitted.
- Alias** If the job is already in production, enter a unique name (up to eight characters starting with a letter) to be assigned to the job. If no Alias is specified, Maestro generates one if necessary.

The command buttons and general Submit dialog behavior are discussed in *Common Submit Dialog Attributes* starting on page 6-15.

Submitting Files

Use the Submit File dialog to submit a script file into the current production as a job. If the Submit File dialog is opened from the Maestro Composer main window, the job is inserted into the generic JOBS schedule (as shown in the picture below). If the Submit File dialog is opened from the SHOWSCHEDULES window, the job is inserted into the schedule selected in the display and the schedule name appears in the title bar.

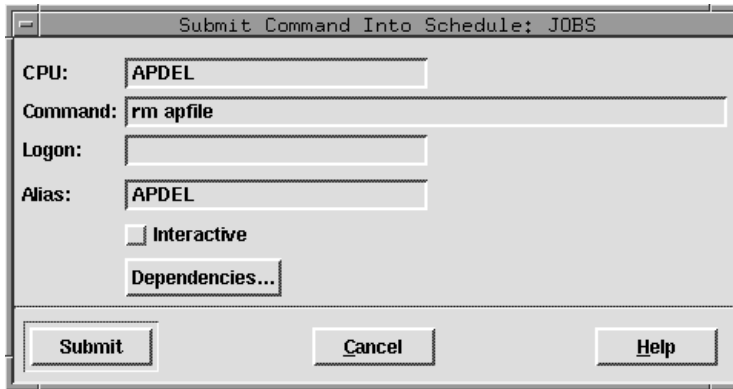


- Filename** Enter the path name of the script file (up to 255 characters). Wildcards are permitted.
- Logon** Enter the user name (up to eight characters) under which the job will run. For jobs that run on Windows NT, this user must have the right to "Log on as batch," and must have a "home path", which must exist, defined in its profile. If none is specified, the user running Console Manager is the default.
- Alias** Enter a unique name (up to eight characters starting with a letter) to be assigned to the job. If no Alias is specified, a job name is constructed using the first two alphanumeric characters of the file's basename followed by a six digit random number. If the Filename does not begin with a letter, you must supply a name.
- interactive** For Windows NT jobs, select this option to indicate that the job runs interactively on the Winfows NT desktop.

The command buttons and general Submit dialog behavior are discussed in *Common Submit Dialog Attributes* starting on page 6-15.

Submitting Commands

Use the Submit Command dialog to submit a command into the current production as a job. If the Submit Command dialog is opened from the Maestro Composer main window, the job is inserted into the generic JOBS schedule (as shown in the picture below). If the Submit Command dialog is opened from the SHOWSCHEDULES window, the job is inserted into the schedule selected in the display and the schedule name appears in the title bar.



CPU Enter the name of the cpu on which the job will run. Wildcards are permitted, in which case, the job is launched on all qualifying cpus. If no entry is made, the default is the cpu running Console Manager. You cannot submit a job to a cpu class.

Command Enter the command (up to 255 characters). Wildcards are permitted.

Logon Enter the user name (up to eight characters) under which the job will run. For jobs that run on Windows NT, this user must have the right to "Log on as batch," and must have a "home path", which must exist, defined in its profile. If none is specified, the user running Console Manager is the default.

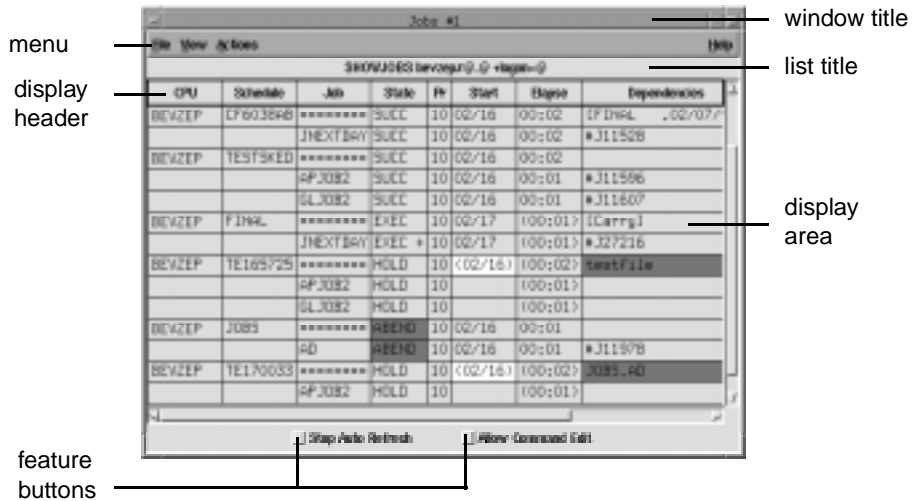
Alias Enter a unique name (up to eight characters starting with a letter) to be assigned to the job. If no Alias is specified, a job name is constructed using the first eight alphanumeric characters of the command.

interactive For Windows NT jobs, select this option to indicate that the job runs interactively on the Winfows NT desktop.

The command buttons and general Submit dialog behavior are discussed in *Common Submit Dialog Attributes* starting on page 6-15.

SHOW Window Basics

Console Manager SHOW windows display information about Maestro production. There is a SHOW window for each of six Maestro objects: cpus, schedules, jobs, prompts, resources, and files. All six SHOW windows share certain common elements: a display area, a menu bar, a list title bar, a display format header, and two window feature buttons. Each of the six SHOW windows is described in detail later in this section.



Opening and Closing a SHOW Window

When you click an icon or select an object from the Objects menu, the SHOW window for that object appears. Console Manager keeps track of how many SHOW windows for each object is opened in a session. The number is displayed in the window title bar. For example, the first SHOWJOBS window you open will have a "#1" in the title bar (as in the picture above) and the second will have a "#2".

To close a SHOW window, select Close Window from the File menu.

Reading the Display Area

The scrollable display area presents information about the selected object type. Above the display area, there is a format header with labels for each column. See the appropriate SHOW window section for the specific display

format for that type of object. Dependencies are listed only one per column but some rows may have multiple associated dependencies and multiple columns. Be sure to scroll all the way to right if you are looking for dependencies.

For SHOWCPUS, SHOWSCHEDULES, and SHOWJOBS windows, the display area is a dynamic table. The columns in the table can be resized by placing the cursor in the header column, pressing on the column border, and dragging the border to the desired width. When the cursor is over the column border, the pointer changes to a resizing tool.

Refreshing a SHOW Window

Windows are refreshed automatically, unless Stop Auto Refresh is selected. The auto refresh rate is defined on the Auto Refresh dialog. See [page 6-5](#). To disable Maestro's Auto Refresh feature for an individual SHOW window, select the Stop Auto Refresh toggle at the bottom of that window. You can select Refresh Now from the View menu to refresh the window on demand.

Opening an Object Filter Dialog

To open the filter dialog for the SHOW window, select Filter... from the View menu. Each SHOW window has a unique Filter dialog. For details on using the filters, see [Object Filters](#) on page 6-12.

Taking Action on Production Elements

Basic Console Manager operation involves clicking on an object in the display (one at a time) and selecting an action from the Actions menu. The Actions menu contains items that perform commands on the selected object in the display area. Some menu items bring up intermediate dialog boxes. The Actions menu for each SHOW window is unique. Each of the six SHOW windows is described in detail later in this section.

List Title

The list title bar is derived from the Maestro command line filter. It indicates the object type being displayed and the current filter selection for that window.

```
SHOWJOBS OCRA#@.@ +logon=maestro
```

In this example, the objects displayed are jobs in any schedule on cpu OCRA, with a job logon name of "maestro". For more information on filtering SHOW windows, see [Object Filters](#) on page 6-12.

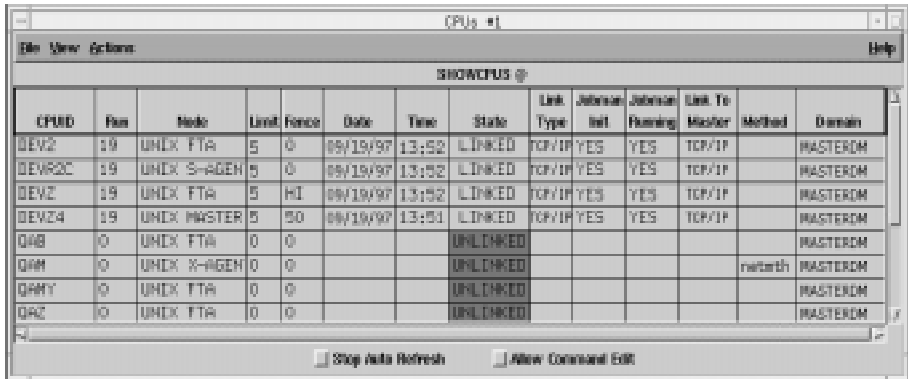
Command Line Editing of Menu Actions

Select the Allow Command Edit toggle at the bottom of the window to enable command line editing of the actions you select from the Actions menu. When enabled, a selection from the Actions menu opens the Command Window and places the Maestro command line equivalent of the action in the Command Input area of the window. If an intermediate dialog is launched from the Actions menu, you are taken to the Command Input area after completing that dialog. You can then edit the command and have it executed when you press the Return key. For a complete description of the Command Window see [Issuing Command Line Commands](#) on page 6-3. See section 9, [Conman Command Line Reference](#) for complete Console Manager command line syntax.

By default, the command line editing feature is disabled, and the actions you select from the Actions menu are executed immediately, or following completion of an intermediate dialog.

SHOWCPUS Window

The SHOWCPUS window displays cpu and cpu link information. Selecting a cell in the table causes the whole row to be selected. Only one row can be selected at a time.



Reading the Display Area

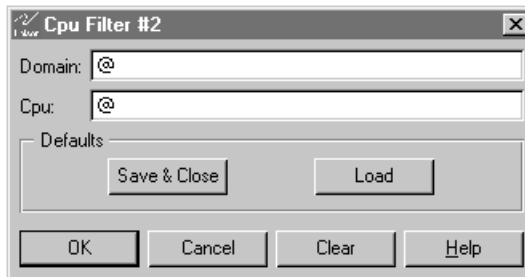
Cpu data is displayed in the table in the following format:

- CPUID** The name of the cpu to which this information applies.
- Run** The run number of the Symphony file.
- Node** The node type, and cpu type. Node types are: WNT, UNIX, MPEV, MPIX, and OTHER. Cpu types are: MASTER (master domain manager), MANAGER (domain manager), FTA (fault-tolerant agent), S-AGENT (standard agent) and X-AGENT (extended agent).
- Limit** The Maestro job limit for that cpu.
- Fence** The Maestro job fence for that cpu.
- Date** The date Batchman started executing the Symphony file.
- Time** The time Batchman started executing the Symphony file.
- State** The state of the cpu link from your login cpu to CPUID. LINKED indicates the cpus are linked. UNLINKED indicates the cpus are not linked. The UNLINKED state uses the Error color. Colors are selected in the Preferences dialog described on [page 6-9](#).

Link Type	The link type from your login cpu to CPUID. Link types are TCP/IP and DS. If the field is blank, the cpus are not linked.
Jobman Init	Indicates if Jobman is initialized. Either YES or NO.
Jobman Running	Indicates if Jobman is running. Either YES or NO.
Link to Master	The link type from the domain manager to CPUID. Link types are TCP/IP and DS. If the field is blank, the cpus are not linked.
Method	For extended agents only. The name of the access method specified in 40the cpu definition.
Domain	The name of the Maestro domain in which the cpu resides.

Filtering the Cpu Display

Complete the entry fields in the Cpu Filter dialog to set the filter for SHOWCPUS windows.



In the Cpu field, enter the Maestro cpu name string you want to use to filter the display. Wildcards are permitted. The command buttons and general Filter dialog behavior are discussed in [Common Filter Attributes](#) on page 6-12.

Viewing Schedules for a Cpu

To view the schedules that run on a cpu, select the cpu and then choose Schedules... from the Actions menu. A SHOWSCHEDULES window opens displaying the schedules that run on the selected cpu. See [SHOWSCHEDULES Window](#) on page 6-32 for more information.

Viewing Jobs for a Cpu

To view the jobs defined for a cpu, select the cpu and then choose Jobs... from the Actions menu. A SHOWJOBS window opens displaying the jobs defined for the selected cpu. See [SHOWJOBS Window](#) on page 6-41 for more information.

Viewing Resources for a Cpu

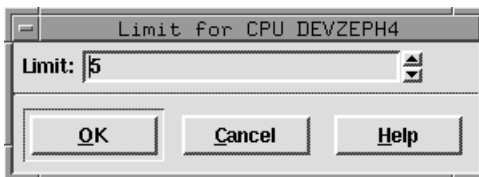
To view the resources defined for a cpu, select the cpu and then choose Resources... from the Actions menu. A SHOWRESOURCES window opens displaying the Needs resources defined for the selected cpu. See [SHOWRESOURCES Window](#) on page 6-56 for more information.

Viewing File Dependencies for a Cpu

To view the file dependencies defined for a cpu, select the cpu and then choose Files... from the Actions menu. A SHOWFILES window opens displaying the Opens file dependencies defined for the selected cpu. See [SHOWFILES Window](#) on page 6-62 for more information.

Changing a Cpu's Job Limit

To change a cpu's job limit, select the cpu and then choose Limit... from the Actions menu. The Limit for CPU dialog opens.



Enter a job limit (0-999) for the selected cpu. A limit of zero means that no jobs will launch.

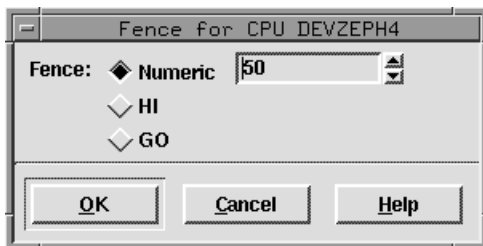
Click OK to change the job limit for the cpu and close the dialog. If the field is left blank and OK is clicked, a limit of zero is assigned. Click Cancel to close the dialog and retain the previous limit. The cpu job limit is carried forward during pre-production to the next day's Symphony file.

Maestro attempts to launch as many jobs as possible up to its cpu job limit. There is a practical limit to the number of processes that can be started on a given system. If the limit is reached, the system responds with a message

indicating that system resources are temporarily not available. When a Maestro job cannot be launched for this reason, the job enters the FAIL state. You can try lowering the cpu job limit to prevent this from occurring.

Changing a Cpu's Fence

To change a cpu's job fence, select the cpu and then choose Fence... from the Actions menu. The Fence for CPU dialog opens. Click Numeric, HI or GO. The Numeric range is 0-99. HI is equivalent to 100 and GO is equivalent to 101. If the fence is set to HI, only those jobs with a priority of GO can be launched. If the fence is set to GO, no jobs can be launched.



Click OK to apply the new cpu job fence and close the dialog. If Numeric is selected without a number in the entry field, zero is set as the cpu job fence. Click Cancel to close the dialog and retain the previous job fence. The cpu job fence is initially set to zero following installation, and the current setting is always carried forward during pre-production processing to the next day's Symphony file.

Maestro's job fence prevents low priority jobs from being launched, regardless of the priorities of their schedules. It is possible, therefore, to hold back low priority jobs in high priority schedules, while allowing high priority jobs in low priority schedules to be launched. Jobs will not be launched on the cpu if their priorities are less than or equal to the fence.

Starting and Stopping a Cpu

To start Maestro's production processes, select a cpu and then choose **Start** from the Actions menu. On a network cpu, **Start** must be executed locally by the **maestro** user to start Netman. When **Start** is executed following pre-production processing in a network, autolinked agents and domain managers are initialized and started. Cpus that are not autolinked are initialized and started when you execute the first **Link** action for them.

To stop Maestro's production processes, select a cpu and then choose **Stop** from the Actions menu. To stop Netman, issue a `shutdown` command from

the Conman command line (see *shutdown* on page 9-95). When **Stop** is executed from a domain manager to an agent or subordinate domain manager, the cpu is stopped even if it is unlinked.

For more information about starting and stopping cpus, refer to section A, *Maestro Networks*.

Linking and Unlinking a Cpu

To open a link, select the cpu and then choose **Link** from the Actions menu. To close a link, select the cpu and then choose **Unlink** from the Actions menu.

For more information about linking and unlinking cpus, refer to section A, *Maestro Networks*.

SHOWDOMAINS Window

The SHOWDOMAINS window displays information about Maestro domains. Selecting a cell in the table causes the whole row to be selected. Only one row can be selected at a time.



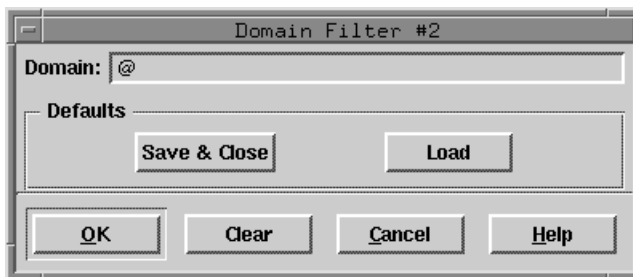
Reading the Display Area

Domain information is displayed as follows:

Domain	The name of the domain to which this information applies.
Manager	The name of the domain manager cpu.
Parent	The name of the parent domain.

Filtering the Domain Display

To open the filter dialog, select **Filter...** from the View menu.



In the Domain field, enter the character string you want to use to filter the display. Wildcards are permitted; for example, **r@** selects domains with names starting with the letter R. The command buttons and general Filter dialog behavior are discussed in *Common Filter Attributes* on page 6-12.

Starting and Stopping Cpus

To start Maestro's production processes in a domain, select the domain and then choose **Start CPUs** from the Actions menu. When cpus are started following pre-production processing in a network, autolinked agents and domain managers are intialized and started. Cpus that are not autolinked are initialized and started when you execute the first **Link** action for them.

To stop Maestro's production processes in a domain, select the domain and then choose **Stop CPUs** from the Actions menu.

For information about starting, stopping, and shutting down individual cpus, see *Starting and Stopping a Cpu* on page 6-27. For more information about starting and stopping cpus, refer to section A, *Maestro Networks*.

Linking and Unlinking Cpus

To open the links in a domain, select the domain and then choose **Link CPUs** from the Actions menu. To close the links, select the domain and then choose **Unlink CPUs** from the Actions menu.

For information about linking and unlinking individual cpus, see *Linking and Unlinking a Cpu* on page 6-28. For more information about linking and unlinking, refer to section A, *Maestro Networks*.

Switching the Domain Manager

To switch the domain manager to another cpu, select the domain and then choose **Switch Manager...** from the Actions menu.



In the Domain Manager field, enter the cpu name of the backup domain manager that will become the new domain manager, or select the name from the list of cpus produced by clicking the **CPUs...** button. The backup domain manager must be in the same domain as the current domain manager, and it must be running Maestro 6.0 or later. For more information about switching domain managers, refer to section A, *Maestro Networks*.

SHOWSCHEDULES Window

The SHOWSCHEDULES window displays schedule information. Selecting a cell in the table, other than Dependencies and Start cells, causes the whole row to be selected. Dependencies and Start cells can be selected individually. Only one row can be selected at a time.



Reading the Display Area

Schedule data is displayed in the table in the following format:

- CPU** The cpu on which the schedule runs.
- Schedule** The name of the schedule.
- State** The state of the schedule. Some states are also indicated by a background color in the table cell. Colors are selected in the Preferences dialog (described on page 6-9). Schedule States (and associated color) are:
 - ABEND** Schedule terminated unsuccessfully (Error color).
 - ADD** Schedule has just been submitted.
 - EXEC** Schedule is executing.
 - HOLD** Schedule is awaiting dependency resolution.
 - READY** Dependencies resolved, schedule is ready to launch.
 - STUCK** Execution interrupted (Error color). No jobs will be launched without operator intervention.
 - SUCC** Schedule completed successfully.

Pr	The priority of the schedule.
Start	The scheduled start time. If the time is in parenthesis (Warning color), it is the estimated start time (that is, the At time). If the time is more than 24 hours in the past or future, the date is listed instead of the time.
Elapse	The schedule's run time. If it is in parenthesis, it is an estimation based on logged statistics.
# Jobs	The number of jobs in the schedule.
OK Jobs	The number of jobs in the schedule that have run successfully.
Limit	The schedule's job limit. If blank, no limit is in effect.
Dependencies	<p>Schedule dependencies are listed one per column, therefore, there can be multiple Dependency columns for one row. Some dependencies use colors to highlight conditions. Colors are selected in the Preferences dialog (described on page 6-9). Comments can also appear in a Dependencies column. Any of the following can be listed:</p> <ul style="list-style-type: none"> • For a Follows dependency, a schedule or job name. If a followed schedule or job is in a state that uses a Warning or Error color, that color is indicated in the cell. • For an Opens file dependency, the file name. If the file does not exist, the Warning color is indicated. If the file resides on an x-agent and its name is longer than 28 characters, it is displayed as follows: <pre>..last_25_characters_of_filename</pre> • For a Needs resources dependency, a resource name enclosed in hyphens (-). If the number of units requested is greater than one, the number is displayed before the first hyphen. If the resource is not available, the Warning color is indicated. • Schedules whose Until times have expired and schedules cancelled with the Pending option whose Until times have expired are labeled: [Until]. The Error color is also indicated. • For an Until time, the time preceded by a less than sign (<). • For a Prompt dependency, the prompt number displayed as: #<i>num</i>. For global prompts, this is followed by the

prompt name in parentheses. If the prompt has not been responded to, the Error color is indicated. If the prompt has been asked or is yet to be issued (INACT state), the Warning color is used

- Cancelled schedules are labeled: [Cancelled].
- Schedules cancelled with Pending option are labeled: [Cancel Pend].
- Schedules that are flagged as Carryforward are labeled: [Carry].
- For schedules that were carried forward from the previous day, the original schedule name and original schedule date are displayed in brackets ([]).

Filtering the Schedule Display

Complete the entry fields in the Schedule Filter dialog to set the filter for SHOWSCHEDULES windows.

The screenshot shows the 'Schedule Filter #0' dialog box. It has a title bar and a standard Windows-style interface. The main area is divided into several sections, each with a small icon on the left. The 'Started:' section is currently active, indicated by a small square icon. The 'Finished:' section is inactive. The 'Priority:' section is also inactive. The 'Follows:', 'Needs:', 'Opens:', and 'Prompts:' sections are inactive. The 'Defaults' section is at the bottom of the main area and contains two buttons: 'Save & Close' and 'Load'. At the very bottom of the dialog are four buttons: 'OK', 'Clear', 'Cancel', and 'Help'.

To enable most entry fields, the toggle button to the far left of the data panel must be selected. For example, in the picture above, the Started data panel is

enabled and the Finished data panel is not. If the data panel is not enabled, its controls are dimmed and the corresponding criteria are not used in the filter.

Cpu To qualify schedules by the cpu in which they are defined, enter the Maestro cpu name string. Wildcards are permitted.

Schedule To qualify schedules by name, enter the schedule name string. Wildcards are permitted.

State To qualify schedules by state, select one or more schedule State buttons. For a description of the schedules states, see [page 6-32](#).

Started Use the pull-down menu to qualify schedules based on whether or not they have started executing.

Select Anytime from the menu to qualify all schedules that have started.

Select Exact from the menu to specify an exact execution start time. When Exact is selected, the Time and Date fields below appear in the data panel.

Started: Exact ▾ Time: 22:45 ▲▼ Date: 05/30/96

Enter the exact time, in 24-hour format, in the Time box. The arrow keys change the time in 15 minute increments. Enter the date, in *mm/dd/yy* format, in the Date box.

Select Range from the menu to specify a window for execution start time. When Range is selected, the Time and Date fields below appear in the data panel.

Started: Range ▾
 Low Time: 01:30 ▲▼ Date: 05/30/96
 High Time: _: : ▲▼ Date:

To specify the bottom of the range, click the Low Time button and enter a Low Time and Date in the same manner as for Exact time. Click the High Time button and enter a High Time and Date to specify the top of the range. If Low Time is used without High Time, schedules are selected that started at or after the Low Time. If High Time is used without Low Time, schedules are selected that started at or before the High Time.

If the Started toggle is selected but no time is specified, schedules are qualified if they have started executing.

Finished In the same manner as selecting Started times, use this data panel to qualify schedules based on whether or not they have finished executing. If the Finished toggle is selected but no time is specified, schedules are qualified if they have finished executing.

Priority Use the pull-down menu to qualify schedules based on an exact priority or a priority range.

Select Exact from the menu to specify an exact priority value. When Exact is selected the entry fields below appear in the data panel.



A screenshot of a software interface for filtering schedules. It features a 'Priority' label followed by a dropdown menu set to 'Exact'. To the right is a text input field containing the number '100', with small up and down arrow buttons on its right side. Further right are three buttons labeled 'HOLD', 'HI', and 'GO'.

Enter a number (0-101) or click HOLD, HI, or GO. HOLD is equivalent to a priority of zero, HI is equivalent to a priority of 100, and GO is equivalent to a priority of 101. The arrow buttons change the value by increments of one.

Select Range from the menu to specify a priority range. When Range is selected the entry fields below appear in the data panel.



A screenshot of a software interface for filtering schedules. It features a 'Priority' label followed by a dropdown menu set to 'Range'. To the right are two input fields: 'Low' with the value '50' and 'High' with the value '99'. Each input field has small up and down arrow buttons on its right side.

To specify the bottom of the range, click the Low button and enter an integer (0-101). To specify the top of the range, click the High button and enter an integer (0-101). If Low is used without High, schedules are selected with priorities equal to or greater than the Low value. If High is used without Low, schedules are selected with priorities equal to or less than the High value.

Follows To qualify schedules based on whether or not they have a Follows dependency, enter a prerequisite job or schedule. In the CPU field, enter the name of the cpu on which the prerequisite schedule runs. In the Schedule field, enter the name of the prerequisite schedule. In the Job field, enter the name of the prerequisite job. Wildcards are permitted in all three fields. If the Follows toggle is selected but no job or schedule is specified, schedules are qualified if they have any Follows dependencies.

- Needs** To qualify schedules based on whether or not they have a Needs resource dependency, enter the dependent resource. In the CPU field, enter the name of the cpu on which the resource is defined. In the Resource field, enter the name of the resource. Wildcards are permitted in both fields. If the Needs toggle is selected but no resource is specified, schedules are qualified if they have any Needs resource dependencies.
- Opens** To qualify schedules based on whether or not they have an Opens file dependency, enter the dependent file. In the CPU field, enter the name of the cpu on which the file exists. In the FileName field, enter the path name of the file. In the Qualifier field, enter the file's Qualifier String. Wildcards are permitted in the CPU and FileName fields. If the Opens toggle is selected but no file is specified, schedules are qualified if they have any Opens file dependencies.
- Prompts** To qualify schedules based on whether or not they have a Prompt dependency, enter either the prompt name or number. For global prompts, enter the name of the prompt in the Prompt Name box and click the corresponding radio button. For local prompts, enter the number of the prompt in the Prompt Number box and click the corresponding radio button. Wildcards are permitted in the Prompt Name field only. If the Prompt toggle is selected but no prompt is specified, schedules are qualified if they have any Prompt dependencies.

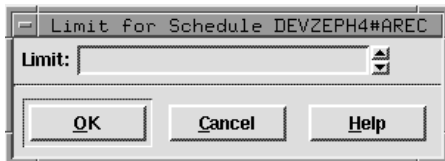
The command buttons and general Filter dialog behavior are discussed in [Common Filter Attributes](#) on page 6-12.

Viewing the Jobs for a Schedule

To view the jobs defined for a schedule, select the schedule and then choose Jobs... from the Actions menu. A SHOWJOBS window opens displaying the jobs defined for the selected schedule. See [SHOWJOBS Window](#) on page 6-41 for more information.

Changing a Schedule's Job Limit

To change a schedule's job limit, select the schedule and then choose Limit... from the Actions menu. The Limit for Schedule dialog opens.

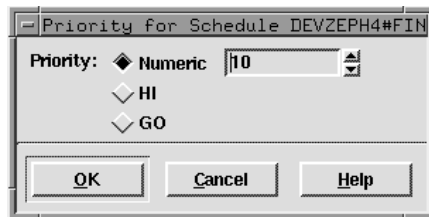


Enter a job limit (0-999) for the selected schedule. A limit of zero means that no jobs will launch from the schedule. Click OK to change the job limit for the schedule and close the dialog. If the field is left blank and OK is clicked, a limit of zero is assigned. Click Cancel to close the dialog and retain the previous limit. The schedule job limit is carried forward during pre-production to the next day's Symphony file.

Maestro attempts to launch as many jobs as possible up to each schedule's job limit. For more information, see [Changing a Cpu's Job Limit](#) on page 6-26.

Changing a Schedule's Priority

To change a schedule's priority, select the schedule and then choose Priority... from the Actions menu. The Priority for Schedule dialog opens.



Click Numeric, HI or GO. The Numeric range is 0-99. HI is equivalent to 100 and GO is equivalent to 101. In the case of HI and GO, all jobs in the schedule are given HI or GO priority. HI and GO jobs are launched as soon as their dependencies are satisfied. They override the cpu job limit, but they do not override the schedule's job limit or Maestro's job fence.

Click OK to apply the new priority for the schedule and close the dialog. If you do not specify a priority level (that is, Numeric is selected but no number is in the entry field) and click OK, the schedule is assigned a priority of zero. Click Cancel to close the dialog and retain the schedule's previous priority. The priority is carried forward during pre-production processing to the next day's Symphony file.

Releasing a Schedule from its Dependencies

Dependencies can be released from a schedule individually or all at once. To release a schedule from all of its dependencies, select the entire schedule row and then choose Release from the Actions menu. To release a schedule from a single dependency, select the appropriate Dependencies or Start cell (not the entire row) and then choose Release from the Actions menu. Either way, a dialog opens prompting you for confirmation to release the schedule. Click Yes to release the schedule or No to take no action. For Needs resources dependencies, the released schedule is given the required number of units of the resource, even though they may not be available. This may cause the Available column of the SHOWRESOURCES window to display a negative number.

Canceling a Schedule

To cancel a schedule, select the schedule and then choose Cancel from the Actions menu. A dialog opens prompting you for confirmation to cancel the schedule. Click Yes to cancel the schedule or No to take no action.

If a schedule is cancelled before it is launched, it will not launch. If cancelled after it is launched, the schedule continues to execute. In either case, jobs and schedules that are dependent on the cancelled schedule or any of the jobs within the cancelled schedule are released.

Adding Dependencies to a Schedule

To add dependencies to an existing schedule, select the schedule and then choose Add Dependencies... from the Actions menu. The Add Schedule Dependencies window opens. For instructions on how to use this window see [*Adding Dependencies to Schedules*](#) on page 6-64.

Deleting Dependencies from a Schedule

To delete a dependency from a schedule, select the appropriate Dependencies or Start cell (not the entire row) and then choose Delete Dependency from the Actions menu. A dialog opens prompting you for confirmation to delete the dependency from the schedule. Click Yes to delete the dependency or No to take no action.

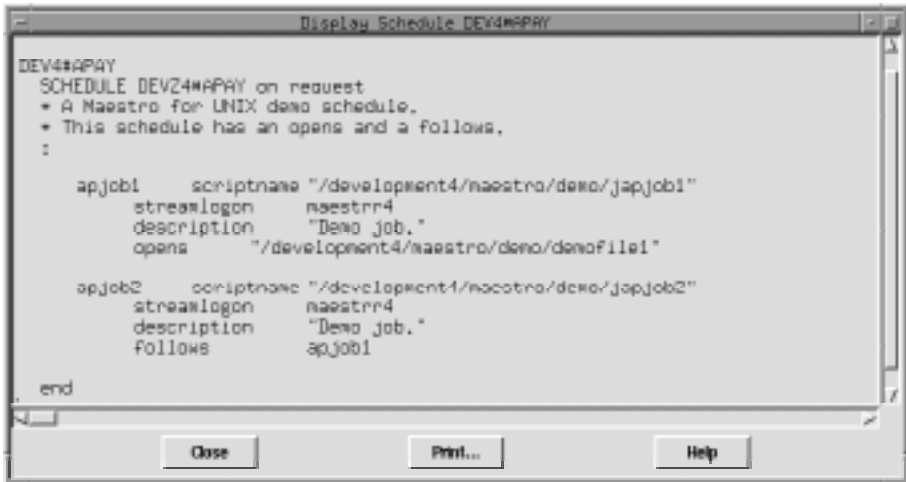
Submitting Ad Hoc Production

To submit a schedule into production, select the schedule and then choose Submit Schedule... from the Actions menu. The schedule is immediately submitted and assigned an alias. To submit a schedule that is not in production or supply your own alias, use the Submit Schedule... option on the Maestro Conman main window, described in [Ad Hoc Scheduling](#) on page 6-15.

To submit jobs, script files, and commands into the selected schedule, select the appropriate Submit option from the Actions menu. A Submit dialog opens with the name of the schedule in the title bar. For instructions on how to use the Submit dialogs, refer to [Ad Hoc Scheduling](#) on page 6-15.

Viewing the Scheduling Language

To view the command line scheduling language version of a schedule, select the schedule and then choose Display... from the Actions menu. The Display Schedule dialog opens. The schedule is presented in Maestro's scheduling language and you can not edit the text. For more information on the scheduling language see [The Scheduling Language](#) on page 8-50.



Click Close to close the dialog or Print... to open the Print dialog.

SHOWJOBS Window

The SHOWJOBS window displays job information. Selecting a cell in the table, other than Dependencies and Start cells, causes the whole row to be selected. Dependencies and Start cells can be selected individually. Only one row can be selected at a time.

CPU	Schedule	Job	State	Pp	Start	Elapse	Dependencies
BEVZEP	CF6038AB	*****	SUCC	10	02/16	00:02	[FINAL ,02/07/
		JNEXTDAY	SUCC	10	02/16	00:02	#J11528
BEVZEP	TESTSKED	*****	SUCC	10	02/16	00:02	
		APJOB2	SUCC	10	02/16	00:01	#J11596
		GLJOB2	SUCC	10	02/16	00:01	#J11607
BEVZEP	FINAL	*****	EXEC	10	02/17	(00:01)	[Carry]
		JNEXTDAY	EXEC +	10	02/17	(00:01)	#J27216
BEVZEP	TE165725	*****	HOLD	10	(02/16)	(00:02)	testfile
		APJOB2	HOLD	10		(00:01)	
		GLJOB2	HOLD	10		(00:01)	
BEVZEP	JOBS	*****	ABEND	10	02/16	00:01	
		AD	ABEND	10	02/16	00:01	#J11978
BEVZEP	TE170033	*****	HOLD	10	(02/16)	(00:02)	JOBS.AD
		APJOB2	HOLD	10		(00:01)	

Reading the Display Area

Jobs are listed in the table below the schedules to which they belong. Data is displayed in the table in the following format:

CPU The cpu on which the job's schedule runs.

Schedule The name of the job's schedule.

Job The name of the job. The following notation may precede a job name:

- >> **rerun as** Job rerun with Rerun command, or as a result of automatic recovery.
- >> **rerun step** Job rerun with `rerun;step` command.
- >> **every run** The second and subsequent runs of an Every job.
- >> **recovery** The run of a recovery job.
- >> **rec (cpu#)** The run of a recovery job on a different cpu.

State

The state of the job or schedule. Some states are also indicated by a background color in the table cell. Colors are selected in the Preferences dialog described on [page 6-9](#).

Job states (and associated color) are:

ABEND	Job terminated with a non-zero exit code (Error color).
ABENP	ABEND confirmation received, but job not completed (Error color).
ADD	Job is being submitted.
CANCL	Job cancelled.
DONE	Job completed in an unknown state.
ERROR	(Internetwork dependencies only) An error occurred while checking for the remote status.
EXEC	Job is executing.
EXTRN	(Internetwork dependencies only) Unknown status. An error occurred, a Rerun action was just performed on the EXTERNAL job, or the remote job or schedule does not exist.
FAIL	Unable to launch job (Error color).
FENCE	Job's priority is below fence (Warning color).
HOLD	Job awaiting dependency resolution.
INTRO	Job introduced for launching by the system.
PEND	Job completed, awaiting confirmation.
READY	Job ready to launch, all dependencies resolved.
SCHED	Job's scheduled start time has not arrived.
SUCC	Job completed with zero exit code.
SUCCP	SUCC confirmation received, but job not completed.
SUSP	(MPE only) Job suspended by breakjob command.
WAIT	(Extended agent jobs and MPE only) Job is in the wait state.
WAITD	(MPE only) Job is in the wait state, and is deferred.

Schedule States (and associated color) are:

ABEND	Schedule terminated unsuccessfully (Error color).
ADD	Schedule has just been submitted.
CANCL	Schedule cancelled.
EXEC	Schedule is executing.
HOLD	Schedule is awaiting dependency resolution.

	READY	Dependencies resolved, schedule is ready to launch.
	STUCK	Execution interrupted (Error color). No jobs will be launched without operator intervention.
	SUCC	Schedule completed successfully.
Pr		The priority of the job or schedule. A plus sign (+) preceding the priority means the job has been launched.
Start		The scheduled start time of the job or schedule. If it is in parenthesis (Warning color), it is the estimated start time (that is, the At time). If the time is more than 24 hours in the past or future, the date is listed instead of the time.
Elapse		The schedule's or job's run time. If it is in parenthesis, it is an estimation based on logged statistics.
Dependencies		<p>Job and schedule dependencies are listed one per column, therefore, there can be multiple Dependencies columns for one row. Some dependencies use colors to highlight conditions. Colors are selected in the Preferences dialog (described on page 6-9). Comments can also appear in the Dependencies column but have no color associated with them. Any of the following can be listed:</p> <ul style="list-style-type: none"> • For a Follows dependency, a schedule or job name. If a followed schedule or job is in a state that uses a Warning or Error color, that color is indicated in the cell. • For an Opens file dependency, the file name. If the file does not exist, the Warning color is indicated. If the file resides on an x-agent and its name is longer than 28 characters, it is displayed as follows: <pre style="margin-left: 40px;">..last_25_characters_of_filename</pre> • For a Needs resources dependency, a resource name enclosed in hyphens (-). If the number of units requested is greater than one, the number is displayed before the first hyphen. If the resource is not available, the Warning color is indicated. • Jobs and schedules whose Until times have expired and jobs and schedules cancelled with the Pending option whose Until times have expired are labeled: [Until]. The Error color is also indicated. • For an Until time, the time preceded by a less than sign (<).

- For a Prompt dependency, the prompt number displayed as: `#num`. For global prompts, this is followed by the prompt name in parentheses. If the prompt has not been responded to, the Error color is indicated. If the prompt has been asked or is yet to be issued (INACT state), the Warning color is used
- For an Every rate, the repetition rate preceded by an ampersand (&).
- For executing jobs, the process id (PID). Shown as: `#Jnnnnn`.
- Jobs submitted using Maestro's `at` and `batch` commands are labeled: [Userjcl].
- Cancelled jobs and schedules are labeled: [Cancelled].
- Jobs and schedules cancelled with the pending option are labeled: [Cancel Pend].
- [Recovery] means that operator intervention is required.
- [Confirm] means that confirmation is required because the job was scheduled using the Needs Confirmation.
- Schedules that are flagged as Carryforward are labeled: [Carry].
- For schedules that were carried forward from the previous day, the original schedule name and original schedule date are displayed in brackets ([]).

Filtering the Job Display

Complete the entry fields in the Job Filter dialog to set the filter for SHOWJOBS windows.

To enable most entry fields, the toggle button to the far left of the data panel must be selected. For example, in the picture above, the Started data panel is enabled and the Finished data panel is not. If the data panel is not enabled, its controls are dimmed and the corresponding criteria are not used in the filter.

- Cpu** To qualify jobs by the schedule's cpu, enter the Maestro cpu name string. Wildcards are permitted.
- Schedule** To qualify jobs by the schedule in which they run, enter the schedule name string. Wildcards are permitted.
- Job** To qualify jobs by job name, enter the job name string. Wildcards are permitted.
- State** To qualify jobs by state, select one or more job State buttons. For a description of the job states, see [page 6-42](#).
- Login** To qualify jobs by the Login specified in the Maestro job definition, enter the login name string. Wildcards are permitted.

Script Files

To qualify jobs by their Script File, enter the path name of the script file. Wildcards are permitted.

Started

Use the pull-down menu to qualify jobs based on whether or not they have started executing.

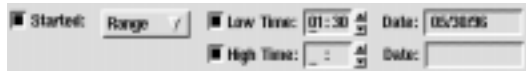
Select Anytime from the menu to qualify all jobs that have started.

Select Exact from the menu to specify an exact execution start time. When Exact is selected, time and date fields appear in the data panel.



Enter the exact time, in 24-hour format, in the Time box. The arrow keys change the time in 15 minute increments. Enter the date, in *mm/dd/yy* format, in the Date box.

Select Range from the menu to specify a window for execution start time. When Range is selected, the time and date fields below appear in the data panel.



To specify the bottom of the range, click the Low Time button and enter a Low Time and Date in the same manner as for Exact time. Click the High Time button and enter a High Time and Date to specify the top of the range. If Low Time is used without High Time, jobs are selected that started at or after the Low Time. If High Time is used without Low Time, jobs are selected that started at or before the High Time.

If the Started toggle is selected but no time is specified, jobs are qualified if they have started executing.

Finished

In the same manner as selecting Started times, use this data panel to qualify jobs based on whether or not they have finished executing. If the Finished toggle is selected but no time is specified, jobs are qualified if they have finished executing.

Priority

Use the pull-down menu to qualify jobs based on an exact priority or a priority range.

When Exact is selected from the menu, the entry fields below appear in the data panel.

The screenshot shows a control panel for job filtering. It features a 'Priority:' label followed by a dropdown menu currently set to 'Exact'. To the right of the dropdown is a numeric input field containing the value '100'. Further right are three buttons: 'HOLD', 'HI', and 'GO'. Small up and down arrow icons are positioned between the numeric field and the 'HOLD' button, and between the 'HI' and 'GO' buttons.

Enter a number (0-101) or click HOLD, HI, or GO. HOLD is equivalent to a priority of zero, HI is equivalent to a priority of 100, and GO is equivalent to a priority of 101. The arrow buttons change the value by increments of one.

When Range is selected from the menu, the entry fields below appear in the data panel.

The screenshot shows the 'Priority:' control panel with 'Range' selected in the dropdown menu. To the right of the dropdown are two numeric input fields: 'Low' with the value '50' and 'High' with the value '99'. Each numeric field has small up and down arrow icons to its right.

To specify the bottom of the range, click the Low button and enter an integer (0-101). To specify the top of the range, click the High button and enter an integer (0-101). If Low is used without High, jobs are selected with priorities equal to or greater than the Low value. If High is used without Low, jobs are selected with priorities equal to or less than the High value.

Follows

To qualify jobs based on whether or not they have a Follows dependency, enter a prerequisite job or schedule. In the CPU field, enter the name of the cpu on which the prerequisite schedule runs. In the Schedule field, enter the name of the prerequisite schedule. In the Job field, enter the name of the prerequisite job. Wildcards are permitted in all three fields. If the Follows toggle is selected but no job or schedule is specified, jobs are qualified if they have any Follows dependencies.

Needs

To qualify jobs based on whether or not they have a Needs resource dependency, enter the dependent resource. In the CPU field, enter the name of the cpu on which the resource is defined. In the Resource field, enter the name of the resource. Wildcards are permitted in both fields. If the Needs toggle is selected but no resource is specified, jobs are qualified if they have any Needs resource dependencies.

Opens

To qualify jobs based on whether or not they have an Opens file dependency, enter the dependent file. In the CPU field, enter the name of the cpu on which the file

exists. In the FileName field, enter the path name of the file. In the Qualifier field, enter the file's Qualifier String. Wildcards are permitted in the CPU and FileName fields. If the Opens toggle is selected but no file is specified, jobs are qualified if they have any Opens file dependencies.

Prompts

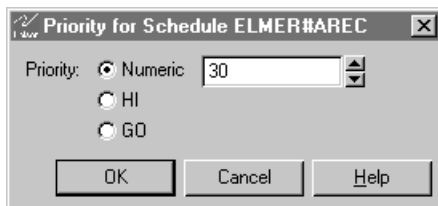
To qualify jobs based on whether or not they have a Prompt dependency, enter either the prompt name or number. For global prompts, enter the name of the prompt in the Prompt Name box and click the corresponding radio button. For local prompts, enter the number of the prompt in the Prompt Number box and click the corresponding radio button. Wildcards are permitted in the Prompt Name field only. If the Prompt toggle is selected but no prompt is specified, jobs are qualified if they have any Prompt dependencies.

Recovery Options To qualify jobs based on their Recovery Options, click the radio button for Stop, Continue, or Rerun.

The command buttons and general Filter dialog behavior are discussed in [Common Filter Attributes](#) on page 6-12.

Changing a Job's Priority

To change a job's priority, select the job and then choose Priority... from the Actions menu. The Priority for Job dialog opens.



Click Numeric, HI or GO. The Numeric range is 0-99. HI is equivalent to 100 and GO is equivalent to 101. HI and GO jobs are launched as soon as their dependencies are satisfied. They override the cpu job limit, but they do not override the schedule's job limit or Maestro's job fence.

Click OK to apply the new priority for the job and close the dialog. If you do not specify a priority level (that is, Numeric is selected but no number is in the entry field) and click OK, the job is assigned a priority of zero. Click Cancel to close the dialog and retain the job's previous priority.

Releasing a Job from its Dependencies

Dependencies can be released from a job or schedule individually or all at once. To release a job or schedule from all of its dependencies, select the entire job or schedule row and then choose Release from the Actions menu. To release a job or schedule from a single dependency, select the appropriate Dependencies or Start cell (not the entire row) and then choose Release from the Actions menu. Either way, a dialog opens confirming that you want to release the dependencies. Click Yes to release the job or No to take no action. For Needs resources dependencies, the released job is given the required number of units of the resource, even though they may not be available. This may cause the Available column of the SHOWRESOURCES window to display a negative number.

Canceling a Job

To cancel a job, select the job and then choose Cancel from the Actions menu. A dialog opens prompting you for confirmation to cancel the job. Click Yes to cancel the job or No to take no action.

If a job is cancelled before it is launched, it will not launch. If cancelled after it is launched, the job continues to execute. In either case, jobs and schedules that are dependent on the cancelled job are released. Cancelled jobs can be Rerun.

Rerunning a Job

A job can only be rerun if it is in the SUCC, CANCL, FAIL or ABEND state. To rerun a job, select the job and then choose Rerun from the Actions menu. A rerun job is placed in the same schedule as the original job, and inherits the original's dependencies. If you rerun a repetitive (**every**) job, the rerun job is scheduled to run at the same rate as the original job. In Conman displays, rerun jobs appear with the notation: >>**rerun as**.

Advanced features are available in the CLI **conman**. See [*rerun*](#) on page 9-68.

Killing a Job

To kill an executing job, select the job and then choose Kill from the Actions menu. The Kill operation is not performed by Conman, but is executed by Jobman, so there may be a short delay. Killed jobs terminate in the ABEND state. Any jobs or schedules that are dependent on a killed job are not released. Killed jobs can be rerun.

Confirming a Job

To confirm the completion of job that was scheduled with the Needs Confirmation option, select the job and choose Confirm from the Actions menu. From the cascading menu, select SUCC to confirm that the job ended successfully or ABEND to confirm that the job ended unsuccessfully.



The following table shows the affect of confirming jobs in various states.

Initial job state	State after confirming SUCC	State after confirming ABEND
ready	no affect	no affect
hold	no affect	no affect
exec	succp	abenp
abenp	succp	no affect
succp	no affect	no affect
pend	succ	abend
done	succ	abend
succ	no affect	no affect
abend	succ	no affect
fail	no affect	no affect
susp	no affect	no affect
skel	no affect	no affect
any EXTERNAL job	succ	abend

Changing a job from **abend** to **succ** does not require the **confirmed** keyword. For more information about EXTERNAL jobs, see appendix D, *Internetwork Dependencies*.

Viewing a Job's Standard List File

To view a job's standard list file, select the job and choose Stdlist... from the Actions menu. The Stdlist for Job window opens. The job number appears in the window title bar.

The screenshot shows a window titled "Stdlist for Job 1007". The content is as follows:

```

MAESTRO for WINDOWS NT/SYXTRACT 6.0 (3.18.1.12.1.4) (C) Tivoli Systems
(X) Installed for General Electric under group 'DEFAULT'.
MAESTRO for WINDOWS NT/REPORTER 6.0 (3.18.1.1.1) Ajax
Report 10A M199806011626 Actual Production Summary F

```

Estimated	Schedule	Run Time	Pri	Start Time	Actual	CPU	Elapsed Time	Run T
ELMER	AREC		10					
Totals>>>	1	00:00					00:00	00:0

```

| MAESTRO for WINDOWS NT/REPORTER 6.0 (3.18.1.1.1) Ajax
Report 10A M199806011626 Actual Production Summary F
Estimated Schedule Actual CPU

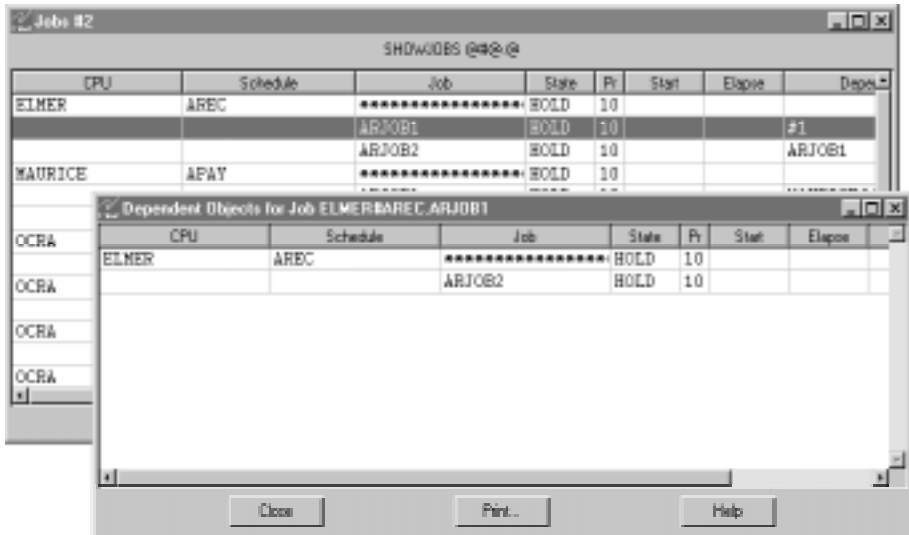
```

At the bottom of the window are three buttons: "Close", "Print...", and "Help".

A standard list file is created for each job launched by Maestro. Standard list files contain header and trailer banners, echoed commands, including errors and warnings, and the `stdout` and `stderr` output. Click Close to close the window or Print... to open the Print dialog.

Viewing Dependent Objects

To view objects dependent on a job, select the job and then choose Dependent Objects... from the Actions menu. The Dependent Objects for Job window opens.



Dependent jobs and schedules are displayed in the standard SHOWJOBS format. This is a read-only display. Click Close to close the dialog or Print... to open the Print dialog.

Adding Dependencies to a Job

To add dependencies to a job, select the job and choose Add Dependencies... from the Actions menu. The Add Job Dependencies window opens. For instructions on how to use this window see [Adding Dependencies to Jobs](#) on page 6-77.

Deleting Dependencies from a Job

To delete dependencies from a job, select the Dependencies cell or the Start cell (not the entire row) and then choose Delete Dependencies from the Actions menu. A dialog opens prompting you for confirmation to delete the dependency from the job. Click Yes to delete the dependency or No to take no action.

Viewing Job Details

To view job details and statistics, select the job and choose Details... from the Actions menu. The Details for Job window opens. Click Close to close the window or Print... to open the Print dialog.



The display presents the following information:

Job Info	Basic information about the job.
Job Number	The process id (PID) for the job, shown as: Jnnnnn
Logon	The user name under which the job executes.
Description	The description from the job's definition. If the job was submitted during production, a description is created.
Script File	The path name of the job's Script File or the job's Command.
Recovery Info	The job Recovery Options, if any:
Option	RERUN, CONTINUE, or STOP
Job	The name of the recovery job.
Prompt	The recovery prompt number.
Job Run Summary	Statistics from the job's current run.
#Total Runs	The number of times the job ran today.

#Successful Runs	The number of times the job ran successfully.
#Aborted Runs	The number of times the job abended or was terminated prematurely.
Total CPU Seconds	The cpu time (rounded to the nearest second) used by the job.
Total Elapsed Minutes	The total elapsed minutes (rounded to the nearest minute) the job ran.
Elapsed Minutes	The job's run time in <i>hh:mm</i> format. If the job has not yet run, the label reads Estimated Elapsed Minutes and an estimation based on logged statistics is printed in parenthesis.
Start Date	The job's start date in <i>mm/dd</i> format. If the job has not yet run, the label reads Estimated Start Date and an estimation on logged statistics is printed in parenthesis.
Run Time History	Statistics from the job's history.
Last Run At	The date, elapsed minutes, and cpu seconds from the last run of the job.
Minimum Run Time	The date, elapsed minutes, and cpu seconds from the shortest run of the job.
Maximum Run Time	The date, elapsed minutes, and cpu seconds from the longest run of the job.

Note: Run Time History will display zeroes until the **logman** program runs for the first time. Logman normally runs at the start of each day as part of the **jnextday** job. For jobs that are submitted in an ad hoc manner during a production day, historical information is unavailable.

Working with Internetwork Dependencies

SHOWJOBS windows display internetwork dependencies in special schedules named EXTERNAL. The dependencies are listed as jobs within EXTERNAL regardless of whether they are Maestro jobs or schedules. There is an EXTERNAL schedule for each Network agent. The Network agent name is found in the CPU column. Unique job names are generated as follows:

E <i>nnnmmss</i>	where:	<i>nnn</i>	is a random number
		<i>mm</i>	is current minutes
		<i>ss</i>	is current seconds

Internetwork dependencies are listed in the Dependencies column of SHOWJOBS and SHOWSCHEDULES windows in the following format:

net::net_dep

where: *net*

The name of the Maestro Network agent where the internetwork dependency is defined.

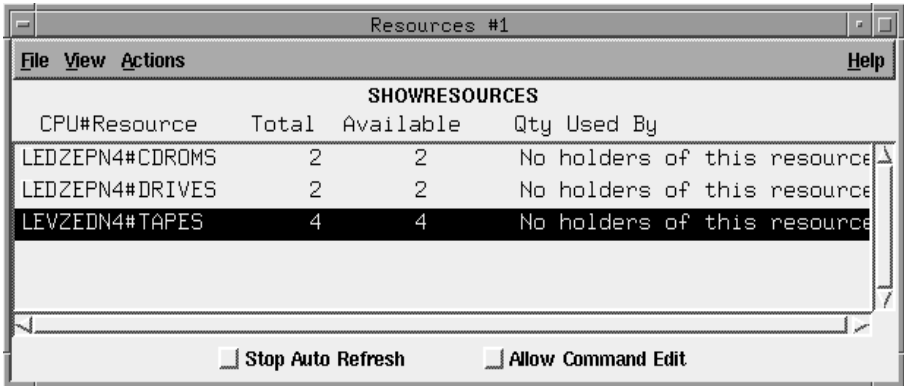
net_dep

The name of the internetwork dependency in the following format: [*cpu#*]*sched*[. *job*] If no *cpu* is specified, the default is the Maestro *cpu* to which the Network agent is connected. This is determined by the Node and TCP Address fields in the Network agent's *cpu* definition.

Refer to appendix D, [Internetwork Dependencies](#) for more information.

SHOWRESOURCES Window

The SHOWRESOURCES window displays resource information. Only one row can be selected at a time.



Reading the Display Area

Resource data is displayed in the following format:

CPU#Resource The fully-qualified name of the resource. The cpu on which the resource is defined, followed by a pound sign (#), and the name of the resource.

Total The total number of available resource units.

Available The number of resource units that have not been allocated.

Qty The number of resource units allocated to a job or schedule. If no units of the resource are allocated, **No holders of this resource** is printed across the Qty and Used By columns.

Used By The name of the job or schedule holding this resource.

Filtering the Resource Display

Complete the entry fields in the Resource Filter dialog to set the filter for SHOWRESOURCES windows.

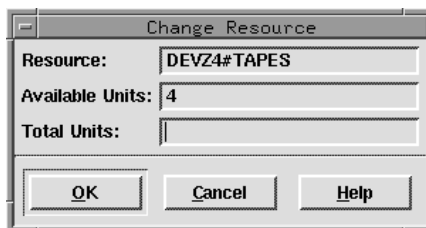


- Cpu** To qualify resources by cpu, enter the Maestro cpu definition name on which the resource is defined. Wildcards are permitted.
- Resource** To qualify resources by name, enter the resource name string. Wildcards are permitted.

The command buttons and general Filter dialog behavior are discussed in [Common Filter Attributes](#) on page 6-12.

Changing Available Units of a Resource

To change a resource's total available units, select the resource and then choose Change Units... from the Actions menu.

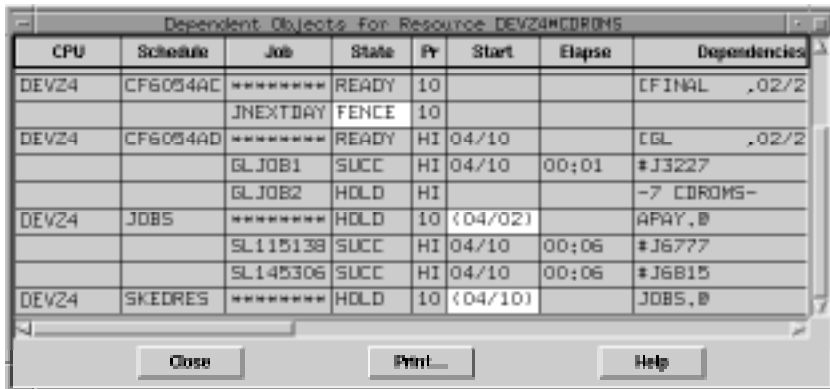


In the Change Resources dialog, the resource name appears in the Resource field. The Available Units field displays the number of unallocated, currently available units. These are both read-only fields.

The Total Units field is used to change the total available units of the resource. Enter a number (0-1024) and click OK to change the total units of the resource. The change is only in effect for the current production. Click Cancel to close the dialog and retain the previous number of Total Units.

Viewing Dependent Objects

To display jobs and schedules dependent on a resource, select the resource and then choose Dependent Objects... from the Actions menu. The Dependent Objects for Resource window opens.



The screenshot shows a window titled "Dependent Objects for Resource DEVZ4#CDROMS". It contains a table with the following columns: CPU, Schedule, Job, State, P#, Start, Elapse, and Dependencies. The data is as follows:

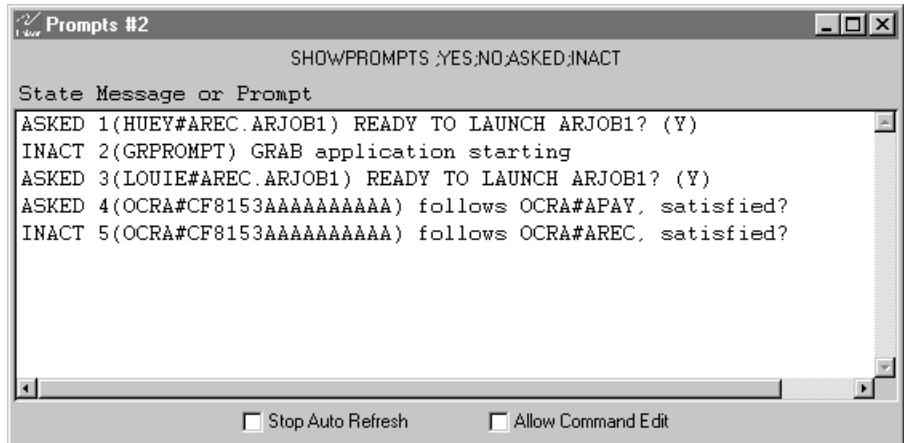
CPU	Schedule	Job	State	P#	Start	Elapse	Dependencies
DEVZ4	CF6054AC	*****	READY	10			CFINAL .02/2
		JNEXTDAY	FENCE	10			
DEVZ4	CF6054AD	*****	READY	HI	04/10		CGL .02/2
		GLJOB1	SUCC	HI	04/10	00:01	#J3227
		GLJOB2	HOLD	HI			-7 CDROMS-
DEVZ4	JOBS	*****	HOLD	10	<04/02		APAY,B
		SL115138	SUCC	HI	04/10	00:06	#J6777
		SL145306	SUCC	HI	04/10	00:06	#J6815
DEVZ4	SKEDRES	*****	HOLD	10	<04/10		JOBS,B

At the bottom of the window are three buttons: "Close", "Print...", and "Help".

Dependent jobs and schedules are displayed in the standard SHOWJOBS format. This is a read-only display. Click Close to close the window or Print... to open the Print dialog.

SHOWPROMPTS Window

The SHOWPROMPTS window displays the status of prompts from jobs and schedules. Only one row can be selected at a time.



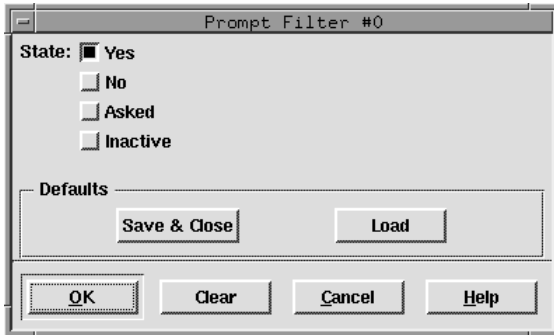
Reading the Display Area

Prompt data is displayed in the following format:

State	The state of the prompt. They are as follows:
YES	The prompt was replied to with Yes.
NO	The prompt was replied to with No.
ASKED	The prompt was issued, but there is no reply.
INACT	The prompt is yet to be issued.
Message or Prompt	For global prompts: the prompt number, the name of the prompt, and the message text. For local prompts: the prompt number, the name of the job or schedule, and the message text.

Filtering the Prompt Display

From the Options menu, select Filter > Prompts to open the Prompt Filter dialog.



To qualify prompts by state, select one or more of the State buttons. For a description of prompt states, see [page 6-59](#). The command buttons and general Filter dialog behavior are discussed in [Common Filter Attributes](#) on page 6-12.

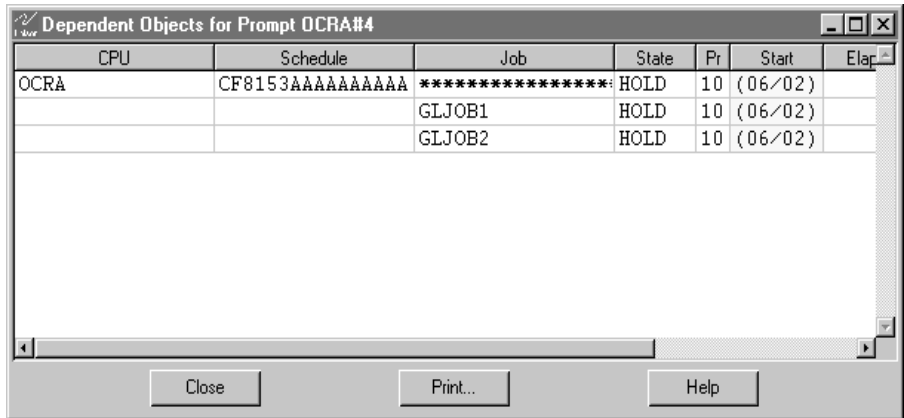
Replying to a Prompt

To reply to a prompt, select the prompt and then select Reply > Yes or Reply > No from the Actions menu.



Viewing Dependent Objects

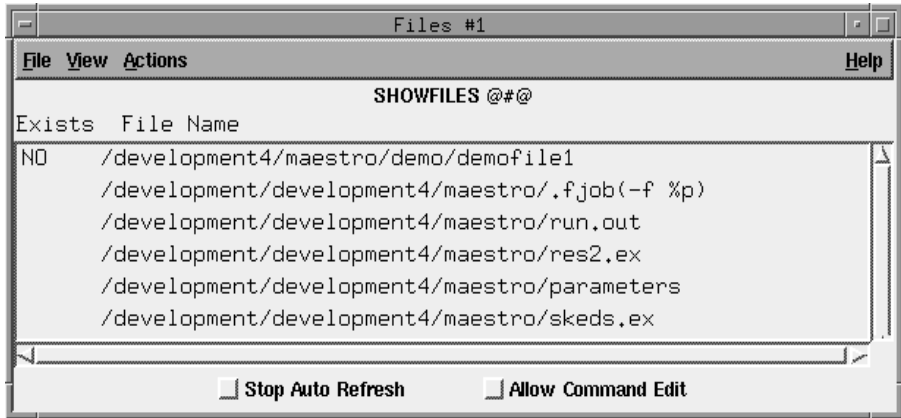
To display jobs and schedules dependent on a prompt, select the prompt and then choose Dependent Objects... from the Actions menu. The Dependent Objects for Prompt dialog opens.



Dependent jobs and schedules are displayed in the standard SHOWJOBS format. This is a read-only display. Click Close to close the dialog or Print... to open the Print dialog.

SHOWFILES Window

The SHOWFILES window displays the status of file dependencies. Only one row can be selected at a time.



Reading the Display Area

File dependency data is displayed in the following format:

- | | |
|----------------------|---|
| Exists | The state of the prompt. They are as follows: |
| YES | The file exists and is available. |
| NO | The file is not available, or does not exist. |
| ? | Availability of the file is being checked. |
| <blank> | The file has not been checked yet. |
| File Name | The cpu, followed by a pound sign (#), and full path name of the file, including test options. If no cpu is listed, the local cpu is the default. |

Filtering the File Display

From the Options menu, select Filter > Files to open the File Filter dialog.



CPU To qualify files by cpu, enter the Maestro cpu name on which the Opens file dependency is specified. Wildcards are permitted.

File To qualify files by name, enter the file path name. Wildcards are permitted.

The command buttons and general Filter dialog behavior are discussed in *Common Filter Attributes* on page 6-12.

Viewing Dependent Objects

To display jobs and schedules dependent on a file, select the file and choose Dependent Objects... from the Actions menu. The Dependent Objects for File dialog opens. Dependent jobs and schedules are displayed in the standard SHOWJOBS format. This is a read-only display. Click Close to close the dialog or Print... to open the Print dialog.

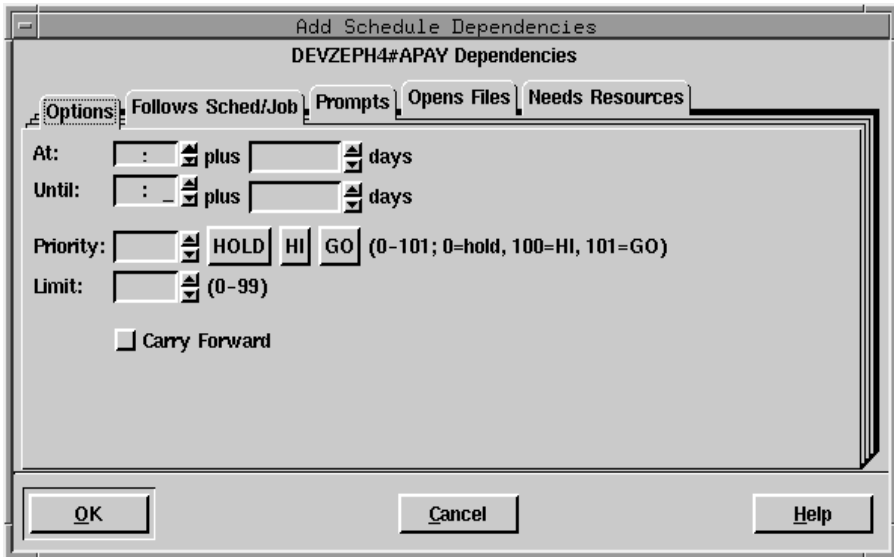
CPU	Schedule	Job	State	Pr	Start	Elapse	Dependencies
DEVZ4	CF6054AA	*****	READY	10			[FINAL ,02/23/9
		JNEXTDAY	FENCE	10			

Adding Dependencies to Schedules

With the Add Schedule Dependencies window, you can add dependencies to schedules that are already part of production or that you are submitting to production.

Add Schedule Dependencies Window: Basic Operation

The Add Schedule Dependencies window is similar to Composer’s Schedule Definition window. They share many of the data panels with common input fields, buttons, and lists.



You should be familiar with the basic operation of the Schedule Definition window before using the Add Schedule Dependencies window. This section provides a summary for each data panel and points out unique behavior for the Add Schedule Dependencies window. For a detailed description of Composer’s Schedule Definition window, refer to *Selecting Schedule Dependencies* on page 5-6.

Opening the Window

The Add Schedule Dependencies window can be opened in two ways:

- By selecting a schedule in a SHOWSCHEDULES or SHOWJOBS window and then choosing Add Dependency... from the Actions menu. Dependencies added from here apply to the schedule's current production run.
- By clicking the Dependencies... button on a Submit Schedule dialog. Dependencies added from here apply to newly submitted production. For more information on submitting production, see [Ad Hoc Scheduling](#) on page 6-15.

The name of the schedule to which you are adding dependencies is displayed above the dependency category tabs.

When an Add Schedule Dependencies window is opened, the data panels are always blank, even if dependencies exist. If you specify a new dependency that conflicts with an existing one, the existing dependency is overwritten with the new one. For example, if a schedule has an existing Priority of 50 and you use the Add Schedule Dependencies window to specify a Priority of 90, the new value (90) overwrites the former (50) value. Use the Delete Dependency option ([page 6-39](#)) to delete dependencies from a schedule.

Applying Dependencies to a Schedule

To apply specified dependencies to the target schedule, click OK on the bottom of the Add Schedule Dependencies window. If you do not want to apply the specified dependencies, click Cancel. In either case the window will close.

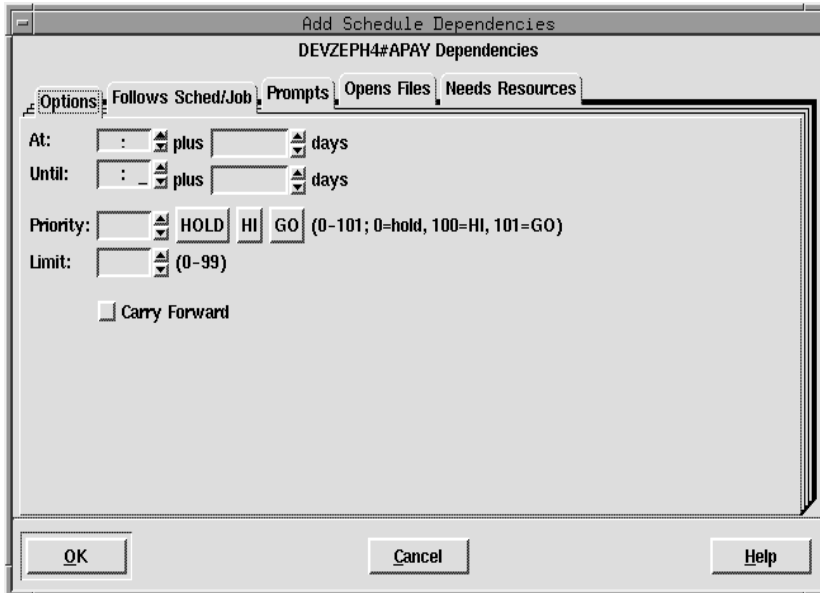
Dependencies may not be immediately displayed in the SHOWSCHEDULES or SHOWJOBS window if the screen has not been refreshed, the dependency has been satisfied, or if Batchman is not running. Select Refresh Now from the View menu to immediately refresh the display.

Using Allow Command Edit and Adding Dependencies

If you select Add Dependency... from a SHOWSCHEDULES or SHOWJOBS window and Allow Command Edit is enabled, you are taken to the Conman Command Window when OK is clicked on the Add Schedule Dependencies window. The command line version of the Add Dependency command and the options selected in the Add Schedule Dependencies are displayed in the Command Input area. For more information on using the Conman Command Window see [Issuing Command Line Commands](#) on page 6-3.

Options Panel: Selecting Schedule Options

The Options panel specifies the time of day a schedule is to start and stop executing, its priority, its job limit, and whether or not it is carried forward to the next day if it does not finish.



In the time fields, the arrows change the value in 15 minutes increments; in all other fields, the arrows change the value by one unit.

Selecting Schedule Execution Start Time

The At fields define the time of day the schedule is launched. Enter the time, in 24-hour format, in the box to the right of At. The range is 00:00-23:59.

You can enter an offset At time using the plus entry box. Enter an offset in days (0-99) from the scheduled launch time. The offset can be used at the schedule or job level, but not both. The offset days are Maestro processing days, not regular calendar days. When used at the schedule level, the offset is applied to all jobs in the schedule.

When using At and Until in the same schedule, make certain that the Until time is later than the At time. Using both At and Until creates a window within which the schedule is executed.

Selecting Schedule Execution Until Time

The Until fields define the time of day after which a schedule will not be launched. See [Selecting Schedule Execution Start Time](#) on page 6-66 for instructions on how to specify the time and offset, and for more information about using Until.

Selecting a Schedule's Priority

The Priority field assigns values to Maestro schedules to prioritize their relative importance during production. If you have two schedules, both of which have all their dependencies satisfied, the one with the higher priority will launch first.

In the Priority field, enter a number (0-101) or click HOLD, HI, or GO. HOLD is equivalent to a priority of zero, HI is equivalent to 100, and GO is equivalent to 101. Only an integer can be entered in the entry box. If HOLD, HI, or GO is selected, then the corresponding value is automatically entered in the box. If no selection is made, a value of 10 is assigned when the schedule is queued for execution.

HOLD, or a priority of zero, prevents the schedule from being launched. In the case of HI and GO, all jobs in the schedule are given HI or GO priority. HI and GO jobs are launched as soon as their dependencies are satisfied. They override the cpu job limit, but they do not override the schedule's job limit or Maestro's job fence.

Selecting a Schedule's Limit

The Limit field specifies the maximum number of jobs that can be executing concurrently in the schedule. Enter a job limit (0-99) for the schedule. A limit of zero means that no jobs will launch. If no limit is set, the maximum number of jobs that can execute concurrently from a single schedule is 99.

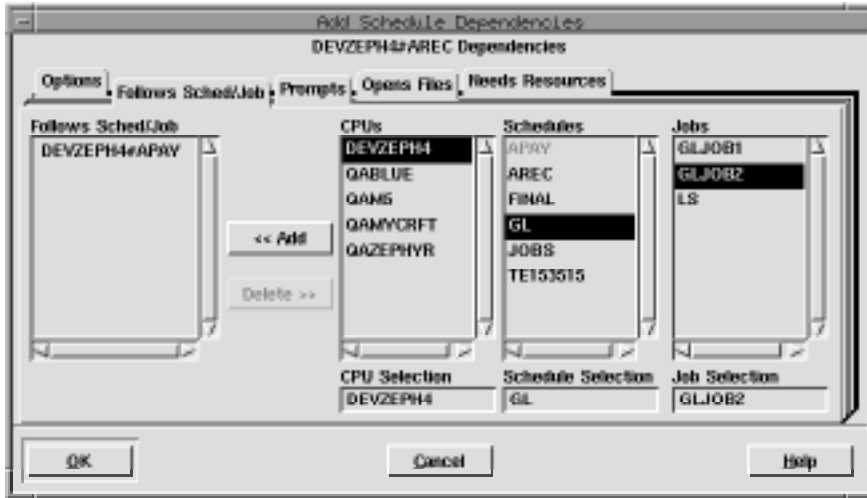
Specifying a Schedule to Carry Forward to the Next Day

Maestro can carry a schedule forward into the next processing day if it has not completed successfully. Click the Carry Forward button to enable this function.

Schedules that are carried forward retain the Carry Forward option, and therefore, may be carried forward again.

Follows Sched/Job Panel: Selecting Schedule Follows Dependencies

The Follows Sched/Job panel specifies the other schedules and jobs that must be completed before this schedule can be launched. The panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of jobs and schedules that this schedule must follow. The right side contains lists from which to choose available jobs and schedules.



If a “followed” job or schedule is not selected to run on the same day as a schedule that “follows” it, the dependency has no effect.

Reading the Follows List

The Follows Sched/Job list displays the schedules and jobs that must be completed successfully before this schedule can be launched. A list item consists of the qualified Maestro schedule or job name.

Selecting and Adding a Job or Schedule to Follow

The three lists on the right are used to select the followed jobs and schedules. The CPU Selection determines what is displayed in the Schedules list. The Schedule Selection determines what is displayed in the Jobs list. A cpu, schedule, or job can be selected by either selecting it from the appropriate list or by typing its name in the appropriate Selection box. For Job Selection, the @ wildcard by itself indicates all jobs in a schedule.

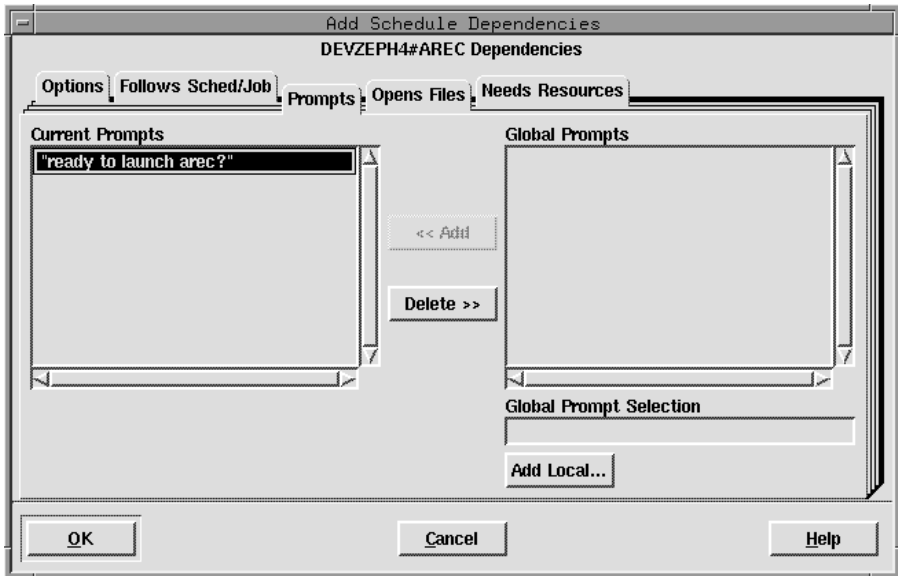
When a job or schedule is in a selection box, the Add button is enabled. Click Add to insert the selection to the Follows Sched/Job list. Double-clicking on the selected job or schedule in the lists also adds it to the Follows Sched/Job list. To select a schedule, the Job Selection box must be empty or have the @ wildcard entered.

Deleting a Job or Schedule as a Follows Dependency

Selecting one or more items from the Follows Sched/Job list enables the Delete button. Click Delete to remove selected items.

Prompts Panel: Selecting Prompts for a Schedule

The Prompts panel specifies the prompts that must be issued or answered affirmatively before the schedule can be launched. Both global and local prompts can be added to the schedule from this panel. The prompts panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of prompts that must be responded to. The right side contains a list from which to choose available prompts.



Reading the Current Prompts List

The Current Prompts list displays the prompts selected for the schedule. Global prompts are listed by name; local prompts are listed with the text of the prompt in double quotes.

Selecting and Adding a Global Prompt

Select a global prompt either by typing its name in the Global Prompt Selection box or by selecting it from the Global Prompts list. The list displays the name of the prompt followed by the prompt text in double quotes.

When a global prompt is selected, the Add button is enabled. Click Add to insert the prompt into the Current Prompts list. Double-clicking on the desired prompt in the Global Prompts list also adds it to Current Prompts.

Defining and Adding a Local Prompt to a Schedule

To define a local prompt for a schedule, click the Add Local... button. In the Add Local Prompt dialog, enter the desired prompt text in the Prompt entry box (up to 255 characters). If the text begins with a colon (:), the prompt is issued, but no reply is necessary to continue processing. If the text begins with an exclamation point (!), the prompt is not issued, but a reply is necessary to proceed. You can include backslash "n" (\n) within the text to cause a new line. Maestro parameters are permitted and must be enclosed in carets (^).

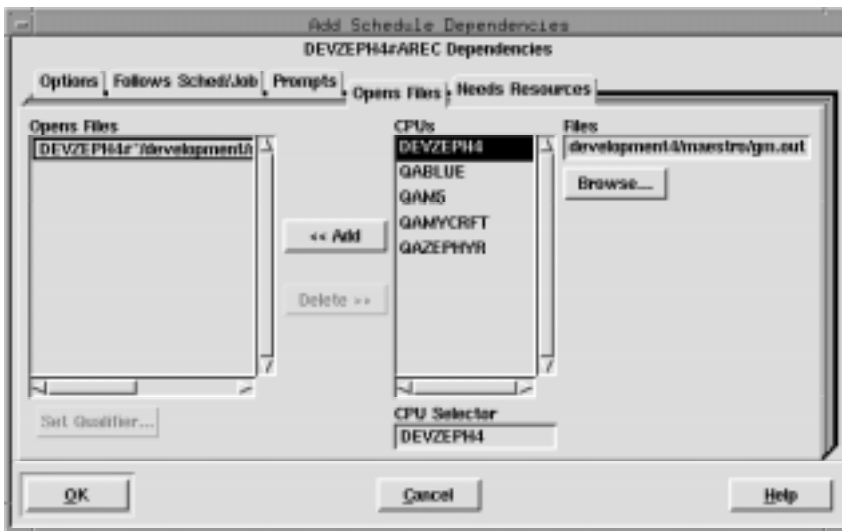
Click OK to add the prompt to the Current Prompts list. Click Cancel to define no local prompt.

Deleting Prompts From a Schedule

Selecting one or more prompts from the Current Prompts list enables the Delete button. Click Delete to remove the selected prompts.

Opens Files Panel: Selecting Schedule File Dependencies

The Opens Files panel specifies files that must exist before a schedule is launched. The panel is split into two sections, separated by the Add and Delete buttons. The left side has a list of file “opens” dependencies, and the right side has a list of available files.



Reading the Opens Files List

The Opens Files list contains the cpu and path names of the files on which the schedule depends. Cpu and path are delimited by a pound sign (#). Any test qualifiers appear at the end of the file name in parenthesis. If no Qualifier is specified the default is (-f %P).

Note: The combination of path and qualifier cannot exceed 148 characters, within which the basename of the path cannot exceed 28 characters.

Selecting a File as a Schedule Dependency

Select the cpu or cpu class on which the desired file resides either by typing its name in the CPU Selection box or by selecting it in the CPUs list. If the

current schedule is defined for a cpu class, only that class is included in the CPUs list.

Select the path name of the file either by selecting it with the file selection dialog opened by clicking Browse, or by typing it directly. One or more Maestro parameters can be used and must be enclosed in carets (^).

Note: The Browse function only displays the files and directories for the cpu running the Console Manager.

Adding a File Dependency to a Schedule

When a file pathname is in the File box, the Add button is enabled. Click Add to move the pathname to the Opens Files list. If the file is already in the list, it is not added again.

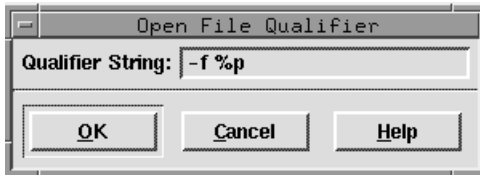
Note: If you choose a cpu class as the CPU Selection for the file dependency, the file must exist on every cpu in the class at schedule execution time. If the file does not exist, the file dependency cannot be resolved and the schedule will not be launched.

Deleting a File Dependency from a Schedule

When one or more file pathnames are selected in the Opens Files list, the Delete button is enabled. Click Delete to remove selected file pathnames.

Specifying a Test Qualifier for a File Dependency

To specify a test option for a file dependency, select the file in the Opens Files list and then click the Set Qualifier... button. The Open File Qualifier dialog opens.



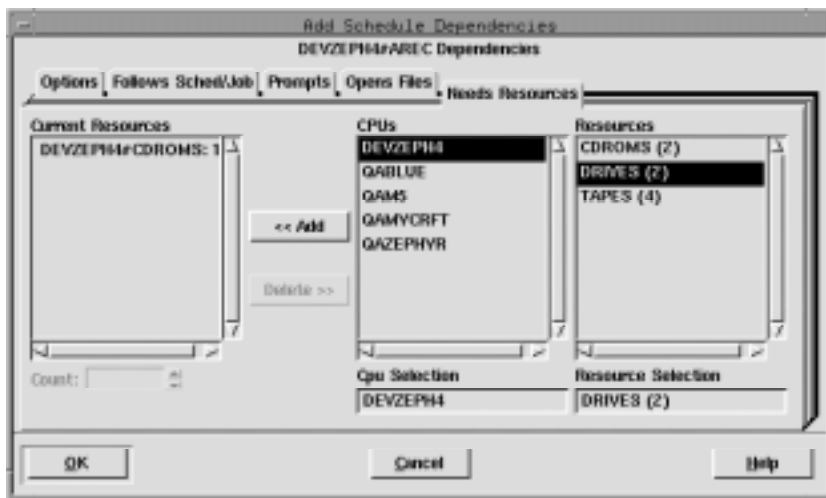
Enter the file qualifier in the Qualifier String box (Maestro adds the parenthesis to the qualifier when the string is appended to the file in the Opens File list). Click OK to append the qualifier to the selected file pathname in the Opens Files list. Click Cancel to retain the previous qualifier.

For information about UNIX and Windows NT qualifiers, and examples see [*Specifying a Qualifier for a File Dependency*](#) on page 5-23.

Needs Resources Panel: Assigning Resources to a Schedule

The Needs Resources panel specifies Maestro resources that must be available to the schedule before it can be launched. You can have a resource dependency at either the job or schedule level, but not both. The number of jobs and schedules using a resource at any one time cannot exceed 32.

The Needs Resources panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of resources for the schedule. The right side contains lists from which to choose available resources.



Reading the Current Resources List

The Current Resources list contains the resources that must be available for the schedule to launch. The resource name is preceded by the name of the cpu or cpu class on which the resource is defined and followed by the number of units of the resource required by the schedule. The cpu and resource are delimited by a pound sign (#) and the resource and units allocated are delimited by a colon (:). If the resource is defined on the same cpu or cpu class as this schedule, then the cpu and delimiter are omitted in the list. If the resource is defined for a cpu class, the class name is omitted in the list.

Selecting and Adding a Resource

To select a resource, enter the name of the cpu or cpu class on which the desired resource is defined by either typing its name in the CPU Selection box or by selecting one from the CPUs list. The Resources list is filtered to display the resources for that cpu or cpu class. If the current schedule is defined for a cpu class, then only that class is included in the list.

Complete the selection either by typing a resource name in the Resource Selection list or by selecting a resource from the Resources list. In the list, the name of the resource is followed by the total number of units in parenthesis.

When a resource name is in the Resource Selection box, the Add button is enabled. Click Add to append it to the Current Resources list. Double-clicking on the resource name in the Resources list also appends it to the Current Resources list.

Deleting a Resource from a Schedule

Selecting one or more resources in the Current Resources list enables the Delete button. Click Delete to remove selected resources.

Changing the Number of Resource Units a Schedule Needs

To change the number of resource units the schedule needs, select the resource in the Current Resources list and enter a number in the entry box or use the up/down arrow buttons. The default count is one unit. If more than one item is selected in the Current Resources list, the Count field is disabled.

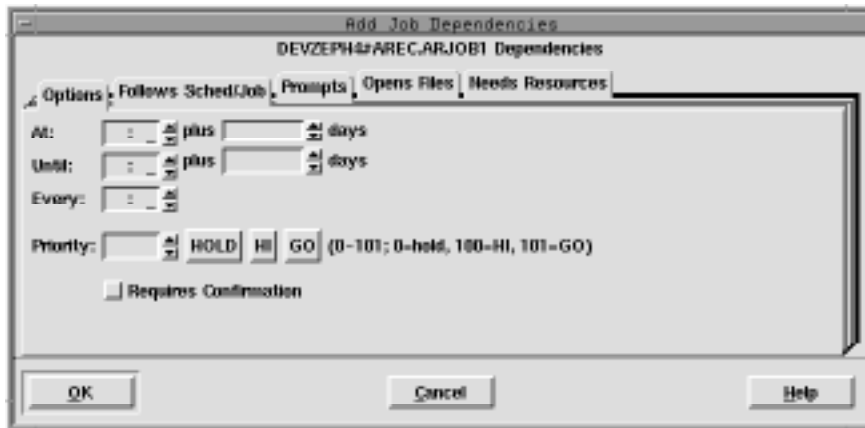
A maximum of 32 units is permitted per resource entry. However, if more than 32 are needed, the resource can be added again. The total number of resources that can be assigned to a schedule is 1024.

Adding Dependencies to Jobs

With the Add Job Dependencies window, Conman permits you to add dependencies to jobs that are already part of production or that you are submitting to production. Script files and commands that are submitted are considered jobs.

Add Job Dependencies Window: Basic Operation

The Add Job Dependencies window is similar to Composer's Job Dependencies window. They share many of the data panels with common input fields, buttons, and lists.



You should be familiar with the basic operation of the Job Dependency window before using the Add Job Dependencies window. This section provides a summary for each data panel and points out unique behavior for the Add Job Dependencies window. For a detailed description of Composer's Job Dependencies window, refer to [Selecting Job Dependencies](#) on page 5-30.

Opening the Window

The Add Schedule Dependencies window can be opened two ways:

- By selecting a job in a SHOWJOBS window and then choosing Add Dependency... from the Actions menu. Dependencies added from here apply to the job's current production run.
- By clicking the Dependencies... button on a Submit Job, Submit File, or Submit Command dialog. Dependencies added from here apply to newly submitted production. For more information on submitting production,

see [Ad Hoc Scheduling](#) on page 6-15.

The name of the schedule and job to which you are adding dependencies is displayed above the dependency category tabs.

When an Add Job Dependencies window is opened, the data panels are always blank, even if dependencies exist. If you specify a new dependency that conflicts with an existing one, the existing dependency is overwritten with the new one. For example, if a job has an existing Priority of 50 and you use the Add Job Dependencies window to specify a Priority of 90, the new value (90) overwrites the former (50) value. Use the Delete Dependency option ([page 6-52](#)) to delete dependencies from a job.

Applying Dependencies to a Job

To apply specified dependencies to the target job, click OK on the bottom of the Add Job Dependencies window. If you do not want to apply the specified dependencies, click Cancel. In either case the window will close.

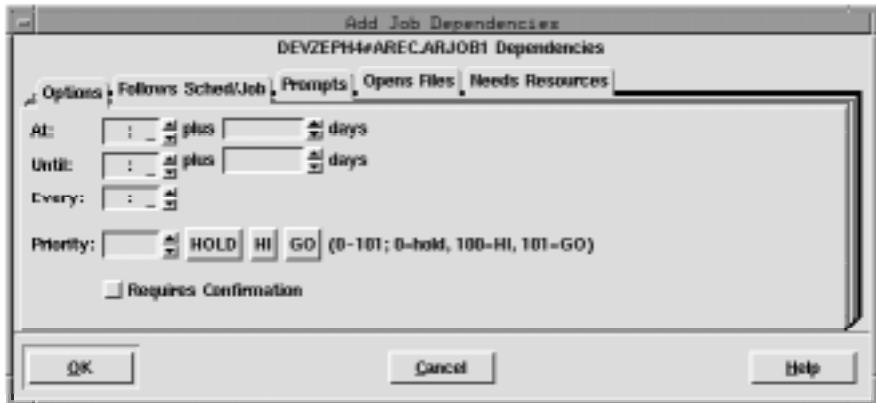
Dependencies may not be immediately displayed in the SHOWJOBS window if the screen has not been refreshed, the dependency has been satisfied, or if Batchman is not running. Select Refresh Now from the View menu to immediately refresh the display.

Using Allow Command Edit and Adding Dependencies

If you select Add Dependency... from a SHOWJOBS window and Allow Command Edit is enabled, you are taken to the Conman Command Window when OK is clicked on the Add Job Dependencies window. The command line version of the Add Dependency command and the options selected in the Add Job Dependencies are displayed in the Command Input area. For more information on using the Conman Command Window see [Issuing Command Line Commands](#) on page 6-3.

Options Panel: Selecting Job Options

The Options panel specifies the time of day a job is to start and stop executing, how often it is to execute, its priority, and whether or not it needs outside intervention to go to a successful state.



In the time fields, the arrows change the value in 15 minutes increments; in all other fields, the arrows change the value by one unit.

Selecting Job Execution Start Time

The At fields define the time of day the job is launched. Enter the time, in 24-hour format, in the box to the right of At. The range is 00:00-23:59.

You can enter an offset At time using the plus entry box. Enter an offset in days (0-99) from the scheduled launch time. The offset can be used at the schedule or job level, but not both. The offset days are Maestro processing days, not regular calendar days. When used at the schedule level, the offset is applied to all jobs in the schedule.

When using At and Until for the same job, make certain that the Until time is later than the At time. Using both At and Until creates a window within which the job is executed.

Selecting Job Execution Until Time

The Until fields define the time of day after which a job will not be launched. See [Selecting Job Execution Start Time](#) above for instructions on how to specify the time and offset, and for more information about using Until.

Specifying a Repeat Rate for a Job

The Every field defines the rate at which a job is to be launched repeatedly. Enter the repeat rate, in hours followed by minutes, by either typing it in or using the up/down arrows.

If an Every value is used, an attempt is made to launch the job repeatedly at the specified rate. The number of times the job actually executes depends on its dependencies, the level of system activity, and the availability of job slots. All job dependencies must be satisfied each time an attempt is made to launch the job.

If you cancel a repetitive job, it will not be launched again. If you rerun a repetitive job, the next iteration of the job is run immediately. If you rerun a repetitive job that had been cancelled, the repetition rate is re-instated. If a repetitive job abends, the repetitions continue following the optional recovery action.

Selecting a Job's Priority

The Priority field assigns values to Maestro jobs to prioritize their relative importance during production. If you have two jobs, both of which have all their dependencies satisfied, the one with the higher priority will launch first.

In the Priority field, enter a number (0-101) or click HOLD, HI, or GO. HOLD is equivalent to a priority of zero, HI is equivalent to 100, and GO is equivalent to 101. Only an integer can be entered in the entry box. If HOLD, HI, or GO is selected, then the corresponding value is automatically entered in the box. If no selection is made, a value of 10 is assigned when the job is queued for execution.

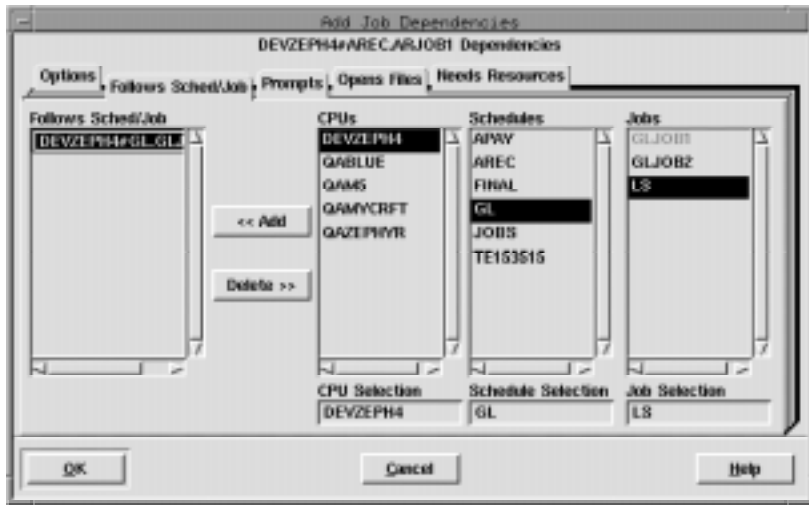
HOLD, or a priority of zero, prevents the job from being launched. HI and GO jobs are launched as soon as their dependencies are satisfied. They override the cpu job limit, but they do not override the schedule's job limit or Maestro's job fence.

Requiring a Job's Completion to be Confirmed

You can specify that a job's completion must be confirmed from Conman (with the Confirm option in the SHOWJOBS Actions menu) before it reaches a termination state. Click the Requires Confirmation button to enable this function.

Follows Sched/Job Panel: Selecting Job Follows Dependencies

The Follows Sched/Job panel specifies the other schedules and jobs that must be completed before this job is launched. The panel is split into two sections, separated by the Add and Delete buttons. The left side has the list of jobs and schedules this job must follow. The right side has a list of available jobs and schedules.



If a “followed” job or schedule is not selected to run on the same day as a job that “follows” it, the dependency has no effect.

Reading the Follows List

The Follows Sched/Job list shows the schedules and jobs that must be completed successfully before this job is launched. If the cpu or cpu class is the same as this schedule, it is not listed. If a followed job is in the same schedule as this job, the schedule name is omitted.

Selecting and Adding Jobs or Schedules to Follow

Use the three lists on the right to select followed jobs and schedules. The CPU Selection determines what is displayed in the Schedules list. The Schedule Selection determines what is displayed in the Jobs list. Cpus, schedules, and jobs are selected from the lists, or by typing their names in the Selection boxes.

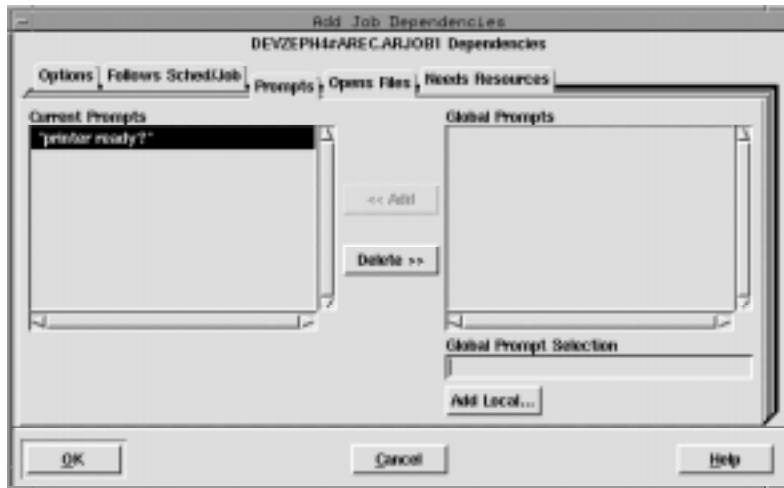
When a job or schedule is selected, the Add button is enabled. Click Add to insert it into the Follows Sched/Job list. Double-clicking a job or schedule in a list also adds it to the Follows Sched/Job list. To select a schedule, the Job Selection box must contain the @ wildcard.

Deleting a Job or Schedule as a Follows Dependency

Selecting one or more items in the Follows Sched/Job list enables the Delete button. Click Delete to remove the items.

Prompts Panel: Selecting Prompts for a Job

The Prompts panel specifies the prompts that must be issued or answered affirmatively before the job can be launched. Both global and local prompts can be added from this panel. The prompts panel is split into two sections, logically divided by the Add and Delete buttons. The left side is a list of prompts that must be responded to. The right side is a list of available prompts.



In the Current Prompts list, global prompts are listed by name, and local prompts are listed with the text of the prompt in double quotes.

Selecting and Adding a Global Prompt

Select a global prompt either by typing its name in the Global Prompt Selection box or by selecting it from the Global Prompts list. The list displays the name of the prompt followed by the prompt text in double quotes.

When a global prompt is selected, the Add button is enabled. Click Add to insert the prompt into the Current Prompts list. Double-clicking on the desired prompt in the Global Prompts list also adds it to Current Prompts.

Defining and Adding a Local Prompt to a Job

To define a local prompt for a job, click the Add Local... button. In the Add Local Prompt dialog, enter the desired prompt text in the Prompt entry box (up to 255 characters). If the text begins with a colon (:), the prompt is issued, but no reply is necessary to continue processing. If the text begins with an

exclamation point (!), the prompt is not issued, but a reply is necessary to proceed. You can include backslash "n" (\n) within the text to cause a new line. Maestro parameters are permitted and must be enclosed in carets (^).

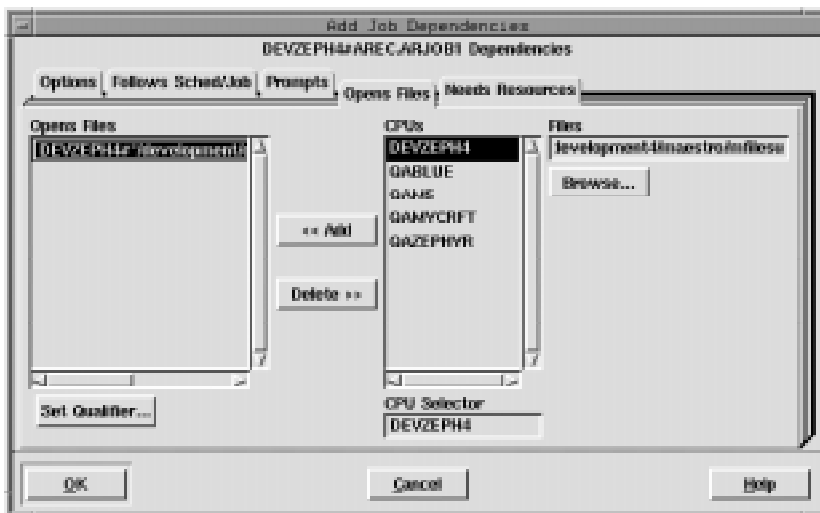
Click OK to add the prompt to the Current Prompts list. Click Cancel to define no local prompt.

Deleting Prompts From a Job

Selecting one or more prompts from the Current Prompts list enables the Delete button. Click Delete to remove the selected prompts.

Opens Files Panel: Selecting Job File Dependencies

The Opens Files panel specifies files that must exist before a job is launched. The panel is split into two sections, separated by Add and Delete buttons. The left side has a list of file dependencies, and the right side lists available files.



Reading the Opens Files List

The Opens File list contains the path names of the files upon which the job is dependent. Any test qualifiers are listed at the end of the file name in parenthesis. If no Qualifier String is specified, the default is (-f %P).

Note: The combination of path and qualifier cannot exceed 148 characters, within which the basename of the path cannot exceed 28 characters.

Selecting a File for a Job Dependency

Select the cpu or cpu class on which the desired file resides either by typing its name in the CPU Selection box or by selecting it from the CPUs list. If the current schedule is defined for a cpu class, then only that class is included in the CPUs list.

Select the path name of the file either by selecting it with the file selection box opened by clicking Browse... or by typing it directly. One or more Maestro parameters can be used and must be enclosed in carets (^).

Note: The file selection box only displays the files and directories for the cpu running Conman.

Adding a File Dependency to a Job

When a file pathname is in the File box, the Add button is enabled. Click Add to move the pathname to the Opens Files list. If the file is already in the list, it is not added again.

Note: If you choose a cpu class as the CPU Selection for the file dependency, the file must exist on every cpu in the class at job execution time. If the file does not exist, the file dependency cannot be resolved and the job will not be launched.

Deleting a File Dependency from a Job

When one or more file pathnames are selected in the Opens Files list, the Delete button is enabled. Click Delete to remove selected file pathnames.

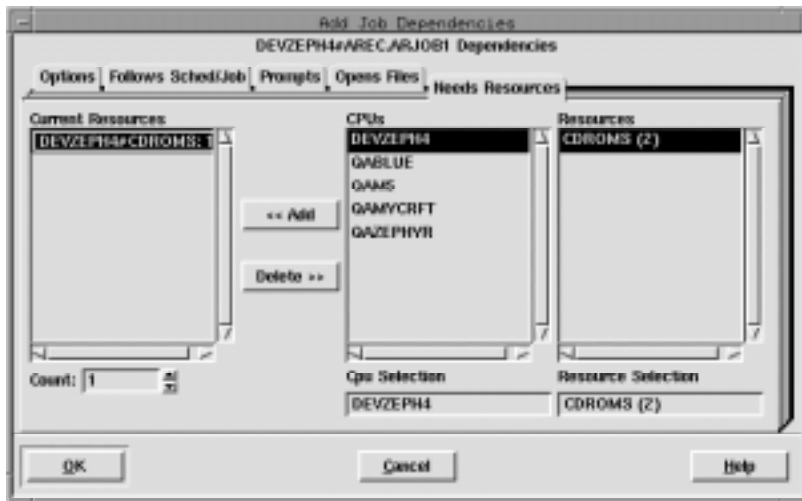
Specifying a Test Qualifier for a File Dependency

To specify a test option for a file dependency, select the file in the Current Files list and then click the Set Qualifier... button. In the Open File Qualifier dialog, enter the file qualifier in the Qualifier String box (Maestro adds the parenthesis to the qualifier when the string is appended to the file in the Opens File list). Click OK to append the qualifier to the selected file pathname in the Open Files list. Click Cancel to retain the previous qualifier. For information about UNIX and Windows NT qualifiers, and examples see [Specifying a Qualifier for a File Dependency](#) on page 5-23.

Needs Resources Panel: Assigning Resources to a Job

The Needs Resources panel specifies Maestro resources that must be available to the job before it can be launched. You can have a resource dependency at either the job or schedule level, but not both. The number of jobs and schedules using a resource at any one time cannot exceed 32.

The Needs Resources panel is split into two sections, logically divided by the Add and Delete buttons. The left side contains the list of resources for the job. The right side contains lists from which to choose available resources.



Reading the Current Resources List

The Current Resources list contains the resources that must be available for the job to launch. The resource name is preceded by the name of the cpu or cpu class on which the resource is defined and followed by the number of units of the resource required by the job. The cpu and resource are delimited by a pound sign (#) and the resource and units allocated are delimited by a colon (:). If the resource is defined on the same cpu or cpu class as the job's schedule, then the cpu and delimiter are omitted in the list. If the resource is defined for a cpu class, the class name is omitted in the list.

Selecting and Adding a Resource

To select a resource, enter the name of the cpu or cpu class on which the desired resource is defined by either typing its name in the CPU Selection box

or by selecting one from the CPUs list. The Resources list is filtered to display the resources for that cpu or cpu class. If the current schedule is defined for a cpu class, then only that class is included in the list.

Complete the selection either by typing a resource name in the Resource Selection list or by selecting a resource from the Resources list. In the list, the name of the resource is followed by the total number of units in parenthesis.

When a resource name is in the Resource Selection box, the Add button is enabled. Click Add to append it to the Current Resources list. Double-clicking on the resource name in the Resources list also appends it to the Current Resources list. If the resource is already in the Current Resources list, it is not added again (unless more than 32 units of the resource are needed).

Deleting a Resource from a Job

Selecting one or more resources in the Current Resources list enables the Delete button. Click Delete to remove selected resources.

Changing the Number of Resource Units a Job Needs

To change the number of resource units the job needs, select the resource in the Current Resources list and enter a number in the entry box or use the up/down arrow buttons. The default count is one unit. If more than one item is selected in the Current Resources list, the Count field is disabled.

A maximum of 32 units is permitted per resource entry. However, if more than 32 units are needed, the resource can be added again.

7

Reports

This section describes Maestro's report commands and extract programs, and contains samples of the reports.

Report Commands

Maestro's report commands are listed below.

Command	Report *	Description
rep1 (page 7-3)	Report 01	Job Details Listing
rep2 (page 7-3)	Report 02	Prompt Messages Listing
rep3 (page 7-3)	Report 03	User Calendar Listing
rep4a (page 7-3)	Report 04A	User Parameters Listing
rep4b (page 7-3)	Report 04B	Maestro Resource Listing
rep7 (page 7-4)	Report 07	Job History Listing
rep8 (page 7-5)	Report 08	Job Histogram
rep11 (page 7-7)	Report 11	Planned Production Schedule
reprtr (page 7-8)	Report 09A	Planned Production Summary
	Report 09B	Planned Production Detail
	Report 09D	Planned Production Detail (Long Names)
	Report 10B	Actual Production Summary
	Report 10B	Actual Production Detail
xref (page 7-10)	Report 12	Cross Reference Report

* See [Sample Reports](#) starting on page 7-12 for examples of the reports.

Offline Output

The output of the report commands is controlled by the following shell variables:

MAESTROLP	The destination of a command's output. You can set it to any of the following: <ul style="list-style-type: none">> <i>filename</i> Redirect output to a file, overwriting the contents of the file. If the file does not exist it is created.>> <i>filename</i> Redirect output to a file, appending to the end of file. If the file does not exist it is created. <i>command</i> Pipe output to a system command or process. The system command is always executed. <i>command</i> Pipe output to a system command or process. The system command is not executed if there is no output.
MAESTROLPLINES	The number of lines per page. If not set, the default is 60.
MAESTROLPCOLUMNS	The number of characters per line. If not set, the default is 132.
MAESTRO_OUTPUT_STYLE	Set to LONG to have Maestro use full length (long) fields for object names. If not set, or set to anything other than LONG , and the global option expanded version is set to yes , long names are truncated to eight characters and a plus sign. For example: A1234567+ . If expanded version is set to no , long names are truncated to eight characters.

If **MAESTROLP** is not set, the output goes to **stdout**.

rep1-rep4b

These commands print the following reports:

Report 01	Job Details Listing
Report 02	Prompt Messages Listing
Report 03	User Calendar Listing
Report 04A	User Parameters Listing
Report 04B	Maestro Resource Listing

```
rep{n} [-v|-u]
```

- n* A number corresponding to the report: 1, 2, 3, 4a, or 4b.
- `-v|-v` Display command version information and exit.
- `-u|-u` Display command usage information and exit.

Examples

1. Print report 03, User Calendar Listing:

```
rep3
```
2. Display usage information for the `rep2` command:

```
rep2 -u
```
3. 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.

```
rep7 [-v|-u]
rep7 [-c cpu] [-s sched] [-j job] [-f date -t date]
```

- | | |
|------------------------|---|
| -v -v | Display command version information and exit. |
| -u -u | Display command usage information and exit. |
| -c <i>cpu</i> | The name of the <i>cpu</i> on which the jobs run. The default is all <i>cpus</i> . |
| -s <i>sched</i> | The name of the schedule in which the jobs run. The default is all schedules. |
| -j <i>job</i> | The name of a job. The default is all jobs. |
| -f <i>date</i> | Print job history from this date forward. The date is entered as: <i>yyyymmdd</i> . The default is the earliest available date. |
| -t <i>date</i> | Print job history up to this date. The date is entered as: <i>yyyymmdd</i> . The default is the most recent date. |

Examples

1. Display version information for the `rep7` command:

```
rep7 -v
```

2. Print all job history for *cpu* `ux3`:

```
rep7 -c ux3
```

3. Print all job history for all jobs in schedule `sked25`:

```
rep7 -s sked25
```

4. Print job history for all jobs in schedule `mysked` on *cpu* `x15` between 1/21/98 and 1/25/98:

```
rep7 -c x15 -s mysked -f 19980121 -t 19980125
```

5. Print two copies of all job history for *cpu* `ux2` on printer `lp2`:

```
MAESTROLP="| lp -dlp2 -n2"
export MAESTROLP
rep7 -c ux2
```

rep8

This command prints Report 08-Job Histogram.

```
rep8 [-v|-u]
rep8 [-f date -b time -t date -e time] [-i file] [-p]]
      [-b time -e time] ]
```

- v|-V** Display command version information and exit.
- u|-U** Display command usage information and exit.
- f date** Print job history from this date. Entered in the form *yyyymmdd*. The default is today.
- b time** Print job history beginning at this time. Entered in the form *hhmm*. The default is Maestro's start-of-day time (a Global Option).
- t date** Print job history to this date. Entered in the form *yyyymmdd*. The default is today.
- e time** Print job history ending at this time. Entered in the form *hhmm*. The default is Maestro's start-of-day time (a Global Option).
- i file** The name of the log file from which job history will be extracted. Note that log files are stored in the `schedlog` directory. The default is the current `symphony` file.
- p** Insert a page eject after each run date.

Examples

1. Display version information for the `rep8` command:


```
rep8 -v
```
2. Print a job histogram which includes all information in the current `symphony` file:


```
rep8
```

3. Print a job histogram beginning at 6:00 a.m. on 1/22/98, and ending at 5:59 a.m. on 1/26/98. This assumes that the dates requested are included in the specified log file. If some dates in the range are missing, the report contains only those available in the log file. Print the report with page ejects after each date:

```
rep8 -p -f 19980122 -b 0600 -t 19980126 -e 0559 -i schedlog/M199801260601
```

4. Print a job histogram, from the current `symphony` file, beginning at 6:00 am, and ending at 10:00 pm. Print the report on printer lp3:

```
MAESTROLP="| lp -dlp3"  
export MAESTROLP  
rep8 -b 0600 -e 2200
```

rep11

This command prints Report 11-Planned Production Schedule.

```
rep11 [-v|-u]
rep11 [-m mm[yy] [...]] [-c cpu [...]] [-o file]
```

- v|-v Display command version information and exit.
- u|-u Display command usage information and exit.
- m The months to be reported:
 - mm* Two digit month of year.
 - yy* Two digit year. The default is the current year, or next year if you specify a month earlier than the current month.

If this option is omitted, the default is the current month and year.
- c The cpus to be reported. If omitted, the default is all cpus.
- o The output file. If omitted, the `MAESTROL_P` variable defines the output file. If `MAESTROL_P` is not set, the default is `stdout`.

Examples

1. Report on June, July, and August of 1997 for cpus main, site1 and sagent1, and direct output according to `MAESTROL_P`, if set, or `stdout`:

```
rep11 -m 0697 0797 0897 -c main site1 sagent1
```

2. Report on June, July, and August of this year for all cpus, and direct output to the file `r11out`:

```
rep11 -m 06 07 08 -o r11out
```

3. Report on this month and year for cpu site2, and direct output according to `MAESTROL_P`, if set, or `stdout`:

```
rep11 -c site2
```

reptr

This command prints the following reports:

Report 09A	Planned Production Summary
Report 09B or 09D	Planned Production Detail (09D is printed instead of 09B when Maestro is running in Expanded Version mode, see page 2-3 for more information)
Report 10A	Actual Production Summary
Report 10B	Actual Production Detail.

```
reptr -v|-u|
reptr -pre [-{summary|detail}] [symfile]
reptr -post [-{summary|detail}] [logfile]
```

-v -V	Display the command version and exit.
-u -U	Display command usage information and exit.
-pre	Print the pre-production reports (09A, 09B, 09D).
-post	Print the post-production reports (10A, 10B).
-summary	Print the summary reports (09A, 10A). If -summary and -detail are omitted, both sets of reports are printed.
-detail	Print the detail reports (09B, 09D, 10B). If -summary and -detail are omitted, both sets of reports are printed.
<i>symfile</i>	For pre-production reports, the name of the Production Control file from which reports will be printed. The default is symnew in the current directory.
<i>logfile</i>	For post-production reports, the name of the Production Control log file from which the reports will be printed. Note that log files are stored in the schelog directory. The default is the symphony file.

Usage

If the command is entered with no options, both pre and post reports are printed. If **-summary** and **-detail** are omitted, both sets of reports are printed.

Examples

1. Print the pre-production detail report from the `symnew` file:
`reptr -pre -detail`
2. Print the pre-production summary report from the file `mysym`:
`reptr -pre -summary mysym`
3. Print the pre-production detail and summary reports from the `symnew` file:
`reptr -pre`
4. Print the post-production summary report from the log file `M199703170935`:
`reptr -post -summary schedlog/M199703170935`

xref

This command prints Report 12-Cross Reference Report.

```
xref -v|-U
xref [-cpu cpu] [-depends ] [...]
                    [-files ]
                    [-jobs ]
                    [-prompts ]
                    [-resource ]
                    [-schedules ]
                    [-when ]
```

- v** Display the command version and exit.
- u** Display command usage information and exit.
- cpu** Print the report for the named *cpu*. The @ wildcard is permitted, in which case, information from all qualified cpus is included. If omitted, all cpus are selected.
- depends** Print a report showing the schedules and jobs that are dependent on each job (based on Follows dependencies).
- files** Print a report showing the schedules and jobs that are dependent on each file (based on Opens files dependencies).
- jobs** Print a report showing the schedules in which each job appears.
- prompts** Print a report showing the schedules and jobs that are dependent on each prompt (based on Prompt dependencies).
- resource** Print a report showing the schedules and jobs that are dependent on each resource (based on Needs resources dependencies).
- schedules** Print a report showing the schedules and jobs that are dependent on each schedule (based on Follows dependencies).
- when** Print a report showing schedule On and Except dates.

Usage

If the command is entered with no options, all cpus and all options are selected.

Examples

1. Print a report for all cpus showing all cross-reference information:

```
xref
```

2. Print a report for all cpus. Include cross-reference information for all Follows dependencies:

```
xref -cpu @ -depends -schedules
```

Sample Reports

Report 01-Job Details Listing

```
MAESTRO for UNIX (HPUX)/REPORT1 5.0                               Tivoli                               Page 1
Report 01                                                         Job Details Listing                 01/09/98

Job: DEMOCPU #APJOB1      Description: Demo job.
JCL File                  : /development/techpubs/maestro/demo/japjob1
Logon                     : maestrtd                      Creator: maestrtd
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 12/19/97 at 14:11)
Maximum                      00:00:01              0 (On 12/19/97)
Minimum                      00:00:01              0 (On 12/19/97)

Job: DEMOCPU #APJOB2      Description: Demo job.
JCL File                  : /development/techpubs/maestro/demo/japjob2
Logon                     : maestrtd                      Creator: maestrtd
Recovery Job              :
Recovery Type             : STOP
Recovery Prompt           :
Composer Autodoc          : Yes
Total Runs                 :      1 -      1      Successful,      0 Aborted

                               Elapsed(mins)      CPU(secs)
Total                       00:00:02              0
Normal                       00:00:02
Last Run                     00:00:02              0 (On 12/19/97 at 14:11)
Maximum                      00:00:02              0 (On 12/19/97)
Minimum                      00:00:02              0 (On 12/19/97)
```

For an explanation of Elapsed and Cpu times, see [*logman Command*](#) on page 3-23.

Report 02–Prompt Messages Listing

MAESTRO for UNIX (HPUX)/REPORT2 5.0
Report 02

Tivoli
Prompt Message Listing

Page 1
01/09/98

Prompt	Message
PRMT1	Ready for job4? (Y/N)
PRMT2	:job4 launched okay
PRMT3	!continue?

Total number of prompts on file: 3

Report 03–User Calendar Listing

MAESTRO for UNIX (HPUX)/REPORT3 5.0
Report 03

Tivoli
User Calendar Listing

Page 1
01/09/97

Calendar Type: MONTHEND

Description:

Jan 1997							Feb 1997							Mar 1997							Apr 1997							May 1997							Jun 1997																												
S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S																						
.									
.									
.									
.	31	28	30	31	30	.
Jul 1997							Aug 1997							Sep 1997							Oct 1997							Nov 1997							Dec 1997																												
S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S	S	M	Tu	W	Th	F	S																						
.									
.									
.									
.	31	31	31	.							
Jan 1998																																																															
S	M	Tu	W	Th	F	S																																																									
.																																																									
.																																																									
.																																																									
.																																																									
.	31																																																									

Report 04A–User Parameters Listing

MAESTRO for UNIX (HPUX)/REPORT4A 5.0
Report 4A

Tivoli
User Parameter Listing

Page 1
01/09/98

Parameter Name	Contents
APATH	/fin/ap/erron/fcl/k009/anrig
DEMOP1	1
PONUM	25
PRINT	/dev/lp4m

Number of Parameters on file: 4

Report 04B–Maestro Resource Listing

MAESTRO for UNIX (HPUX)/REPORT4B 5.0 (3.17)
Report 4B

Tivoli
Maestro Resources List

Page 1
01/09/98

Resource CPU	Name	Number Avail	Description
DEMOCPU	#DATTAPES	1	DAT tape units
DEMOCPU	#QUKTAPES	2	Quick tape units
DEMOCPU	#TAPES	2	two tape units
FTAGENT	#JOBSLOTS	24	Job slots
SAGENT	#JOBSLOTS	12	Job slots

Number of Resources on file: 5

Report 07–Job History Listing

MAESTRO for UNIX (HPUX)/REPORT7 5.0 Report 07		Tivoli Job History Listing				Page 1 01/09/98
Date	Time	Schedule Name	Elapsed	CPU	Status	
Job:DEMOCPU #APJOB1 12/19/97	14:11	Runs: Aborted DEMOCPU #APAY	0 Successful 1	1	Elapsed Time: Normal 0 SU	1 Min 1 Max 1
Job:DEMOCPU #APJOB2 12/19/97	14:11	Runs: Aborted DEMOCPU #APAY	0 Successful 2	1	Elapsed Time: Normal 0 SU	2 Min 2 Max 2
Job:DEMOCPU #ARJOB1 12/19/97	13:29	Runs: Aborted DEMOCPU #AREC	0 Successful 1	1	Elapsed Time: Normal 0 SU	1 Min 1 Max 1
Job:DEMOCPU #ARJOB2 12/19/97	13:30	Runs: Aborted DEMOCPU #AREC	0 Successful 1	1	Elapsed Time: Normal 0 SU	1 Min 1 Max 1
Job:DEMOCPU #GLJOB1 12/19/97	14:51	Runs: Aborted DEMOCPU #CF9963AB	2 Successful 1	0	Elapsed Time: Normal 0 AB	0 Min 1 Max 2
12/19/97	14:55	DEMOCPU #CF9963AB	1	0	AB	

Report 08-Job Histogram

MAESTRO for UNIX (HPUX)/REPORTS 5.0 Tivoli Page 1
 Report 08 Job Histogram 12/19/96 06:00 - 12/20/96 05:59 01/09/97
 Interval Per Column: 15 minutes

CPU is DEMOCPU	0	0	0	1	1	1	1	1	1	2	2	0	0	0	0	0	
	6	7	9	0	2	3	5	6	8	9	1	2	0	1	3	4	5
Job	0	3	0	3	0	3	0	3	0	3	0	3	0	3	0	3	5
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9

12/19/96

```

AREC .ARJOB1 SU          *...
AREC .ARJOB2 SU          .....*
JOBS .TOUCH AB           *...
JOBS .MKFILE1 AB        *...
JOBS .MKFILE2 AB        *...
JOBS .MKFILE4 SU        *...
APAY .APJOB1 SU          *...
APAY .APJOB2 SU          *...
GL   .GLJOB1 AB          .....*
JOBS .RMPFILE2 SU       *...
    
```

**** End of Report ****

Report 09B-Planned Production Detail

Job Name	Estimated Run Time	Pri	Start Time	Until	Every	Limit Dependencies
MAESTRO for UNIX (HPUX)/REPORTER 5.0						
Report 09B Symnew		Tivoli			Page 1	
			Planned Production Detail For 01/09/98			
01/09/98						
Schedule DEMOCPU #MYSKED1	00:02	10				
MYJOB2	00:01	10				SAGENT#MYJOB1, 2-TAPES-
MYJOB3	00:01	10				SAGENT#MYJOB1, 2-TAPES-
(SAGENT #)MYJOB1		10				
Total	00:02					
Schedule DEMOCPU #FINAL	00:01	10	05:59(01/10/98)			
JNEXTDAY	00:01	10				
Total	00:01					
Schedule DEMOCPU #GL	00:02	10				FTAGENT#APAY, SAGENT#AREC
GLJOB1		10				
GLJOB2	00:02	10				GLJOB1
Total	00:02					
Schedule DEMOCPU #MYSKED2	00:03	10				MYSKED1
MYJOB1	00:01	10				
MYJOB2	00:01	10				
MYJOB3	00:01	10				
Total	00:03					
Schedule DEMOCPU #CLASSKED		10				GL
CLASSJOB		10				
Total	00:00					
Total	00:08					

Report 10B-Actual Production Detail

MAESTRO for UNIX (HPUX)/REPORTER 5.0 Tivoli Page 1
 Report 10B M199801091908 Actual Production Detail For 12/30/97 01/09/98

Job Name	Estimated Run Time	Priority	Start Time	Actual Run Time	CPU Seconds	Job Number	Status	
Schedule DEMOCPU #CF9963AA	00:01	10	13:07 (12/30/97)				SUCC	[FINAL]
JNEXTDAY	00:01	GO	13:07 (12/30/97)	00:01	4	#J9814	SUCC	
Total	00:01			00:01	4			
Schedule DEMOCPU #CF9963AB	00:02	10		00:01			HOLD	[GL]
GLJOB1		10					HOLD	
GLJOB2	00:02	10					HOLD	
Total	00:02			00:00	0			
Schedule DEMOCPU #MYSKED1	00:02	10					READY	
MYJOB2	00:01	10					HOLD	
MYJOB3	00:01	10					HOLD	
(SAGENT #)MYJOB1	00:01	10					READY	
Total	00:02			00:00	0			
Schedule DEMOCPU #FINAL	00:01	10					HOLD	
JNEXTDAY	00:01	10					HOLD	
Total	00:01			00:00	0			
Schedule DEMOCPU #GL	00:02	10					HOLD	
GLJOB1		10					HOLD	
GLJOB2	00:02	10					HOLD	
Total	00:02			00:00	0			
Schedule DEMOCPU #MYSKED2	00:03	10					HOLD	
MYJOB1	00:01	10					HOLD	
MYJOB2	00:01	10					HOLD	
MYJOB3	00:01	10					HOLD	
Total	00:03			00:00	0			

Report 12–Cross Reference Report

MAESTRO for UNIX (HPUX)/CROSSREF 5.0
Report 12
CPU: DEMOCPU

Tivoli
Cross Reference Report for Job Dependencies.

Page 1
01/09/98

Job Name	Dependencies
GL#GLJOB1	GL.GLJOB2
MYSKED1#MYJOB1	MYSKED1.MYJOB2, MYSKED1.MYJOB3
APAY#APJOB1	APAY.APJOB2
AREC#ARJOB1	AREC.ARJOB2

Report Extract Programs

Data extraction programs are used to generate several of Maestro's reports. The programs are:

<code>jbextract</code>	Used for Report 01-Job Details Listing and Report 07-Job History Listing
<code>prxtract</code>	Used for Report 02-Prompt Messages Listing
<code>caxtract</code>	Used for Report 03-User Calendar Listing
<code>paxtract</code>	Used for Report 4A-User Parameters Listing
<code>reextract</code>	Used for Report 4B- Maestro Resource Listing
<code>r11xtr</code>	Used for Report 11-Planned Production Schedule
<code>xxtract</code>	Used for Report 12-Cross Reference Report

Offline Output

The output of the extract programs is controlled by the `MAESTRO_OUTPUT_STYLE` variable, which defines the way long object names are handled. See [page 7-2](#) for more information.

jbextract Program

Extract job information.

```
jbextract [-v|-u] [-j job] [-c cpu] [-o file]
```

- `-v|-v` Display program version and exit.
- `-u|-u` Display program usage information and exit.
- `-j job` The job for which extraction is performed. The default is all jobs.
- `-c cpu` The cpu for which extraction is performed. The default is all cpus.
- `-o file` The output file. If omitted, output is directed to `stdout`.

Example

Report on job myjob for cpu main, and direct output to the file `jinfo`:

```
jbextract -j myjob -c main -o jinfo
```

jbextract Output Format

Each job record contains the following variable length fields, tab delimited:

Field	Description	Max Length (bytes) *
1	cpu name	8/16
2	job name	8/16
3	job script file name	256/4096
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 (<i>yyymmdd</i>)	8
17	last run time (<i>hhmm</i>)	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 (<i>yyymmdd</i>)	8
23	minimum cpu seconds	8
24	minimum elapsed time	8
25	minimum run date (<i>yyymmdd</i>)	8
* non-expanded databases/expanded databases		

prxtract Program

Extract prompt information.

```
prxtract [-v|-u] [-o file]
```

- v|-v Display program version and exit.
- u|-u Display program usage information and exit.
- o *file* The output file. If omitted, the output is directed to `stdout`.

Example

List all global prompt definitions, and direct output to the file `prinfo`:

```
prxtract -o prinfo
```

prxtract Output Format

Each prompt record contains the following variable length fields, tab delimited:

Field	Description	Max Length (bytes)
1	prompt name	8
2	prompt text	200

caxtract Program

Extract calendar information.

```
caxtract [-v|-u] [-o file]
```

- v|-v Display program version and exit.
- u|-u Display program usage information and exit.
- o *file* The output file. If omitted, the output is directed to `stdout`.

Example

List all calendar definitions, and direct output to the file `cainfo`:

```
caxtract -o cainfo
```

caxtract Output Format

Each calendar record contains the following variable length fields, tab delimited:

Field	Description	Max Length (bytes)
1	calendar name	8
2	calendar description	64

paxtract Program

Extract parameter information.

```
paxtract [-v|-u] [-o file]
```

- `-v|-v` Display program version and exit.
- `-u|-u` Display program usage information and exit.
- `-o file` The output file. If omitted, the output is directed to `stdout`.

Example

List all parameter definitions, and direct output to the file `painfo`:

```
paxtract -o painfo
```

paxtract Output Format

Each parameter record contains the following variable length fields, tab delimited:

Field	Description	Max Length (bytes)
1	parameter name	8
2	parameter value	64

reextract Program

Extract resource information.

```
reextract [-v|-u] [-o file]
```

- v|-V Display program version and exit.
- u|-U Display program usage information and exit.
- o file The output file. If omitted, the output is directed to `stdout`.

Example

List all resource definitions, and direct output to the file `reinfo`:

```
reextract -o reinfo
```

reextract Output Format

Each resource record contains the following variable length fields, tab delimited:

Field	Description	Max Length (bytes) *
1	cpu name	8/16
2	resource name	8
3	total resource units	4
4	resource description	72
* non-expanded databases/expanded databases		

r11xtr Program

Extract schedule information.

```
r11xtr [-v|-u] [-m mm[yy]] [-c cpu] [-o file]
```

- v|-v** Display program version and exit.
- u|-u** Display program usage information and exit.
- m** Month to be reported:
- mm* Two digit month of year.
- yy* Two digit year. Default is current year, or the next year if you specify a month earlier than the current month.
- If omitted, the default is current month, current year.
- c *cpu*** Cpu to be reported. If omitted, the default is your login cpu.
- o *file*** The output file. If omitted, the output goes to `stdout`.

Examples

1. Report on June of 1994 for cpus main, fta1 and sagent1, and direct output to `stdout`:

```
r11xtr -m 0694 -c main fta1 sagent1
```
2. Report on June of this year for all cpus, and direct output to file `r11out`:

```
r11xtr -m 06 -o r11out
```

r11xtr Output Format

Each schedule record contains the following variable length fields, tab delimited:

Field	Description	Max Length (bytes) *
1	cpu name	8/16
2	schedule name	8/16
3	schedule date (<i>yyymmdd</i>)	6
4	estimated cpu seconds	6
5	multiple cpu flag (* means some jobs run on other cpus)	1
6	number of jobs	4
7	day of week (Su, Mo, Tu, We, Th, Fr, Sa)	2
* non-expanded databases/expanded databases		

xrxtct Program

Extract cross reference information.

```
xrxtct [-v|-u]
```

-v|-v Display program version and exit.

-u|-u Display program usage information and exit.

Example

Extract cross-reference information, and direct output to eight files (see *xrxtct Output Format* below):

```
xrxtct
```

xrxtct Output Format

The program's output is written to eight files, listed and described below.

xsched

This file contains information about schedules. Each schedule record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"00"	2
2	cpu name	8/16
3	schedule name	8/16
4	unused	248
5	cpu name (same as 2 above)	8/16
6	schedule name (same as 3 above)	8/16
7	unused	8
8	unused	8
9	unused	6
10	unused	6
11	unused	8
12	end-of-record (null)	1
* non-expanded databases/expanded databases		

xwhen

This file contains information about when schedules will run. Each schedule record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"01"	2
2	cpu name	8/16
3	ON/EXCEPT name or date	8
4	except flag (*=EXCEPT)	1
5	unused	247
6	cpu name	8/16
7	schedule name	8/16
8	unused	8
9	unused	8
10	unused	6
11	offset num	6
12	offset unit	8
13	end-of-record (null)	1
* non-expanded databases/expanded databases		

xdep_sched

This file contains information about schedules that are dependent on a schedule. Each dependent schedule record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"02"	2
2	cpu name	8/16
3	schedule name	8/16
4	unused	248
5	dependent schedule cpu name	8/16
6	dependent schedule name	8/16
7	unused	8
8	unused	8
9	unused	6
10	unused	6
11	unused	8
12	end-of-record (null)	1
* non-expanded databases/expanded databases		

xdep_job

This file is split into two record types. The first contains information about jobs and schedules that are dependent on a job. Each dependent job and schedule record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"03"	2
2	cpu name	8/16
3	job name	8/40
4	schedule name	8/16
5	unused	240
6	dependent schedule cpu name	8/16
7	dependent schedule name	8/16
8	dependent job cpu name	8/16
9	dependent job name	8/40
10	unused	6
11	unused	6
12	unused	8
13	end-of-record (null)	1
* non-expanded databases/expanded databases		

The second record type contains information about jobs and schedules that are dependent on an internetwork dependency. Each dependent job and schedule record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"08"	2
2	cpu name	8/16
3	job name	120
4	unused	128
5	dependent schedule cpu name	8/16
6	dependent schedule name	8/16
7	dependent job cpu name	8/16
8	dependent job name	8/40
9	unused	6
10	unused	6
11	unused	8
12	end-of-record (null)	1
* non-expanded databases/expanded databases		

Note that all internetwork dependencies are treated as jobs in the local Conman, therefore both remote job and schedule dependencies are listed as jobs in the file. See appendix D, *Internetwork Dependencies* for more information.

xjob

This file contains information about the schedules in which each job appears. Each job record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"04"	2
2	cpu name	8/16
3	job name	8/40
4	unused	248
5	schedule cpu name	8/16
6	schedule name	8/16
7	unused	8
8	unused	8
9	unused	6
10	unused	6
11	unused	8
12	end-of-record (null)	1
* non-expanded databases/expanded databases		

xprompt

This file contains information about jobs and schedules that are dependent on a prompt. Each prompt record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"05"	2
2	cpu name	8/16
3	prompt name or text	20
4	unused	236
5	dependent schedule cpu name	8/16
6	dependent schedule name	8/16
7	dependent job cpu name	8/16
8	dependent job name	8/40
9	unused	6
10	unused	6
11	unused	8
12	end-of-record (null)	1
* non-expanded databases/expanded databases		

xresource

This file contains information about jobs and schedules that are dependent on a resource. Each resource record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"06"	2
2	cpu name	8/16
3	resource name	8
4	unused	248
5	dependent schedule cpu name	8/16
6	dependent schedule name	8/16
7	dependent job cpu name	8/16
8	dependent job name	8/40
9	units allocated	6
10	unused	6
11	unused	8
12	end-of-record (null)	1
* non-expanded databases/expanded databases		

xfile

This file contains information about jobs and schedules that are dependent on a file. Each file record contains the following fixed length fields, with no delimiters:

Field	Description	Max Length (bytes) *
1	"07"	2
2	cpu name	8/16
3	file name	256
4	dependent schedule cpu name	8/16
5	dependent schedule name	8/16
6	dependent job cpu name	8/16
7	dependent job name	8/40
8	unused	6
9	unused	6
10	unused	8
11	end-of-record (null)	1
* non-expanded databases/expanded databases		

8

Composer Command Line Reference

The Composer program is used to write schedules and define scheduling objects (cpus, cpu classes, jobs, resources, prompts, calendars, and parameters). This reference section describes the command line version of Composer in three parts:

- running Composer and program basics,
- syntax for scheduling objects and schedules, and
- syntax and usage for Composer commands.

For a complete description of Composer, scheduling objects and schedules, refer to section 4, *Composing Scheduling Objects* and section 5, *Composing Schedules*.

Running Composer

Composer is run as follows:

```
composer ["command[&...][&]"]
```

Examples

1. Composer enters conversational mode and prompts for a command:

```
composer
```

2. Composer executes the `print` and `version` commands, and quits:

```
composer "p parms&v"
```

3. Composer executes the `print` and `version` commands, and then enters conversational mode and prompts for a command.:

```
composer "p parms&v&"
```

4. Composer reads commands from `cmdfile`:

```
composer < cmdfile
```

5. Commands from `cmdfile` are piped to Composer:

```
cat cmdfile | composer
```

Control Characters

The following control characters can be entered in conversational mode to interrupt Composer (if your `stty` settings are configured as such).

Control c Composer stops executing the current command at the next interruptible step, and returns a command prompt.

Control d Composer quits after executing the current command.

Terminal Output

Output to your terminal is controlled by shell variables, `MAESTROLINES` and `MAESTROCOLUMNS`. If either 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 less, Composer does not pause at the end of a page.

Offline Output

The `;offline` option in Composer commands is generally used to print the output of a command. When you include it, the following shell variables control the output:

<code>\$MAESTROLFP</code>	The destination of a command's output. You can set it to any of the following:
<code>> file</code>	Redirect output to a file, overwriting the contents of the file. If the file does not exist it is created.
<code>>> file</code>	Redirect output to a file, appending to the end of file. If the file does not exist it is created.

	<i>command</i>	Pipe output to a system command or process. The system command is always executed.
	<i>command</i>	Pipe output to a system command or process. The system command is not executed if there is no output.
		If not set, the default is " lp -tCONLIST".
\$MAESTROLPLINES		The number of lines per page. If not set, the default is 60.
\$MAESTROLPCOLUMNS		The number of characters per line. If not set, the default is 132.

The variables must be exported before running Composer.

Composer's Editor

Several of Composer's commands automatically invoke a text editor. The specific 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, the **vi** editor is invoked by default.

Composer's Command Prompt

The Composer command prompt is, by default, a dash (-). This is defined in the message catalog file *maestrohome/catalog/maestro.msg*. To select a different prompt, edit the file, locate set 208, message number 01, and change the character on that line. The prompt can be up to ten characters in length, not including the required trailing pound sign (#). After changing the prompt, generate a new catalog file as follows:

```
cd maestrohome/catalog
gencat maestro.cat maestro.msg
```

Defining and Managing Objects and Schedules

For command line syntax regarding scheduling objects, see [Managing Objects](#) on page 8-31. For scheduling language syntax see [The Scheduling Language](#) on page 8-50.

Composer Commands

Composer command syntax consists of the following elements:

```
command selection options
```

command The command name.

selection The object or set of objects to be acted upon.

options The command options.

Wildcards

The following wildcard characters are permitted in some of Composer's commands.

- @ Replaces one or more alphanumeric characters.
- ? Replaces one alphabetic character.
- % Replaces one numeric character.

Delimiters and Special Characters

The following characters have special meanings in composer commands:

&	Command delimiter. See <i>Running Composer</i> on page 8-1.
;	Option delimiter. For example: <code>;info;offline</code>
=	Value delimiter. For example: <code>sched=sked5</code>
: !	Command prefixes which cause the command to be passed on to the system. These prefixes are optional—if Composer does not recognize the command, it is passed automatically to the system. For example: <code>!ls</code> or <code>:ls</code>
<<>>	Comment brackets. Comments can be enclosed in double less than (<<) and greater than (>>) signs on a single line anywhere in a schedule. For example: <code>schedule foo <<comment>> on everyday</code>
*	Comment prefix. The prefix must be the first character on a command line or following a command delimiter. For example: <code>*comment</code> or <code>print&*comment</code>

>	Redirect command output to a file, overwriting the contents of the file. If the file does not exist it is created. For example: <code>display parms > parmlist</code>
>>	Redirect command output to a file, appending to the end of file. If the file does not exist it is created. For example: <code>display parms >> parmlist</code>
	Pipe command output to a system command or process. The system command is always executed. For example: <code>display parms grep alparm</code>
	Pipe command output to a system command or process. The system command is not executed if there is no output. For example: <code>display parms grep alparm</code>

Command Descriptions

The following pages describe Composer's commands.

List of Commands

Command	Description	Page
<code>add</code>	Add jobs, schedules, users, and cpus or cpu classes.	page 8-7
<code>build</code>	Build or rebuild a Maestro file.	page 8-8
<code>command</code>	Pass a system command to the system.	page 8-9
<code>continue</code>	Ignore the next error.	page 8-10
<code>create</code>	Create a file for editing.	page 8-11
<code>delete</code>	Delete jobs, schedules, users, and cpus.	page 8-13
<code>display</code>	Display scheduling objects and cpus or cpu classes.	page 8-15
<code>edit</code>	Edit a file.	page 8-19
<code>exit</code>	Terminate Composer.	page 8-20
<code>help</code>	Display command information.	page 8-21
<code>list</code>	List scheduling objects and cpus or cpu classes.	page 8-15
<code>modify</code>	Modify scheduling objects and cpus or cpu classes.	page 8-22
<code>new</code>	Edit and add scheduling objects and cpus or cpu classes.	page 8-24
<code>print</code>	Print scheduling objects and cpus or cpu classes.	page 8-15
<code>redo</code>	Edit the previous command.	page 8-25
<code>replace</code>	Replace an existing job, schedule, user, cpu or cpu class.	page 8-27
<code>validate</code>	Validate a file.	page 8-28
<code>version</code>	Display Composer's program banner.	page 8-30

Command names and keywords can be entered in either uppercase or lowercase. They can also be abbreviated to as few leading characters as needed to distinguish them from one another.

add

Add jobs, schedules, cpus and cpu classes, and users.

Security: You must have **add** access for new objects. If an object already exists, you must have **modify** access to the object.

```
add file
```

file The name of a file containing:

- One or more jobs. The first line of the file must be `$JOBS`.
- One or more schedules.
- One or more cpu or cpu class definitions.
- One or more user definitions.

Operation

The file is always syntax-checked before its contents are written to the master file. All errors and warnings are reported. If there are syntax errors, you are asked whether or not you wish to edit the file to make corrections. If the object you are adding already exists, you are asked whether or not to replace it.

Examples

1. Add the jobs from the file `myjobs` to the master job file:

```
add myjobs
```

2. Add the schedules from the file `mysked` to the master schedule file:

```
a mysked
```

3. Add the cpu and cpu class definitions from the file `cpus.src` to the master cpu file:

```
a cpus.src
```

4. Add the user definitions from the file `users_nt` to the master user file:

```
a users_nt
```

build

Build or rebuild Maestro's master file.

Security: You must have `build` access to the master file.

```
build object_file
```

Where *object_file* can be:

`calendars` The Master Calendar file.

`cpudata` The Master Cpu file.

`jobs` The Master Job file.

`mastsked` The Master Schedule file. This will also build the `jobs.sched` file.

`parms` The Master Parameter file.

`prompts` The Master Prompt file.

`resources` The Master Resource file.

`userdata` The Master User file.

Operation

If the master file does not exist, it is created. If the file exists, it is rebuilt. Rebuilding a master file can be useful when it becomes fragmented from numerous additions and deletions. The rebuild will remove unused records and optimize the keys.

Examples

1. Rebuild the master schedule and `jobs.sched` files:

```
build mastsked
```

2. Rebuild the master calendars file:

```
build calendars
```

3. Rebuild the master cpu file:

```
b cpudata
```

Command

Execute a system command.

```
[ : | ! ] command
```

command Any valid system command. The prefix (: or !) is required only if a system command is spelled the same as a Composer command.

Operation

The system command is executed by the shell defined by the `SHELL` variable. The shell's effective user id is set to the id of the user running Composer to prevent security breaches. If the `SHELL` variable is not set, the command is executed by `/bin/sh`.

Examples

```
date -u  
!help 102
```

continue

Ignore the next command error.

```
continue
```

Operation

This command is useful when multiple commands are entered on the command line, or redirected from a file. It instructs Composer to continue executing commands even if the next command, following `continue`, results in an error. This command is not needed when you enter commands interactively since Composer will not quit on an error.

Example

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

create

Create an editable file.

Security: You must have display access to the object being copied.

```

create file from {calendars          }
                  {cpu=cpu           }
                  {jobs=[cpu#] job   }
                  {parms             }
                  {prompts           }
                  {resources         }
                  {sched=[cpu#] sched }
                  {users=[cpu#] user  }

```

<i>file</i>	The name of the file. Unless otherwise specified, the file is created in your current directory. You will get an error message if you do not have write permission.
calendars	Copy all calendars into <i>file</i> .
cpu=	Copy a cpu definition into <i>file</i> .
<i>cpu</i>	The name of the cpu or cpu class. Wildcards are permitted.
jobs=	Copy a job definition into <i>file</i> .
<i>cpu</i>	The name of the cpu or cpu class on which the job is defined (the pound sign is a required delimiter). Wildcards are permitted.
<i>job</i>	The name of the job. Wildcards are permitted.
parms	Copy all parameters into <i>file</i> .
prompts	Copy all prompts into <i>file</i> .
resources	Copy all resources into <i>file</i> .
sched=	Copy a schedule definition into <i>file</i> .
<i>cpu</i>	The name of the cpu or cpu class on which the schedule runs (the pound sign is a required delimiter). Wildcards are permitted.
<i>sched</i>	The name of the schedule. Wildcards are permitted.
users=	Copy a user definition into <i>file</i> .

<i>cpu</i>	The name of the cpu on which the user is defined (the pound sign is a required delimiter). Wildcards are permitted.
<i>user</i>	The name of the user. Wildcards are permitted.

Note that the password field is empty for security purposes.

Operation

If the `cpu` or schedule definition specified in the `from` clause does not exist, the command quits with an error message. Once the file is created, you can use the `edit` command to make modifications. The `add` or `replace` command can then be used to add to or update a master file.

Examples

1. Create an edit file containing the current list of calendars:

```
create caltemp from calendars
```

2. Create an edit file containing the schedule sked12:

```
cr stemp from sched=sked12
```


delete

Delete scheduling objects.

Security: You must have `delete` access to the `schedule` or `cpu`.

```
delete {cpu=cpu           }
       {jobs=[cpu#] job   }
       {sched=[cpu#] sched }
       {users=[cpu#] user  }
```

cpu= Delete a cpu definition.

cpu The name of the cpu or cpu class. Wildcards are permitted.

jobs= Delete a job definition.

cpu The name of the cpu or cpu class on which the job is defined (the pound sign is a required delimiter). Wildcards are permitted.

job The name of the job. Wildcards are permitted.

sched= Delete a schedule definition.

cpu The name of the cpu or cpu class on which the schedule runs (the pound sign is a required delimiter). Wildcards are permitted.

sched The name of the schedule. Wildcards are permitted.

users= Delete a user definition.

cpu The name of the cpu on which the user is defined (the pound sign is a required delimiter). Wildcards are permitted.

user The name of the user. Wildcards are permitted.

Operation

If the specified definition does not exist, the command quits with an error message. If wildcards are used to specify a set of definitions, Composer prompts for confirmation before deleting each matching definition. If no matches are found, the command quits with an error message.

Examples

1. Delete job job3 that is launched on cpu site3:

```
delete jobs=site3#job3
```

2. Delete all cpus whose names begin with "ux":

```
de cpu=ux@
```

3. Delete all schedules that begin with "test" on all cpus:

```
de sched=@#test@
```

display, list, print

Display, list, or print scheduling objects.

Security: For **display** and **print**, you must have **display** access to the object.

```

{display } {calendars }
{list } {cpu=cpu|class|domain[;offline] }
{print } {jobs=[cpu#]job[;offline] }
        {parms }
        {prompts }
        {resources }
        {sched=[cpu#]sched[;offline] }
        {users=[cpu#]user[;offline] }

```

calendars Display all calendars.

cpu= Display a cpu definition.

cpu The name of the cpu. Wildcards are permitted.

class The name of the cpu class. Wildcards are permitted.

domain The name of the Maestro network domain. Wildcards are permitted.

jobs= Display a job definition.

cpu The name of the cpu or cpu class on which the job is defined (the pound sign is a required delimiter). Wildcards are permitted.

job The name of the job. Wildcards are permitted.

parms Display all parameters.

prompts Display all prompts.

resources Display all resources.

sched= Display a schedule definition.

cpu The name of the cpu or cpu class on which the schedule runs (the pound sign is a required delimiter). Wildcards are permitted.

sched The name of the schedule. Wildcards are permitted.

users= Display a user definition.

cpu The name of the cpu on which the user is defined(the pound sign is a required delimiter). Wildcards are permitted.

user The name of the user. Wildcards are permitted.

For security reasons, user passwords are not displayed.

offline For **display** and **list** commands only. The output of the command is controlled by the shell variable **MAESTROLF**. See *Offline Output* on page 8-2 for more information.

Operation

In the **list** command, only the object names are listed. The output of the **print** command is controlled by the shell variable **MAESTROLF**. See *Offline Output* on page 8-2 for more information.

Output Format—calendars

Calendar The calendar name.

Description The free-form description of the calendar.

Following these fields is a list of calendar dates.

Output Format—cpu

CPU id The cpu or cpu class name.

Creator The name of the user who created the cpu definition.

Last Updated The date the cpu definition was last updated.

Following these fields is the cpu or cpu class definition.

Output Format—jobs

CPU id The cpu name on which the job runs.

Job The name of the job.

Logon The name of the logon user for the job.

Last Run Date The date the job was last run.

Following these fields is the job definition.

Output Format—parms

Parameter	The parameter name.
Value	The value of the parameter.

Output Format—prompts

Prompt	The prompt name.
Message	The prompt message text.

Output Format—resources

CPU id	The cpu name on which the resource is defined.
Resource	The resource name.
No Avail	The total number of resource units.
Description	The free-form description of the resource.

Output Format—sched

CPU id	The cpu name on which the schedule runs.
Schedule	The name of the schedule.
Creator	The name of the user who created the schedule definition.
Last Updated	The date the schedule definition was last updated.

Following these fields is the schedule definition.

Output Format—users

CPU id	The cpu name on which the user is allowed to run jobs.
User	The name of the user.
Creator	The name of the user who created the user definition.
Last Updated	The date the user definition was last updated.

Following these fields is the user definition.

Examples

1. Display all calendars:

```
display calendars
```

2. Print all schedules that are launched on cpu site2:

```
print sched=site2#@
```

edit

Edit an existing file.

Security: You must have read and write access to the file.

```
edit file
```

file The name of a file.

Operation

An editor is started and the file is opened for editing. If the file does not exist, an error is reported and the command quits.

See *Composer's Editor* on page 8-3 for more information.

Examples

```
edit mytemp
```

```
ed file4
```

exit

Exit the Composer program.

```
exit
```

Operation

When in `help` mode, Composer returns to command input mode.

Examples

```
exit  
e
```

help

Display command help information.

```
help command
```

command The name of any Composer command. Enter the full command name, not abbreviated. Entering `commands` displays a list of Composer commands.

Examples

1. Display a list of all composer commands:

```
help commands
```

2. Display information about the display command:

```
h display
```

modify

Modify or add scheduling objects.

Security: You must have `modify` access to the object or the affected master file.

```

modify {calendars           }
        {cpu=cpu           }
        {jobs=[cpu#] job   }
        {parms            }
        {prompts         }
        {resources       }
        {sched=[cpu#] sched }
        {users=[cpu#] user }
    
```

calendars Modify the calendar list.

cpu= Modify a cpu or cpu class definition.

cpu The name of the cpu or cpu class. Wildcards are permitted.

jobs= Modify a job definition.

cpu The name of the cpu or cpu class on which the job is defined (the pound sign is a required delimiter). Wildcards are permitted.

job The name of the job. Wildcards are permitted.

parms Modify the parameter list.

prompts Modify the prompt list.

resources Modify the resource list.

sched= Modify a schedule definition.

cpu The name of the cpu or cpu class on which the schedule runs (the pound sign is a required delimiter). Wildcards are permitted.

sched The name of the schedule. Wildcards are permitted.

users= Modify a user definition.

cpu The name of the cpu on which the user is defined (the pound sign is a required delimiter). Wildcards are permitted.

user The name of the user. Wildcards are permitted.

Operation

The **modify** command copies the definition or object list into a temporary edit file, edits the file, and then adds the contents of the file to the appropriate master file. This is equivalent to the following sequence of commands:

```
create file from object-specification
edit file
add file
```

If the add operation is successful, the edit file is purged. For more information, refer to the descriptions of the above commands in this section.

For user definitions, if a password field remains empty when you exit the editor, the old password is retained. To specify a null password use two consecutive double quotes without any spaces in between (“”).

Examples

1. Modify the calendar list:

```
modify calendars
```
2. Modify schedule sked9 that is launched on cpu site1:

```
m sched=site1#sked9
```

new

Add new scheduling objects.

Security: You must have `add` access for new objects on the `cpu`. If an object already exists, you must have `modify` access to the object or the affected master file.

`new`

Operation

The `new` command creates a temporary edit file, edits the file, and then adds the contents of the file to the appropriate master file.

Note: For calendars, parameters, resources, and prompts, the contents of the temporary file added by the `new` command replace the entire contents of the existing file. See the `modify` command (described on [page 8-22](#)) to create a temporary edit file of the existing object file.

Examples

`new`

`n`

redo

Edit and re-execute the previous command.

```
redo
```

Directives

When you issue `redo`, Composer displays the previous command, so that it can be edited and re-executed. Use the spacebar to move the cursor under the character to be modified, and enter one of the following directives.

<code>d[<i>dir</i>]</code>	Delete the character above the <code>d</code> . This can be followed by other directives.
<code>i_{text}</code>	Insert <i>text</i> before the character above the <code>i</code> .
<code>r_{text}</code>	Replace one or more characters with <i>text</i> , beginning with the character above the <code>r</code> . Replace is implied if no other directive is entered.
<code>>_{text}</code>	Append <i>text</i> to the end of the line.
<code>d[<i>dir</i> <i>text</i>]</code>	Delete characters at the end of the line. This can be followed by another directive or <i>text</i> .
<code>>r_{text}</code>	Replace characters at the end of the line with <i>text</i> .

Directive Examples

<code>ddd</code>	Delete the three characters above the <code>d</code> 's.
<code>iabc</code>	Insert "abc" before the character above the <code>i</code> .
<code>rabc</code>	Replace the three characters, starting with the one above the <code>r</code> , with "abc".
<code>abc</code>	Replace the three characters above "abc" with "abc".
<code>d diabc</code>	Delete the character above the first <code>d</code> , skip one space, delete the character above the second <code>d</code> , and insert "abc" in its place.
<code>>abc</code>	Append "abc" to the end of the line.
<code>>ddabc</code>	Delete the last two characters in the line, and append "abc" in their place.
<code>>rabc</code>	Replace the last three characters in the line with "abc".

Examples

```
redo
display site1#sa@
  ip
display site1#sa@
```

(insert a character, press the Return key)
(press the Return key to execute command)

```
redo
display site1#sa@
  2
display site2#sa@
```

(implicit replace, press the Return key)
(press the Return key to execute command)

replace

Replace scheduling objects.

Security: You must have `modify` access to the object or the affected master file.

```
replace file
```

file The name of a file containing:

1. One or more schedules,
2. One or more jobs (where the first line in the file is `$JOBS`),
3. One or more domain, `cpu`, and/or `cpu class` definitions,
4. One or more user definitions,
5. A complete list of calendars, parameters, prompts or resources.

Operation

The `replace` command executes in the same manner as an `add` command, except that there is no prompt to replace existing objects. For more information, refer to [add](#) on page 8-7.

Examples

```
replace mysked
```

```
rep myprompts
```

validate

Validate a file.

```
validate file[;syntax]
```

file The name of a file containing Maestro scheduling objects: calendars, cpus, cpu classes, domains, jobs, parameters, prompts, resources, or schedules. To validate the production control file, enter **prodsked**.

syntax Check the file for syntax errors only.

Operation

The **validate** command performs the same syntax checks and validation that are performed when you add or modify Maestro objects. It can also be used to validate the production schedule file (**prodsked**), created during the pre-production phase of Maestro's processing day. **Prodsked** contains the schedules to be run on a particular day, and it can be modified with an editor to include ad hoc changes before the day's processing begins. **Validate** should be used in such cases to be certain that the production schedule is still valid.

For schedules, if the **syntax** option is omitted, validation and reporting include the following:

1. Verify job names against the master job file.
2. Examine dependencies to ensure that the objects exist. For example, a "needs" dependency on a non-existent resource will be reported. This will also check for references to non-existent calendars.
3. Check for circular dependencies. For example, "job1 follows job2, and job2 follows job1" is a circular dependency.

The output of the **validate** command can be redirected to a file as follows (from the UNIX command line only):

```
composer "validate filename" > outfile
```

If you wish to redirect the error messages to the output file in addition to the actual output, use the following (from the UNIX command line only):

```
composer "validate filename" > outfile 2>&1
```


Examples

1. Check the syntax of a file containing cpu definitions:

```
validate mycpus;syntax
```

2. Re-validate all schedules. This can be done to verify the integrity of references to other scheduling objects after changes have been made.

```
create allskeds from sched=@#@  
validate allskeds
```

version

Display Composer's program banner.

```
version
```

Examples

```
version
```

```
v
```

Managing Objects

Command line syntax for defining Maestro objects is explained on the pages that follow. For more information about Maestro objects and their options using the graphical interface section 4, [*Composing Scheduling Objects*](#).

Managing Master Files

Maestro objects are managed with the **composer** program. The objects are ultimately stored in master files, but they are managed using intermediate edit files. For example, when adding a new object, it is first written in an edit file, which is then syntax-checked and added to the appropriate master file.

Schedule, Job, User, and Cpu Definitions

The master files for schedules, jobs, users, and cpus (which includes cpu classes) are structured such that definitions can be managed individually. For example, to modify an existing schedule: 1) it is copied from the master schedule file into an edit file, 2) the schedule is edited, 3) the edit file is syntax-checked, and 4) the modified schedule replaces the existing schedule in the master schedule file.

Calendars, Parameters, Prompts, and Resources

The master files for calendars, parameters, prompts, and resources are managed as complete lists. For example, to modify an existing resource: 1) the entire resource list is copied from the master resource file into an edit file, 2) the line defining the resource is edited, 3) the edit file is syntax-checked, and 4) the new list overwrites the contents of the master resource file.

Cpu Definitions

A cpu is a processing object capable of running Maestro-scheduled jobs. Cpu definitions can be included in files, along with cpu class and domain definitions, that are added to Maestro's database with the Composer program. The cpu definition syntax is:

```
# comment
cpuname this-cpu [description "desc"]
os os-type
node this-node[domain] [tcpaddr this-tcpaddr]
[domain domainname]
[for maestro [host host-cpu [access method] ] [...]]
    [type {fta|s-agent|x-agent} ]
    [ignore ]
    [autolink {on|off} ]
    [fullstatus {on|off} ]
    [resolvedep {on|off} ]
    [server server ]
end
[cpuname ...]
[cpuclass ...]
[domain ...]
```

# <i>comment</i>	Treat everything from the pound sign to the end-of-line as a comment.
cpuname	The name of this cpu (up to 16 alphanumeric, dash (-), and underscore (_) characters, starting with a letter).
description	A free-form description of this cpu (up to 40 characters), enclosed in double quotes.
os	This cpu operating system. The valid entries are: MPEV , MPIX , UNIX , WNT , and OTHER .
node	The node name or the IP address of this cpu. Fully-qualified domain names are accepted.
tcpaddr	The TCP port number of Netman on this cpu. If omitted, the default is 31111.
domain	Enter the name of the Maestro domain of the cpu. The default for a fault-tolerant agent, or a standard agent is the master domain, usually named MASTERDM. The default for a

	domain manager is the domain in which it is defined as the manager- see <i>Domain Definitions</i> on page 8-37. The default for an extended agent is the domain of its host cpu.						
host	Required for extended agents. Enter the name of the agent's host cpu. The host is the Maestro cpu with which the extended agent communicates and where its access method resides. Note that the host cannot be another extended agent. See appendix A, <i>Maestro Networks</i> and appendix C, <i>Extended Agent Reference</i> for more information on extended agents.						
access	For extended and Network agents. Enter name of the access method. This must be the name of a file that resides in the <i>maestrohome/methods</i> directory on the agent's host cpu.						
type	Enter the cpu type as one of the following: <table> <tr> <td>fta</td> <td>Fault-tolerant agent, including domain managers, and backup domain managers.</td> </tr> <tr> <td>s-agent</td> <td>Standard agent.</td> </tr> <tr> <td>x-agent</td> <td>Extended agent.</td> </tr> </table>	fta	Fault-tolerant agent, including domain managers, and backup domain managers.	s-agent	Standard agent.	x-agent	Extended agent.
fta	Fault-tolerant agent, including domain managers, and backup domain managers.						
s-agent	Standard agent.						
x-agent	Extended agent.						
ignore	Include this keyword only if you want Maestro to ignore this cpu definition.						
autolink	For a fault-tolerant agent or standard agent, enter on to have the domain manager open a communications path to the agent when the domain manager is started. For the domain manager, enter on to have its agents open communications paths to the domain manager when they are started. Autolink is useful primarily during the initial start up sequence at the beginning of each day. At that time, a new production schedule is created and compiled on the master domain manager, and all cpus are stopped and restarted. For each agent with autolink turned on, the domain manager automatically opens a path to it, sends a copy of the new Production Control file, and starts the agent. If autolink is also turned on for the domain manager, the agent, in turn, opens a path back to the domain manager to complete the link. If autolink is off for an agent, it is initialized when you issue a Console Manager link command on the agent's domain manager or the master domain manager.						
fullstatus	For fault-tolerant agents only. Enter on to have the link from the domain manager operate in Full Status mode. In this mode, the agent is kept updated about the status of jobs and schedules running on other cpus in the network.						

If `fullstatus` is `off`, the agent is only informed about the status of jobs and schedules on other cpus that affect its own jobs and schedules. This can improve operation by reducing network traffic.

To keep the agent's Production Control file at the same level of detail as its domain manager set `fullstatus` and `resolvedep` to `on`. Always set these modes `on` for backup domain managers. See `resolvedep` [below](#).

`resolvedep`

For fault-tolerant agents only. Enter `on` to have the agent's Production Control process operate in Resolve All Dependencies Mode. In this mode, the agent tracks dependencies for its all jobs and schedules, including those running on other cpus. Note that `fullstatus` must also be `on` so that the agent is informed about activity on other cpus.

If `resolvedep` is `off`, the agent tracks dependencies for its own jobs and schedules only. This reduces cpu usage by limiting processing overhead.

To keep the agent's Production Control file at the same level of detail as its domain manager set `fullstatus` and `resolvedep` to `on`. Always set these modes `on` for backup domain managers. See `fullstatus` [above](#).

`server`

For fault-tolerant and standard agents only. Omit this option for domain managers. This identifies a server (**Mailman**) process on the domain manager that will send messages to the agent. This can be a letter or a number (A-Z or 0-9). The IDs are unique to each domain manager, so you can use the same IDs for agents in different domains without conflict. If more than 36 server ids are required in a domain, consider dividing it into two or more domains.

If a server ID is not specified, messages to a fault-tolerant or standard agent are handled by a single **Mailman** process on the domain manager. Entering a server ID causes the domain manager to create an additional **Mailman** process. The same server ID can be used for multiple agents. The use of servers reduces the time required to initialize agents, and generally improves the timeliness of messages. As a guide, additional servers should be defined to prevent a single **Mailman** from handling more than eight agents.

For more information about Maestro networks and cpu types, see [appendix A, *Maestro Networks*](#).

Examples

1. A master domain manager named `hdq-1`, and a fault-tolerant agent named `hdq-2` in the master domain (note that a domain definition is not required for the master domain--its name defaults to **masterdm**):

```
cpuname hdq-1 description "Headquarters master"
  os unix
  node sultan.unison.com
  domain masterdm
  for maestro type fta
    autolink on
    fullstatus on
    resolvedep on
end
cpuname hdq-2
  os wnt
  node opera.unison.com
  domain masterdm
  for maestro type fta
    autolink on
end
```

2. 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
  node pewter.unison.com
  domain distr-a
  for maestro type fta
    autolink on
    fullstatus on
    resolvedep on
end
cpuname distr-a2
  os wnt
  node quatro.unison.com
  domain distr-a
  for maestro type s-agent
    host distr-a1
end
```

Cpu Class Definitions

A cpu class is a group of Maestro cpus for which common schedules can be written. Cpu class definitions can be included in files, along with cpu and domain definitions, that are added to Maestro's database with the Composer program. The cpu class definition syntax is:

```
# comment
cpuclass cpuclass
    members
        cpuid|@
        ...
end
[cpuclass ...]
[domain ...]
[cpu ...]
```

- | | |
|------------------|--|
| # <i>comment</i> | Treat everything from the pound sign to the end-of-line as a comment. |
| <i>cpuclass</i> | The name of the cpu class (up to eight alphanumeric, dash (-), and underscore (_) characters, starting with a letter). |
| <i>cpuid @</i> | The Maestro names of the cpu's that are members of the class. The @ wildcard can be used to include all cpu's. |

Examples

1. The cpu class definition for a class named `backup`:

```
cpuclass backup
    members
        main
        site1
        site2
end
```

2. The cpu class definition for class named `allcpus` that contains every cpu:

```
cpuclass allcpus
    members
        @
end
```

Domain Definitions

A domain is a group of Maestro cpus consisting of one or more agents and a domain manager. The domain manager acts as the management hub for the agents in the domain. Domain definitions can be included in files, along with cpu and cpu class definitions, that are added to Maestro's database with the Composer program. The domain definition syntax is:

```
# comment
domain domain [description "text"]
    manager dmcpu
    [parent parentdomain]
end
[domain ...]
[cpuclass ...]
[cpu ...]
```

<i>comment</i>	Free-form text. Any text from the pound sign to end-of-line is treated as a comment.
<i>domain</i>	The name of the domain— up to eight characters starting with a letter. It can contain imbedded dashes and underscores.
<i>text</i>	Free-form description of the domain— up to 40 characters.
<i>dmcpu</i>	The Maestro cpu name of the domain manager. This must be an FTA running in full status mode.
<i>parentdomain</i>	The name of the parent domain. If omitted, the default is the master domain.

Note: The names used for cpus, cpu classes, and domains must be unique. That is, a domain cannot have the same name as a cpu or a cpu class. The master domain does not require a domain definition. The global options **master** and **master domain** provide the information. See [Global Options](#) on page 2-1.

Examples

A domain named `east`, whose parent is the master domain, 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
```

Jobs

A job is an executable file, program, or command that is scheduled and launched by Maestro. Job definitions can be included in files that are added to Maestro's database with the Composer program. Note that job dependencies (for example, "job2 follows job1") are specified when you write your schedules, at which time, you can also add and modify job definitions. See [Job Statement](#) on page 8-65 for more information about defining jobs in schedules. The job definition syntax is:

```
$jobs
[cpu#]job
  {scriptname|jobfilename}file|docommand "cmd"
  streamlogon user
  [interactive]
  [description "desc"]
  [recv-opt]
[[cpu#]job...]

where recv-opt is:
  recovery {stop      } [after rcpu#rjob] [abendprompt "rtext"]
           {continue }
           {rerun   }
```

cpu The cpu or cpu class on which the job runs. If omitted, the local cpu is used. The pound sign (#) is a required delimiter. If a cpu class is used, it must match the cpu class of the job's schedule.

job The Maestro job name (up to 40 alphanumeric, dash (-), and underscore (_) characters, starting with a letter).

scriptname|jobfilename

Use **scriptname** for UNIX and Windows NT jobs, and use **jobfilename** for MPE jobs. For an executable file, enter the path name and any options and arguments (up to 4095 characters). Maestro parameters are permitted— see [Using Parameters in Job Definitions](#) on page 8-41.

For Windows NT jobs, include the file extension (for example: .cmd, .exe, .bat). UNC names are permitted. Do not use mapped drives.

If spaces or special characters, other than slashes (/) and backslashes (\), are included, the entire string must be enclosed in quotes (").

	If the path name contains spaces, enter the name in another file and use that file's path name here.
<code>docommand</code>	Use this keyword for a command, and enter a valid command and any options and arguments (up to 4095 characters) enclosed in quotes ("). A command is executed directly and, unlike <code>scriptname</code> , the configuration script (<code>jobmanrc</code>) is not executed. Otherwise, the command is treated as a job, and all job rules apply. Maestro parameters are permitted— see Using Parameters in Job Definitions on page 8-41.
<code>streamlogon</code>	<p>The user name under which the job will run. It can contain up to eight characters. If the name contains special characters it must be enclosed in quotes ("). This user must be able to log in on the computer on which the job will run. Maestro parameters are permitted— see Using Parameters in Job Definitions on page 8-41.</p> <p>For Windows NT jobs, the user must have a Maestro user definition. See User Definitions on page 8-43 for user requirements.</p>
<code>interactive</code>	For Windows NT jobs, include this keyword to indicate that the job runs interactively on the Windows NT desktop.
<code>description</code>	A free-form description of the job, enclosed in quotes.
<code>recovery</code>	Recovery option keywords and their parameters. If omitted, the default is <code>stop</code> with no recovery job and no recovery prompt.
<code>stop</code>	If the job abends, do not continue with the next job.
<code>continue</code>	If the job abends, continue with the next job.
<code>rerun</code>	If the job abends, rerun the job.
<code>after</code>	<p>The name of a recovery job to run if the parent job abends. Recovery jobs are run only once for each abended instance of the parent job. The keyword <code>recoveryjob</code> can be used in place of <code>after</code>.</p> <p>You can specify the recovery job's <code>cpu</code> if it is different than the parent job's <code>cpu</code>. If no <code>cpu</code> is specified, the parent job's <code>cpu</code> is assigned. Not all jobs are eligible to have recovery jobs run on a different <code>cpu</code>. Follow these guidelines:</p>

-
- The parent job's and recovery job's cpu must have Maestro 4.5.5 or later installed.
 - If either cpu is an extended agent, it must be hosted by a domain manager or a fault-tolerant agent running in Full Status mode.
 - The recovery job's cpu must be in the same domain as the original job's cpu.
 - If the recovery job's cpu is a fault-tolerant agent, it must be running in Full Status mode.

abendprompt

The text of a recovery prompt, enclosed in quotes, to be issued if the job abends. It can contain up to 64 characters. If the text begins with a colon (:), the prompt will be issued, but no reply is necessary to continue processing. If the text begins with an exclamation mark (!), the prompt will not be issued, but a reply is necessary to proceed. The keyword `recoveryprompt` can be used in place of `abendprompt`.

Using Parameters in Job Definitions

Parameters have the following uses and limitations in job definitions:

- Parameters are allowed in the Logon, Script File, and Command fields.
- A parameter can be used as the entire string or a part of it.
- Multiple parameters are permitted in a single field.
- Enclose parameters in carets (^).

For more information see [Parameters](#) on page 8-46.

Example

A file containing two job definitions:

```
$jobs
cpul#gll
    scriptname "/usr/acct/scripts/gll"
    streamlogon "acct"
    description "general ledger job1"
bkup
    scriptname "/usr/mis/scripts/bkup"
    streamlogon "mis"
    recovery continue after recjob1
```

User Definitions

User definitions are required for Windows NT users only. These users are used in the **streamlogon** field of job definitions. They are entered with the **composer** program, and must adhere to the syntax shown below.

```
# comment
username [cpu#]username
    password "password"
end
[username...]
```

<i>username</i>	The name of the Windows NT user.
<i>cpu</i>	The Maestro cpu on which the user is allowed to launch jobs. The pound sign is a required delimiter. If no <i>cpu</i> is specified the cpu on which composer is running is used.
<i>username</i>	The user name. It can include a domain name. Note that Windows NT user names are case-sensitive. The user must be able to log on to the computer on which it will have jobs launched by Maestro, and must have the right to "Log on as batch." If the name is not unique in Windows NT, it is considered to be a local user, a domain user, or a trusted domain user, in that order.
<i>password</i>	The user's password. This must be enclosed in quotes. To indicate no password for the user, use two consecutive quotes with no space (""). Once a user definition is compiled, there is no way to read the password. Users with appropriate security privileges can modify or delete a user, but password information is never displayed.

Trusted Domain User

If it will be necessary for Maestro to launch jobs for a trusted domain user, special attention must be given to defining user accounts. Assuming Maestro is installed in Domain1 for user account **maestro**, and user account **sue** in Domain2 needs to have jobs launched by Maestro, the following must be true:

- There must be mutual trust between Domain1 and Domain2.
- In Domain1 on the computers where jobs will be 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."

Example

The following is an example file with four user definitions:

```
username joe
    password "okidoki"
end
#
username server#jane
    password "okitay"
end
#
username dom1\jane
    password "righto"
end
#
username jack
    password ""
end
```

Calendars

Calendar definitions are entered with the `composer modify` command. When you enter the `modify` command, Composer copies the complete list of calendar definitions into an edit file and starts an editor to permit you to modify the list. The definitions must adhere to the following syntax:

```
$calendar  
calname ["desc"] dates  
[calname ...]
```

- calname* The name of the calendar (up to eight alphanumeric, dash (-), and underscore (_) characters, starting with a letter).
- desc* A free-form description of the calendar (up to 64 characters).
- dates* The dates to be included on the calendar. Multiple dates must be separated by at least one space. The format for dates is *mm/dd/yy*.

Examples

A file containing three calendars:

```
$calendar  
monthend "Month end dates 1st half '99"  
    01/31/99 02/28/99 03/31/99 04/30/99 05/31/99 06/30/99  
paydays  
    01/15/99 02/15/99  
    03/15/99 04/15/99  
    05/14/99 06/15/99  
holidays  
    01/01/99 02/15/99 05/31/99
```

Parameters

Parameter definitions are entered with the `composer modify` command. When you enter the `modify` command, Composer copies the complete list of parameter definitions into an edit file and starts an editor, permitting you to modify the list. The definitions must adhere to the following syntax:

```
$parm
  parmname "value"
  [parmname ...]
```

parmname The name of the parameter (up to eight alphanumeric, dash (-), and underscore (_) characters, starting with a letter).

value The value of the parameter (up to 72 characters). Do not include other parameters.

Examples

A file containing three parameters:

```
$parm
  arpath    "/usr/lib/arfiles"
  pdnum    "25"
  print    "/dev/lp03"
```

Using Parameters in a CLI Schedule or Job Definition

Maestro parameters, among other things, are intended for use as strings in `streamlogon`, `docommand`, and `scriptname` entries in job definitions and for Opens files and Prompt dependencies in a job or schedule where variables are useful. Parameters must be enclosed in a pair of carets (^) in the keyword field. When a parameter is used in a keyword field, the field must be enclosed in quotes ("). Multiple parameters can be used in a single field. The backslash (\) character may be used to escape a character that follows it, thus a "\" can be used to escape a "^".

The following example illustrates the use of two parameters in a job definition. The use of parameters in a schedule for Opens and Prompt dependencies is identical to the use of them in a job, as illustrated below.

Two parameters, **glpath** and **gllogon**, are defined as:

```
$parm
  glpath      "/glfiles/daily"
  gllogon     "gluser"
```

The **glpath** and **gllogon** parameters are then used in a job as follows:

```
schedule site1#glsched on monthend
:
gljob2
  scriptname  "/usr/gl^glpath^"
  streamlogon "^gllogon^"
  opens       "^glpath^/datafile"
  prompt      " :^glpath^ started by ^gllogon^"
end
```

Parameters are accessible to all Maestro jobs, schedules, and prompts. Parameters used in scheduling are expanded during Maestro pre-production processing. If the value of a parameter is altered, the job definition will automatically take on the new value when pre-production processing occurs.

Prompts

Prompt definitions are entered with the `composer modify` command. When you enter the `modify` command, Composer copies the complete list of prompt definitions into an edit file and starts an editor, permitting you to modify the list. The definitions must adhere to the following syntax:

```
$prompt
promptname "[:|!]text"
[promptname ...]
```

promptname The name of the prompt (up to eight alphanumeric, dash (-), and underscore (_) characters, starting with a letter).

text The text of the prompt (up to 200 characters). If the text begins with a colon (:), the prompt will be issued, but no reply is necessary to continue processing. If the text begins with an exclamation mark (!), the prompt will not be issued, but a reply is necessary to proceed.

One or more Maestro parameters may be used as part or all of the text string for a global prompt. If a parameter is used, the parameter string must be enclosed in a pair of carets (^) and the entire text string must be enclosed in quotes (").

You can include backslash "n" (\n) within the text to cause a new line.

Examples

A file containing three prompts:

```
$prompt
prmt1 "ready for job4? (y/n)"
prmt2 " :job4 launched"
prmt3 "!continue?"
```

Resources

Resource definitions are entered with the `composer modify` command. When you enter the `modify` command, Composer copies the complete list of resource definitions into an edit file and starts an editor, permitting you to modify the list. The definitions must adhere to the following syntax:

```
$resource
cpu#resname number ["desc"]
[cpu#resname...]
```

<i>cpu</i>	The name of the cpu or cpu class to which the resource belongs. The pound sign (#) is a required delimiter.
<i>resname</i>	The name of the resource (up to eight alphanumeric, dash (-), and underscore (_) characters, starting with a letter).
<i>number</i>	The number of available resource units in the range 0-1024.
<i>desc</i>	A free-form description of the resource (up to 40 characters).

Examples

A file containing four resources:

```
$resource
ux1#tapes          3 "tape units"
ux1#jobslots      24 "job slots"
ux2#tapes          2 "tape units"
ux2#jobslots      16 "job slots"
```

The Scheduling Language

This section describes the command line language used to write Maestro schedules. Command line schedules are written with Composer using the scheduling language syntax summarized below. The scheduling language keywords are explained later in this section.

```
schedule [cpu#]sched on date[,...] [except date[,...]]
    [at time[+n day[s]] ] [...]
    [carryforward ]
    [follows {[cpu#]sched[.job][,...]} ]
        {net::net_dep }
    [limit limit ]
    [needs [n] [cpu#]resource[,...]] ]
    [opens [cpu#]file[(qual)][,...]] ]
    [priority pri|hi|go ]
    [prompt {"[:|!]text" } ]
        {prompt }
    [until time[+n day[s]] ]
    [in order ]
:
job-statement
    [at time[+n day[s]] ] [...]
    [confirmed ]
    [every rate ]
    [follows {[cpu#]sched{.job|@} [,...]} ]
        {job }
        {net::net_dep }
    [needs [n] [cpu#]resource[,...]] ]
    [opens [cpu#]path[(qual)][,...]] ]
    [priority pri|hi|go ]
    [prompt {"[:|!]text" } ]
        {prompt }
    [until time[+n day[s]]]
[job-statement ...]
end
```

Scheduling Keywords

The scheduling language keywords are briefly described below.

Keyword	Description	Page
at	Define the time of day that schedule or job execution begins.	page 8-53
carryforward	Carry this schedule forward if it is not completed.	page 8-54
confirmed	The completion of this job requires confirmation.	page 8-56
end	Mark the end of a schedule.	page 8-57
every	Launch this job repeatedly at a specified rate.	page 8-58
except	Do not select this schedule on these dates.	page 8-59
follows	Do not launch this job or schedule until other jobs and schedules have completed successfully.	page 8-61
in order	Launch jobs in this schedule in the order they are listed.	page 8-63
<i>job-statement</i>	Define a job and its dependencies.	page 8-65
limit	Set a limit on the number of jobs that can be launched concurrently from this schedule.	page 8-67
needs	Define the number of units of a resource required by this job or schedule before it can be launched.	page 8-68
on	Define the dates on which this schedule is selected for execution.	page 8-70
opens	Define files that must be accessible before this job or schedule is launched.	page 8-72
priority	Define the Maestro run priority for a job or schedule.	page 8-74
prompt	Define prompts that must be replied to before this job or schedule is launched.	page 8-75
schedule	Assign a name to the schedule.	page 8-77
until	Define a time of day after which this job or schedule is not launched.	page 8-78

Dependencies

A dependency is a condition that must be satisfied before a job or schedule is launched. They are introduced in a schedule with the `follows`, `needs`, `opens` and `prompt` keywords. The maximum number of dependencies permitted for a job or schedule is 40. Note that dependencies are resolved in the order you specify them, except when you modify a schedule with the GUI. See [Order of Schedule Dependencies](#) on page 5-9 and [Order of Job Dependencies](#) on page 5-31 for more information.

Case Sensitivity

With the exception of path names, user names, and UNIX commands, which are case-sensitive, you can use either upper or lower case characters when writing your schedules.

Scheduling Keyword Descriptions

The following pages describe the syntax for scheduling language keywords. This is a reference section only; for complete descriptions of the keywords' functionality, refer to section 5, [Composing Schedules](#).

at

Define the time of day that a schedule or job will be launched.

```
at time [+n day[s]]
```

time The time of day in the range 0000 to 2359.

n An offset in days from the scheduled date and time.

Examples

The example assume Maestro's processing day starts at 6:00 am.

1. The following schedule, selected on Tuesdays, will be launched no sooner than 3:00 a.m. Wednesday morning.

```
schedule sked7 on tu at 0300:
```

2. The following schedule is launched after 10:00 a.m. Wednesday morning, and job1 is launched at that time. job2 is launched after 11:00 a.m. Wednesday morning, and job3 is launched after 3:00 a.m. Thursday morning.

```
schedule sked8 on tu at 1000 + 1 day :  
    job1  
    job2 at 1100  
    job3 at 0300  
end
```

carryforward

Mark a schedule for possible carry forward to the next processing day.

```
carryforwardv
```

Example

The following schedule is marked for carry forward if it has not completed at the time pre-production processing begins for a new day:

```
schedule sked43 on th
  carryforward
:
  job12
  job13
  job13a
end
```

Comments

Include comments in a schedule.

```
*[text] | <<text>>
```

Operation

Lines beginning with an asterisk (*) in column one are treated as a comment. Double less than (<<) and greater than (>>) signs can be used to enclose comments anywhere in a schedule.

Example

```
*****
* The weekly cleanup jobs
*****
*
schedule wkend on fr at 1830
    in order
:
    job1 <<final totals and reports>>
    job2 <<update database          >>
end
```

confirmed

Specifies that the job's completion must be confirmed with a Conman Confirm (see *Confirming a Job* on page 6-50 for general information or *confirm* on page 9-38 for command line information).

```
confirmed
```

Example

In the following schedule, successful confirmation for job1 must be received before job2 and job3 will be launched.

```
sched test1 on fr:  
    job1 confirmed  
    job2 follows job1  
    job3 follows job1  
end
```

end

Marks the end of a schedule. This must be the last keyword in a schedule.

```
end
```

Example

```
schedule sked1 on monthend
  in order
  :
  job1
  job2
  job3
end          <<end of schedule>>
```

every

Launch a job repeatedly at a specified rate.

```
every rate
```

rate The repetition rate expressed in hours and minutes, as: *hhmm*
(leading zeroes are not required). The rate can be greater than 24
hours.

Examples

1. Launch the job testjob every hour:

```
testjob every 100
```

2. Launch the job testjob1 every 15 minutes, between the hours of 6:00 p.m.
and 8:00 p.m.:

```
testjob1 at 1800 every 15 until 2000
```

except

Specify exceptions to a schedule's **on** dates.

```
except {date|day|calendar} [,...]
```

date A date. The format for dates is *mm/dd/yy*.

day A day of the week. The following keywords are allowed:

mo	Monday	fr	Friday
tu	Tuesday	sa	Saturday
we	Wednesday	su	Sunday
th	Thursday	weekdays	Every day except Saturday and Sunday

calendar The dates listed on this calendar. This can be followed by an offset expression in the form:

```
{+|-}n {day[s]|weekday[s]|workday[s]}
```

day[s] Includes every day of the week.

weekday[s] Includes every day except Saturday and Sunday.

workday[s] Same as **weekday[s]**, but excludes all dates that appear on a calendar named **holidays**, if it exists.

Examples

1. Select schedule `testskd2` to run every weekday except those days whose dates appear on calendars named `monthend` and `holidays`:

```
schedule testskd2 on weekdays
  except monthend,holidays :
```

2. Select schedule `testskd3` to run every weekday except 5/15/93 and 5/23/93:

```
schedule testskd3 on weekdays
  except 05/15/93,05/23/93 :
```

3. Select schedule 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 :
```


follows

Specify jobs and schedules that must complete successfully before this job or schedule is launched.

For schedules:

```
follows { [cpu#]sched[.job|@] } [,...]  
        { net::net_dep          }
```

For jobs:

```
follows { [cpu#]sched{.job|@} } [,...]  
        { job                    }  
        { net::net_dep          }
```

cpu The cpu on which the job or schedule runs. If omitted, the same cpu as the dependent job or schedule is used. The pound sign (#) is a required delimiter.

sched The name of the schedule. If omitted for a job, the same schedule as the dependent job is used.

job The name of the job. An at sign (@) can be used to indicate all jobs in the named schedule.

net The name of the Maestro network agent where the internetwork dependency is defined. The two colons (::) are a required delimiter.

net_dep The internetwork dependency in the form:

```
[cpu#]sched[.job]
```

If no cpu is specified, the default is the Maestro cpu to which the network agent is connected. This is defined by the Node and TCP Address fields of the network agent's cpu definition.

Examples

- Do not launch schedule skedc until schedule sked4 on cpu site1, and job joba in schedule sked5 on cpu site2 have completed successfully:

```
schedule skedc on fr  
follows site1#sked4,site2#sked5.job
```

2. Do not launch sked6 until jobx in the schedule skedx on remote network cluster4 has completed successfully:

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

3. Do not launch jobd until joba in the same schedule, and job3 in schedule skeda have completed successfully:

```
jobd follows joba,skeda.job3
```

4. Do not launch jobe until all jobs in schedule skedb on cpu unix1 have completed successfully:

```
jobe follows unix1#skedb.@"
```

in order

Specify that jobs are to be launched in the order they appear in a schedule. Each job must complete successfully before the next job is launched. `in order` must be the last keyword to appear at the schedule level, prior to the colon. No dependencies are permitted at the job level.

```
in order
```

Usage

If an `in order` job abends, and cannot be re-run successfully, the jobs that follow it must be released with a `conman release` command. For example, to release the jobs in schedule `sked1`, you would enter the Conman command:

```

rj sked1.@.
```

If an `in order` job is cancelled, the next job must be released with a Conman Release action. The remaining jobs in the schedule will continue as normal. For example, if job `j1` is cancelled, you can release job `j2` by entering the Conman command:

```
rj sked1.j2
```

`In order` schedules launched on fault-tolerant agent cpus are affected by `full status` mode if one or more of the jobs runs on a different cpu. `Full status` mode determines whether or not a fault-tolerant agent is informed about jobs running on other cpus. The fault-tolerant agent on which an `in order` job runs must have `full status` mode selected (`on`) in its cpu definition if it follows a job that runs on a different cpu. See [Examples](#) below. For more information about `full status` mode refer to [CPU Definition](#) on page 4-9.

Examples

1. Launch `joba`, `jobb` and `jobc`, one at a time, in the order they appear in `sked2`:

```

schedule sked2 on everyday in order :
    joba
    jobb
    jobc
end
```

2. Launch job1, site2#job2 and job3, one at a time, in the order they appear in sked3:

```
schedule sked3 on tu,th in order :  
    job1  
    site2#job2  
    job3  
end
```

If the schedule is launched on a fault-tolerant agent cpu (site3, for example), job site2#job2 will not be launched when job1 completes unless Full Status mode was selected in the cpu definition for site2. In addition, job3 will not be launched when site2#job2 completes unless Full Status mode was selected in the cpu definition for site3.

Job Statement

Job statements are used to place jobs in a schedule and define their dependencies. You can also include automatic documentation and recovery options. This causes a job definition to be added or modified in Maestro's database at the same time the schedule is added or modified.

```
[cpu#]job [auto-doc [rec-opt]] [job-dependency] [...]  
where auto-doc is:  
    {scriptname|jobfilename}file|docommand "cmd"  
    streamlogon user  
    [interactive]  
    [description "desc"]  
where recv-opt is:  
    recovery {stop      } [after rcpu#rjob] [abendprompt "rtext"]  
            {continue }  
            {rerun    }
```

[cpu#] job

The Maestro job name and, optionally, the cpu and cpu class on which it runs. If the cpu and cpu class are omitted, the job runs on the same cpu and cpu class as the schedule. Note that the cpu class for a job must match the cpu class for the schedule.

auto-doc and recv-opt

Automatic documentation and recovery options. See [Jobs](#) on page 8-39 for descriptions.

job-dependency

Scheduling keywords and parameters. See [The Scheduling Language](#) on page 8-50 for eligible job dependency keywords.

Documenting Jobs in Schedules with auto-doc

A job needs to be defined only once in Maestro's database, and can be used in multiple schedules. When a schedule is added or modified in the database, any jobs within it that contain automatic documentation or recovery options are also added or modified.

When documenting jobs, keep the following in mind:

- Jobs can be defined independently, or as objects in schedules. In either case, the changes take place in Maestro's databases, and do not affect

production until the start of the next processing day.

- If you add or replace a schedule that results in updating an already documented job, the new job information will affect all other schedules that use the job. Note that the Cross Reference Report can be used to determine the names of schedules in which a particular job appears (see [xref](#) on page 7-10).

Examples

1. A schedule with three previously defined jobs:

```
schedule bkup on fr at 20:00 :
  cpu1#jbk1
  cpu2#jbk2
    needs 1 tape
  cpu3#jbk3
    follows jbk1
end
```

2. A schedule with two auto-doc jobs:

```
schedule sked4 on mo :
  job1      scriptname "/usr/lib/maestro/scripts/jcljob1"
            streamlogon jack
            recovery stop abendprompt "continue production"
  sitel#job2 scriptname "/usr/lib/maestro/scripts/jcljob2"
            streamlogon jack
            follows job1
end
```

limit

Set a limit on the number of jobs that can be executing concurrently in a schedule.

```
limit limit
```

limit The number of concurrent jobs allowed (0-99). A limit of zero prevents any jobs from being launched.

Example

Limit to five the number of jobs that can be executing concurrently in schedule sked2:

```
schedule sked2 on fr  
  limit 5 :
```

needs

Specify resources that must be available before a job or schedule can be launched.

```
needs [n] [cpu#]resource [,...]
```

- n* The number of resource units required (up to 32). If omitted, one is the default.
- cpu* The name of the cpu on which the resource is defined. If omitted, the cpu of the dependent job or schedule is assumed. The pound sign (#) is a required delimiter. Resources can be used as dependencies only by jobs and schedules that run on the same cpu as the resource. However, a standard agent and its host can reference the same resources.
- resource* The name of the resource.

Note: Adding a resource dependency to a job that runs on a fault-tolerant agent (FTA), or to a schedule that contains a job that runs on an FTA, may prevent the job from executing if the resource is not defined on the FTA. To prevent this from happening, make certain that the cpu definition for the FTA has Full Status on, or, alternatively, make certain that the resource is defined on the FTA on which the job runs.

Examples

1. Prevent schedule 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 :
```


-
2. Allow no more than two jobs to execute concurrently in schedule sked4 (assumes that the jlimit resource has been defined with two available units):

```
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 required keyword that specifies when, or how often, a schedule is selected for execution by Maestro. **on** must follow the **schedule** keyword.

```
on {date|day|calendar|request} [, ...]
```

date A date. The format for dates is *mm/dd/yy*.

day A day of the week. The following keywords are allowed:

mo	Monday	th	Thursday	su	Sunday
tu	Tuesday	fr	Friday	weekdays	Every day except sa and su
we	Wednesday	sa	Saturday	everyday	Every day of the week

calendar The dates listed on this calendar. This can be followed by an offset expression in the form:

```
{+|-}n {day[s]|weekday[s]|workday[s]}
```

day[s] Includes every day of the week.

weekday[s] Includes every day except Saturday and Sunday.

workday[s] Same as **weekday[s]**, but excludes all dates that appear on a calendar named **holidays**, if it exists.

request On request only—do not select the schedule automatically.

Examples

1. Select schedule sked1 on Mondays and Wednesdays:

```
schedule sked1 on mo,we
```

2. Select schedule sked3 on 6/15/97, and on the dates listed on the updates calendar:

```
schedule sked3 on 6/15/97,updates
```

3. Select schedule sked4 two weekdays before each date appearing on the monthend calendar:

```
schedule sked4 on monthend -2 weekdays
```

4. Select schedule testskd1 every weekday except Wednesdays:

```
schedule testskd1 on weekdays
except we
```

-
5. Select schedule testskd3 every weekday except 5/16/97 and 5/24/97:

```
schedule testskd3 on weekdays  
  except 05/16/97,05/24/97
```
 6. Select schedule testskd4 every day except two weekdays prior to any date appearing on a calendar named monthend:

```
schedule testskd4 on everyday  
  except monthend -2 weekdays
```

opens

Specify files that must be available before a job or schedule can be launched.

```
opens [cpu#]"path"[(qualifier)] [,...]
```

<i>cpu</i>	The name of the cpu or cpu class on which the file exists. If omitted, the cpu or cpu class of the dependent job or schedule is assumed. The pound sign (#) is a required delimiter. If a cpu class is used, it must be the cpu class of the current schedule.
<i>path</i>	The path name of the file enclosed in quotes ("). One or more Maestro parameters can be used as part or all of the path string. If a parameter is used, the parameter string must be enclosed in a pair of carets (^). S
<i>qualifier</i>	<p>A valid test condition. On UNIX systems, the qualifier is passed to a <code>test(1)</code> command executed as <code>root</code> by <code>bin\sh</code>. The expression "%p" inserts the path name of the file.</p> <p>On NT systems, the function is executed as the <code>maestro</code> user. Valid qualifiers are:</p> <ul style="list-style-type: none"> -d %p True if the path name is a directory. -e %p True if the file exists. -f %p True if the file is an ordinary file. -r %p True if the file is readable. -s %p True if the file's size is not zero. -w %p True if the file is writeable. <p>The expression "%p" inserts the path name of the file.</p> <p>Entering "(notempty)" is the same as entering "(-s %p)". If no qualifier is specified, the default is "(-f %p)".</p>

Note: The combination of *path* and *qualifier* cannot exceed 148 characters, within which the basename of the *path* cannot exceed 28 characters.

Examples

1. Check to see that file `/usr/fred/datafiles/file88` on cpu `ux5` is available for read access before launching `ux2#sked6`:

```
schedule ux2#sked6 on tu
  opens ux5#"/usr/fred/datafiles/file88" :
```

2. Check to see if three home directories- `/john`, `/mary`, and `/roger`- exist, before launching job `jobr2`:

```
jobr2 opens "/users"(-d %p/john -a -d %p/mary -a -d %p/roger)
```

3. Check to see if `cron` has created its `FIFO` file before launching job `job6`:

```
job6 opens "/usr/lib/cron/FIFO"(-p %p)
```

4. Check to see that file `/users/john/execit1` on cpu `dev3` exists and is executable, before running job `jobt2`:

```
jobt2 opens dev3#"/users/john/execit1"(-x %p)
```

5. Check to see that Windows NT file `C:\usr\tech\checker\data-file` on cpu `nyc` exists with a size greater than zero, and is writable, before running job `job77`:

```
job77 opens nyc#"C:\usr\tech\checker\data-file"(-s %p -a -w %p)
```

priority

Specify the Maestro priority of a job or schedule.

```
priority pri|hi|go
```

pri The priority in the range 0-99. A priority of zero prevents the job or schedule from being launched.

hi The equivalent of priority 100.

go The equivalent of priority 101.

Example

The following example illustrates the relationship between schedule and job priorities. The jobs are launched in the order: job1, job2, joba, jobb.

```
schedule sked1 on tu          schedule sked2 on tu
  priority 50                  priority 10
:                               :
  job1 priority 15             joba priority 60
  job2 priority 10             jobb priority 50
end                             end
```

If the schedule priorities were the same, the jobs would be launched in the order: joba, jobb, job1, job2.

prompt

Include a prompt that must be answered affirmatively before a job or schedule will be launched.

```
prompt { "[:|!]text" [...] } [,...]
       {prompt}
```

prompt The name of a global prompt.

text A literal prompt in the form of a text string (up to 200 characters) enclosed in quotes ("). Multiple quoted strings 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 not displayed but requires a reply. You can include backslash "n" (\n) within the text to cause a new line.

One or more Maestro parameters can be used as part or all of the text string for a prompt. To use a parameter, place its parameter string between carets (^).

Examples

1. This example illustrates both literal and global prompts. The literal prompt, the first one, uses a parameter named sys1. When a single affirmative reply is received for the global prompt 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
```

2. A literal prompt with text that exceeds one line, or that you wish to have appear on more than one line, can be defined with multiple quoted strings:

```
schedule sked5 on fr
  prompt"The jobs in this schedule consume "
    "an enormous amount of cpu time. "
    "Do you want to launch it now? (y/n)"
:
  j1
  j2 follows j1
end
```

schedule

Specify the schedule name. With the exception of comments, this must be the first keyword in a schedule, and must be followed by the `on` keyword.

```
schedule [cpu#]sched on ...
```

- cpu* The name of the cpu on which the schedule will be launched. The pound sign (#) is a required delimiter. If omitted, the name of your login cpu is used.
- sched* The name of the schedule (up to eight alphanumeric, dash (-), and underscore (_) characters, starting with a letter).
- on* Specifies when, or how often, the schedule is selected for execution. See [on](#) on page 8-70 for more information.

Examples

```
schedule sked1 on tu
schedule sked-2 on everyday except fr
schedule hpux3#skedux7 on monthend
```

until

Specify a time of day after which a job or schedule will not be launched, regardless of other dependencies.

```
until time [+n day[s]]
```

- time* The time of day in the range 0000 through 2359.
- n* An offset in days from the production schedule date.

Examples

1. Do not launch schedule sked1 after 5:00 p.m. on Tuesdays:

```
schedule sked1 on tu until 1700 :
```
2. Launch job1 between 1:00 p.m. and 5:00 p.m. on weekdays:

```
schedule sked2 on weekdays :  
  job1 at 1300 until 1700  
end
```
3. Launch 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
```
4. Launch schedule sked4 on Sundays between 8:00 a.m. and 1:00 p.m. The jobs will also be launched within this window:

```
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
```

9

Conman Command Line Reference

Maestro's production environment is managed with the Conman program. Conman is used to start and stop production processing, alter and display the Production Control file, and control cpu linking in a network. This reference section describes the command line version of Conman in three parts:

- running Conman and program basics,
- selecting and qualifying jobs and schedules, and
- syntax and usage for Conman commands.

For a complete description of Conman refer to section 6, *Managing Production*.

Running Conman

Conman is run as follows:

```
conman ["command[&...][&"]]
```

Examples

1. Conman enters conversational mode and prompts for a command:

```
conman
```

2. Conman executes the `sj` and `sp` commands, and quits:

```
conman "sj&sp"
```

3. Conman executes the `sj` and `sp` commands, and then enters conversational mode and prompts for a command.:

```
conman "sj&sp&"
```

4. Conman reads commands from *cfile*:

```
conman < cfile
```

5. Commands from *cfile* are piped to Conman:

```
cat cfile|conman
```

Control Characters

The following control characters can be entered in conversational mode to interrupt Conman (if your `sety` settings are configured as such).

Control c Conman stops executing the current command at the next interruptible step, and returns a command prompt.

Control d Conman quits after executing the current command.

Executing System Commands

When a system command is entered—using a pipe or a system command prefix (: or !)—it is executed by a child process. The child's effective user id is set to the id of the user running Conman to prevent security breaches.

User Prompting

When wildcards are used to select the objects to be acted upon by a command, Conman will prompt for "yes" or "no" confirmation after finding each matching object. Responding "yes" will cause the action to be taken on the object, and "no" will skip the object without taking action.

When Conman is run interactively, the confirmation prompts are issued at your terminal. 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 you run Conman non-interactively, 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 running non-interactively.

Terminal Output

Output to your terminal is controlled by shell variables, **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	The number of lines per screen. If not set, the default is 24. At the end of each screen page, Conman prompts to continue. If MAESTROLINES (or LINES) is set to zero or less, Conman does not pause at the end of a page.
\$MAESTROCOLUMNS	The number of characters per line. If not set, the default is 80.
\$MAESTRO_OUTPUT_STYLE	The method of displaying object names. If set to LONG , names are displayed in full. If not set, or set to any value other than LONG , and the global option expanded version is set to yes , long names are truncated to eight characters followed by a plus (+) sign; if expanded version is set to no , long names are truncated to eight characters.

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:

\$MAESTROLFP	The destination of a command's output. You can set it to any of the following:
> file	Redirect output to a file, overwriting its contents. If the file does not exist it is created.
>> file	Redirect output to a file, appending to the end of file. If the file does not exist it is created.
 command	Pipe output to a system command. The system command is always executed.
 command	Pipe output to a system command. The system command is not executed if there is no output.

	If not set, the default is " <code> lp -tCONLIST</code> ".
<code>\$MAESTROLPLINES</code>	The number of lines per page. If not set, the default is 60.
<code>\$MAESTROLPCOLUMNS</code>	The number of characters per line. If not set, the default is 132.

The variables must be exported before running Conman.

Conman's Command Prompt

The Conman command prompt is, by default, a percent sign (%). This is defined in the message catalog file `maestrohome/catalog/maestro.msg`. To select a different prompt, edit the file, locate set 202, message numbers 110 and 111, and change the characters on those lines. Message number 110 is the standard prompt, and message number 111 is the prompt used after you have selected a different `symphony` file with the `setsym` command. The prompts can be up to eight characters in length. After changing the prompts, generate a new catalog file as follows:

```
cd maestrohome/catalog
gencat maestro.cat maestro.msg
```

Command Syntax

Conman commands consist of the following elements:

command selection options

<i>command</i>	The command name.
<i>selection</i>	The object or set of objects to be acted upon. This generally consists only of the object names, but, in some cases, it may include additional qualifiers to further define the objects.
<i>options</i>	The actions to be performed, or command options to be selected.

For example:

```
sj sked1. @+state=hold~priority=0;info;offline
```

where:

sj	is the abbreviated form of the <code>showjobs</code> command.
sked1. @	selects all jobs in the schedule "sked1".
+state=hold	qualifies only those jobs in the "hold" state.
~priority=0	excludes those jobs with a priority of zero.
;info and ;offline	are command options.

Wildcards

The following wildcard characters are permitted in *selection* values:

- @ Replaces one or more alphanumeric characters.
- ? Replaces one alphabetic character.
- % Replaces one numeric character.

Delimiters and Special Characters

The following characters have special meanings in Conman commands:

&	Command delimiter. See <i>Running Conman</i> on page 9-1.
+	Qualifier delimiter used to add qualification criteria. For example: <code>sked1. @+priority=0</code>
~	Qualifier delimiter used to exclude qualification criteria. For example: <code>sked1. @~priority=0</code>
;	Option delimiter. For example: <code>;info;offline</code>
,	Repetition and range delimiter. For example: <code>state=hold,sked,pend</code>
=	Value delimiter. For example: <code>state=hold</code>
: !	Command prefixes which cause the command to be passed on to the system. These prefixes are optional—if Conman does not recognize the command, it is passed automatically to the system. For example: <code>!ls</code> or <code>:ls</code>
<<>>	Comment brackets. Comments can be enclosed in double less than (<<) and greater than (>>) signs on a single line anywhere in a schedule. For example: <code>sj @#@.@ <<comment>></code>
*	Comment prefix. The prefix must be the first character on a command line or following a command delimiter. For example: <code>*comment</code> or <code>sj&*comment</code>
>	Redirect command output to a file, overwriting the contents of the file. If the file does not exist it is created. For example: <code>sj> joblist</code>
>>	Redirect command output to a file, appending to the end of file. If the file does not exist it is created. For example: <code>sj >> joblist</code>
	Pipe command output to a system command or process. The system command is always executed. For example: <code>sj grep ABEND</code>
	Pipe command output to a system command or process. The system command is not executed if there is no output. For example: <code>sj grep ABEND</code>

List of Commands

The following table summarizes Conman's command set. Command names and keywords can be entered in either uppercase or lowercase, and can be abbreviated to as few leading characters as needed to distinguish them from one another. Some of the command names also have short-forms.

Table notes:

1. Cpu types: M= domain managers, F=fault-tolerant agent, A=standard agent.
2. `display file (df)` only.
3. `limit cpu (lc)` only.
4. `submit job (sbj)` and `submit sched (sbs)` only if the `mozart` directory on the master domain manager is accessible to the standard agent cpu.
5. Not available on Windows NT.

Command	Short Form	Description	Cpu Type ¹	Page
<code>addep</code>	<code>adj ads</code>	Add job or schedule dependencies.	M,F	9-26
<code>altpass</code>		Alter a User object definition password.	M,F	9-30
<code>altpri</code>	<code>ap</code>	Alter job or schedule priorities.	M,F	9-31
<code>cancel</code>	<code>cj cs</code>	Cancel a job or a schedule.	M,F	9-33
<code>command</code>		Send a command to the system.	M,F,A	9-37
<code>confirm</code>		Confirm job completion.	M,F	9-38
<code>console</code>		Assign the Maestro console.	M,F,A	9-40
<code>continue</code>		Ignore the next error.	M,F,A	9-42
<code>deldep</code>	<code>ddj dds</code>	Delete job or schedule dependencies.	M,F	9-43
<code>display</code>	<code>df dj ds</code>	Display files, jobs, and schedules.	M,F,A ²	9-47
<code>exit</code>		Terminate Conman.	M,F,A	9-50
<code>fence</code>		Set Maestro's job fence.	M,F,A	9-51
<code>help</code> ⁵		Display command information.	M,F,A	9-52
<code>kill</code>		Stop an executing job.	M,F	9-53
<code>limit</code>	<code>lc ls</code>	Change a cpu or schedule job limit.	M,F,A ³	9-54
<code>link</code>	<code>lk</code>	Open cpu links.	M,F,A	9-56

Command	Short Form	Description	Cpu Type ¹	Page
listsym		Display a list of Symphony log files.	M,F	9-58
recall	rc	Display prompt messages.	M,F	9-59
redo		Edit the previous command.	M,F,A	9-61
release	rj rs	Release job or schedule dependencies.	M,F	9-63
reply		Reply to prompt message.	M,F	9-67
rerun	rr	Rerun a job.	M,F	9-68
resource		Change the number of resource units.	M,F	9-71
setsym		Select a Symphony log file.	M,F	9-72
showcpus	sc	Display cpu and link information.	M,F,A	9-73
showdomain		Display domain information.	M,F,A	9-76
showfiles	sf	Display information about files.	M,F	9-77
showjobs	sj	Display information about jobs.	M,F	9-80
showprompts	sp	Display information about prompts.	M,F	9-87
showresources	sr	Display information about resources.	M,F	9-90
showschedules	ss	Display information about schedules.	M,F	9-92
shutdown		Stop Maestro's production processes.	M,F,A	9-95
start		Start Maestro's production processes.	M,F,A	9-96
status		Display Maestro's production status.	M,F,A	9-98
stop		Stop Maestro's production processes.	M,F,A	9-99
submit	sbd sbf sbj sbs	Submit a command, file, job, or schedule.	M,F,A ⁴	9-101
switchmgr		Switch the domain manager.	M,F	9-114
tellop	to	Send a message to the console.	M,F,A	9-115
unlink		Close cpu links.	M,F,A	9-116
version	v	Display Conman's program banner.	M,F,A	9-118

See *Table notes*: on page 9-7.

Selecting and Qualifying Jobs

In commands that operate on jobs, the *jobselection* parameters identify one or more jobs by name or job number, along with optional qualifiers. The syntax for selecting and qualifying jobs is summarized below, and described on the following pages.

```
command jobselection options
jobselection is:
    [cpu#] {sched.job }[+|~jobqual][...]
           {jobnum   }
    {net::net_dep}
```

For *command* and *options*, see the individual command descriptions later in this section.

<i>cpu</i>	When used with <i>sched.job</i> , this is the name of the cpu on which the schedule runs. When used with <i>jobnum</i> , it is the cpu on which the job runs. Wildcards are permitted.
<i>sched</i>	The name of the schedule in which the job runs. Wildcards are permitted.
<i>job</i>	The name of the job. Wildcards are permitted.
<i>jobnum</i>	The job number.
<i>jobqual</i>	See Job Qualifiers below.

The internetwork dependency syntax can be used to select jobs on remote Maestro networks. The format is:

```
net::net_dep
```

<i>net</i>	The name of the Maestro Network agent that interfaces with the network containing the dependency. The two colons (::) are a required delimiter. Wildcards are permitted.
<i>net_dep</i>	The name of the job dependency in the following format: [<i>cpu#</i>] <i>sched.job</i>

If no `cpu` is specified, the default is the Maestro `cpu` to which Network agent is connected. This is determined by the Node and TCP Address fields of the Network agent's `cpu` definition. Wildcards are valid.

See appendix D, [Internetwork Dependencies](#) for more information.

Job Qualifiers

Job qualifiers specify attributes of jobs to be acted upon by a command. They are prefixed by `+` or `~`. A `+` means that jobs with the attribute qualify for the command. A `~` means that jobs with the attribute are excluded from the command.

Job qualifier keywords can be abbreviated to as few leading characters as needed to distinguish one from another.

at

Qualify jobs based on their scheduled `at` times.

```
at [=time           ]
   [=lowtime,       ]
   [=,hightime      ]
   [=lowtime,hightime]
```

<i>time</i>	The scheduled execution time, expressed as: <i>hhmm</i> [<i>+n days</i>] <i>date</i>
<i>hhmm</i>	The hour and minute.
<i>n</i>	The number of days from the next occurrence of <i>hhmm</i> .
<i>date</i>	The execution date. The default date format is <i>mm/dd/yy</i> .
<i>lowtime</i>	The lower limit of a time range, expressed the same as <i>time</i> . If used without <i>hightime</i> , jobs are selected that have <code>at</code> times on or after this time.
<i>hightime</i>	The upper limit of a time range, expressed the same as <i>time</i> . If used without <i>lowtime</i> , jobs are selected that have <code>at</code> times on or before this time.

If no times are specified (that is, "`at`"), the range is open-ended, and jobs are qualified or excluded if they were scheduled using the `at` keyword.

confirmed

Qualify jobs based on the fact that job completion must be confirmed by executing a `comman confirm` command. These are jobs that were scheduled using the `confirm` keyword.

```
confirmed
```

every

Qualify jobs based on their scheduled **every** rates.

```
every [=rate           ]
      [=lowrate,       ]
      [=,highrate     ]
      [=lowrate,highrate]
```

rate The scheduled execution rate, expressed as: *hhmm*
hhmm The hour and minute.

lowrate The lower limit of a rate range, expressed the same as *rate*. If used without *highrate*, jobs are selected that have **every** rates equal to or greater than this rate.

highrate The upper limit of a rate range, expressed the same as *rate*. If used without *lowrate*, jobs are selected that have **every** rates equal to or less than this rate.

If no rates are specified (that is, "every"), the range is open-ended, and jobs are qualified or excluded if they were scheduled using the **every** keyword.

finished

Qualify jobs based on whether or not they have finished executing.

```
finished [=time           ]
         [=lowtime,       ]
         [=,hightime     ]
         [=lowtime,hightime]
```

<i>time</i>	The exact time execution was completed, expressed as: <i>hhmm</i> [<i>date</i>] <i>hhmm</i> The hour and minute. <i>date</i> The completion date. The default date format is <i>mm/dd/yy</i> . There must be at least one space between <i>hhmm</i> and <i>date</i> .
<i>lowtime</i>	The lower limit of a time range, expressed the same as <i>time</i> . If used without <i>hightime</i> , jobs are selected that finished at or after this time.
<i>hightime</i>	The upper limit of a time range, expressed the same as <i>time</i> . If used without <i>lowtime</i> , jobs are selected that finished at or before this time.

If no times are specified (that is, "finished"), the range is open-ended, and jobs are qualified or excluded if they have finished executing.

follows

Qualify jobs based on whether or not they have a follows dependency.

```
follows [= [cpu#] sched.job ]
        [= job ]
        [= net::net_dep ]
```

<i>cpu</i>	The name of the cpu on which the prerequisite job runs. Wildcards are permitted.
<i>sched</i>	The name of the schedule in which the prerequisite job runs. Wildcards are permitted. If entered as <i>sched.@</i> , it means that the target job follows all jobs in this schedule.
<i>job</i>	The name of the prerequisite job. When entered without a <i>sched</i> , it means that the prerequisite job is in the same schedule as the target job. Wildcards are permitted.
<i>net::net_dep</i>	The name of the internetwork dependency. See follows on page 8-61 for information.

If no prerequisite jobs are specified, jobs are qualified or excluded if they were scheduled using the `follows` keyword.

logon

Qualify jobs based on the user names under which they run.

```
logon=user
```

user The user name under which the job runs. If the name contains special characters it must be enclosed in quotes ("). Wildcards are permitted.

needs

Qualify jobs based on whether or not they have a **needs** dependency.

```
needs [= [cpu#] resource]
```

cpu The name of the cpu on which the resource is defined. Wildcards are permitted.

resource The name of the resource. Wildcards are permitted.

If no resources are specified, jobs are qualified or excluded if they were scheduled using the **needs** keyword.

opens

Qualify jobs based on whether or not they have an **opens** dependency.

```
opens [= [cpu#] {path } [(qualifier)] ]
           {filename }
```

cpu The name of the cpu on which the file exists. Wildcards are permitted.

path The path name of the file. The name must be enclosed in quotes (") if it contains characters *other than* the following: alphanumerics, dashes (-), slashes (/), and underscores (_). Wildcards are permitted.

filename The filename (basename). This is not case-sensitive. Wildcards are permitted.

qualifier Valid **test** condition. For information see *opens* on page 8-72.
If omitted, jobs are qualified without regard to a qualifier.

If no files are specified, jobs are qualified or excluded if they were scheduled with an *opens* dependency.

priority

Qualify jobs based on their priorities.

```
priority= {pri           }
           {lowpri,      }
           {,highpri     }
           {lowpri,highpri}
```

pri The priority value, 0-99, hi or go.

lowpri The lower limit of a priority range. If used without *highpri*, jobs are selected with priorities equal to or greater than this value.

highpri The upper limit of a priority range. If used without *lowpri*, jobs are selected with priorities equal to or less than this value.

prompt

Qualify jobs based on whether or not they have a **prompt** dependency.

```
prompt [=prompt |msgnum]
```

prompt The name of a global prompt. Wildcards are permitted.

msgnum The message number of a local prompt.

If no prompts are specified, jobs are qualified or excluded if they were scheduled using the **prompt** keyword.

recovery

Qualify jobs based on their recovery options.

```
recovery=recv-option
```

recv-option The recovery option: **stop**, **continue**, or **rerun**.

scriptname

Qualify jobs based on their script file names.

```
scriptname=path
```

path The path name of script file. The name must be enclosed in quotes ("") if it contains characters other than the following: alphanumeric, dashes (-), slashes (/), and underscores (_). Wildcards are permitted.

started

Qualify jobs based on whether or not they have started executing.

```
started [=time                    ]
         [=lowtime,               ]
         [=,hightime             ]
         [=lowtime,hightime ]
```

time The time execution was started, expressed as: *hhmm* [*date*]

hhmm The hour and minute.

date The start date. The default date format is *mm/dd/yy*. The must be at least one space between *hhmm* and *date*.

lowtime The lower limit of a time range, expressed the same as time. If used without *hightime*, jobs are selected that started at or after this time.

hightime The upper limit of a time range, expressed the same as time. If used without *lowtime*, jobs are selected that started at or before this time.

If no times are specified (that is, "started"), the range is open-ended, and jobs are qualified or excluded if they have started executing.

state

Qualify jobs based on their states.

```
state=state[,...]
```

state The current state of the job. Valid job states are:

abend	Job terminated with a non-zero exit code.
abensp	"abend" confirmation received, but job not completed.
add	Job is being submitted.
done	Job completed in an unknown state.
error	For internetwork dependencies only, an error occurred while checking for the remote status.
exec	Job is executing.
extrn	For internetwork dependencies only, an unknown status. An error occurred, a rerun action was just performed on the EXTERNAL job, or the remote job or schedule does not exist.
fail	Unable to launch job.
fence	Job's priority is below the fence.
hold	Job awaiting dependency resolution.
intro	Job introduced for launching by the system.
pend	Job completed, awaiting confirmation.
ready	Job ready to launch, all dependencies resolved.
sched	Job's at time has not arrived.
succ	Job completed with zero exit code.
succp	"succ" confirmation received, but job not completed.
susp	(MPE only) Job suspended by breakjob command.
wait	(Extended agent and MPE only) Job is in the wait state.
waitd	(MPE only) Job is in the wait state, and is deferred.

until

Qualify jobs based on their scheduled `until` times.

```

until [=time           ]
        [=lowtime,      ]
        [=,hightime     ]
        [=lowtime,hightime ]

```

time The until time, expressed as: *hhmm*[{+*n* **days**} | *date*]

hhmm The hour and minute.

n The number of days from the next occurrence of *hhmm*.

date The `until` date. The default date format is *mm/dd/yy*.

lowtime The lower limit of a time range, expressed the same as *time*. If used without *hightime*, jobs are selected with `until` times equal to or after this time.

hightime The upper limit of a time range, expressed the same as *time*. If used without *lowtime*, jobs are selected with `until` times equal to or before this time.

If no times are specified (that is, "until"), the range is open-ended, and jobs are qualified or excluded if they were scheduled using the `until` keyword.

Selecting and Qualifying Schedules

In commands that operate on schedules, the *schedselection* parameters identify one or more schedules by name, along with optional qualifiers. The syntax for selecting and qualifying schedules is summarized below, and described on the following pages.

```
command schedselection options
schedselection is:
    [cpu#]sched [+|~schedqual][...]
```

For *command* and *options*, see the individual command descriptions later in this section.

- cpu* The name of the cpu on which the schedule runs. Wildcards are permitted.
- sched* The name of the schedule. Wildcards are permitted.
- schedqual* See Schedule Qualifiers below.

Schedule Qualifiers

Schedule qualifiers specify attributes of schedules to be acted upon by a command. They are prefixed by + or ~. A + means that schedules with the attribute qualify for the command. A ~ means that schedules with the attribute are excluded from the command.

Schedule qualifier keywords can be abbreviated to as few leading characters as needed to distinguish one from another.

at

Qualify schedules based on their scheduled at times.

```
at  [=time           ]
    [=lowtime,       ]
    [=,hightime      ]
    [=lowtime,hightime]
```

<i>time</i>	The scheduled execution time, expressed as: <i>hhmm</i> [{ + <i>n</i> days } <i>date</i>]
<i>hhmm</i>	The hour and minute.
<i>n</i>	The number of days from the next occurrence of <i>hhmm</i> .
<i>date</i>	The execution date. The default date format is <i>mm/dd/yy</i> .
<i>lowtime</i>	The lower limit of a time range, expressed the same as <i>time</i> . If used without <i>hightime</i> , schedules are selected that have at times on or after this time.
<i>hightime</i>	The upper limit of a time range, expressed the same as <i>time</i> . If used without <i>lowtime</i> , schedules are selected that have at times on or before this time.

If no times are specified (that is, "at"), the range is open-ended, and schedules are qualified or excluded if they contain the **at** keyword.

carriedforward

Qualify schedules based on whether or not they were carried forward.

```
carriedforward
```

carryforward

Qualify schedules based on whether or not they contain the **carryforward** keyword.

```
carryforward
```

finished

Qualify jobs based on whether or not they have finished executing.

```
finished [=time           ]
           [=lowtime,      ]
           [=,hightime     ]
           [=lowtime,hightime ]
```

time The exact time execution was completed, expressed as: *hhmm* [*date*]

hhmm The hour and minute.

date The completion date. The default date format is *mm/dd/yy*. There must be at least one space between *hhmm* and *date*.

lowtime The lower limit of a time range, expressed the same as *time*. If used without *hightime*, schedules are selected that finished at or after this time.

hightime The upper limit of a time range, expressed the same as *time*. If used without *lowtime*, schedules are selected that finished at or before this time.

If no times are specified (that is, "finished"), the range is open-ended, and schedules are qualified or excluded if they have finished executing.

follows

Qualify schedules based on whether or not they have a follows dependency.

```
follows [= [cpu#] sched [ . job ] ]
           [= net::net_dep           ]
```

cpu The name of the *cpu* on which the prerequisite schedule runs. Wildcards are permitted.

sched The name of the prerequisite schedule. Wildcards are permitted.

job The name of the prerequisite job. Wildcards are permitted.

net::net_dep The name of the internetwork dependency. See *follows* on page 8-61 for information.

If no prerequisite schedules are specified, schedules are qualified or excluded if they contain the *follows* keyword.

limit

Qualify schedules based on whether or not they contain the *limit* keyword.

```

limit [=time           ]
        [=lowlimit,     ]
        [=,highlimit   ]
        [=lowlimit,highlimit]

```

limit The job limit value.

lowlimit The lower limit of a range. If used without *highlimit*, schedules are selected that have job limits equal to or greater than this value.

highlimit The upper limit of a range. If used without *lowlimit*, schedules are selected that have job limits equal to or less than this value.

If no limits are specified (that is, "*limit*"), the range is open-ended, and schedules are qualified or excluded if they have a job limit.

needs

Qualify schedules based on whether or not they have a *needs* dependency.

```

needs [= [cpu#] resource]

```

cpu The name of the cpu on which the resource is defined. Wildcards are permitted.

resource The name of the resource. Wildcards are permitted.

If no resources are specified, schedules are qualified or excluded if they contain the *needs* keyword.

opens

Qualify schedules based on whether or not they have an **opens** dependency.

```
opens [= [cpu#] {path } [(qualifier)]]
           {filename }
```

<i>cpu</i>	The name of the cpu on which the file exists. Wildcards are permitted.
<i>path</i>	The path name of the file. The name must be enclosed in quotes (") if it contains characters <i>other than</i> the following: alphanumerics, dashes (-), slashes (/), and underscores (_). Wildcards are permitted.
<i>filename</i>	The case-insensitive basename of the file. Wildcards are permitted.
<i>qualifier</i>	Valid test condition. For information see opens on page 8-72. If omitted, schedules are qualified without regard to a qualifier.

If no files are specified, schedules are qualified or excluded if they contain an **opens** dependency.

priority

Qualify schedules based on their priorities.

```
priority= {pri           }
           {lowpri,      }
           {,highpri     }
           {lowpri,highpri}
```

<i>pri</i>	The priority value, 0-99, hi or go .
<i>lowpri</i>	The lower limit of a priority range. If used without <i>highpri</i> , schedules are selected with priorities equal to or greater than this value.
<i>highpri</i>	The upper limit of a priority range. If used without <i>lowpri</i> , schedules are selected with priorities equal to or less than this value.

prompt

Qualify schedules based on whether or not they have a **prompt** dependency.

```
prompt [=prompt | msgnum]
```

prompt The name of a global prompt. Wildcards are permitted.

msgnum The message number of a literal prompt.

If no prompts are specified, schedules are qualified or excluded if they contain the **prompt** keyword.

started

Qualify schedules based on whether or not they have started executing.

```
started [=time                    ]
          [=lowtime,               ]
          [=,hightime             ]
          [=lowtime,hightime ]
```

time The time execution was started, expressed as: *hhmm* [*date*]

hhmm The hour and minute.

date The start date. The default date format is *mm/dd/yy*. The must be at least one space between *hhmm* and *date*.

lowtime The lower limit of a time range, expressed the same as time. If used without *hightime*, schedules are selected that started at or after this time.

hightime The upper limit of a time range, expressed the same as time. If used without *lowtime*, schedules are selected that started at or before this time.

If no times are specified (that is, "started"), the range is open-ended, and schedules are qualified or excluded if they have started executing.

state

Qualify schedules based on their states.

```
state=state[,...]
```

<i>state</i>	The current state of the schedule. Valid schedule states are:
abend	Schedule terminated unsuccessfully.
add	Schedule has just been submitted.
exec	Schedule is executing.
hold	Awaiting dependency resolution.
ready	Dependencies resolved, ready to launch.
stuck	Execution interrupted. No jobs will be launched without operator intervention.
succ	Schedule completed successfully.

until

Qualify schedules based on their `until` times.

```
until [=time           ]  
      [=lowtime,       ]  
      [=,hightime     ]  
      [=lowtime,hightime]
```

<i>time</i>	The until time, expressed as: <i>hhmm</i> [{+ <i>n</i> days } <i>date</i>]
<i>hhmm</i>	The hour and minute.
<i>n</i>	The number of days from the next occurrence of <i>hhmm</i> .
<i>date</i>	The <code>until</code> date. The default date format is <i>mm/dd/yy</i> .
<i>lowtime</i>	The lower limit of a time range, expressed the same as <i>time</i> . If used without <i>hightime</i> , schedules are selected with <code>until</code> times equal to or after this time.
<i>hightime</i>	The upper limit of a time range, expressed the same as <i>time</i> . If used without <i>lowtime</i> , schedules are selected with <code>until</code> times equal to or before this time.

If no times are specified (that is, "until"), the range is open-ended, and schedules are qualified or excluded if they contain the `until` keyword.

Command Descriptions

The following pages contain descriptions of Conman commands.

addep job

Add job dependencies.

Security: You must have **addep** access to the job. To include **needs** and **prompt** dependencies, you must have **use** access to the resources and global prompts.

```
{addep [job=] }jobselection[;dependency[;...]][;noask]
{adj          }
```

jobselection See [Selecting and Qualifying Jobs on page 9-9](#).

dependency See [Job Dependencies](#) below.

noask Do not prompt for confirmation before taking action against each qualifying job.

Job Dependencies

Following is a syntax summary of job dependencies. Wildcards are not permitted. For a complete description of job dependencies refer to [The Scheduling Language](#) on page 8-50.

```
at=hhmm[+n days |mm/dd/yy]
confirmed
every=rate
follows={ [cpu#]sched{.job|@}|job|net::net_dep}[,...]
needs=[num] [cpu#]resource[,...]
opens=[cpu#]"path"[(qualifier)][,...]
priority[=pri|hi|go]
prompt={"[:|!]text"|promptname}[,...]
until=hhmm[+n days |mm/dd/yy]
```

Usage

If you omit the priority level in **priority**, the job reverts back to its original scheduled priority. If you omit the *cpu* in **follows**, **needs**, or **opens**, the default is the *cpu* of the job. To add prompt dependencies while running

conman on a fault tolerant agent, you must have access to the *maestrohome/mozart* directory on the master domain manager.

Examples

1. Add a resource dependency to job3 in schedule sked9:

```
adddep sked9.job3;needs=2 tapes
```

2. Add a file dependency, and an `until` time to job6 in schedule sked2:

```
adj sked2.job6;opens="/usr/lib/prdata/file5"(-s %p);until=2330
```

adddep sched

Add schedule dependencies.

Security: You must have `adddep` access to the schedule. To include `needs` and `prompt` dependencies, you must have `use` access to the resources and global prompts.

```
{adddep [sched=] }schedselection[;dependency[;...]][;noask]
{ads }
```

schedselection See [Selecting and Qualifying Schedules](#) on page 9-18.

dependency See [Schedule Dependencies](#) below.

noask Do not prompt for confirmation before taking action against each qualifying schedule.

Schedule Dependencies

Following is a syntax summary of schedule dependencies. Wildcards are not permitted. For a complete description of schedule dependencies refer to [The Scheduling Language](#) on page 8-50.

```
at=hhmm[+n days|mm/dd/yy]
carryforward
follows={ [cpu#]sched[.job|@]]|job|net::net_dep}[,...]
limit=joblimit
needs=[num] [cpu#]resource[,...]
opens=[cpu#]"path"[(qualifier)][,...]
priority[=pri|hi|go]
prompt={"[:|!]text"|promptname}[,...]
until=hhmm[+n days|mm/dd/yy]
```

Usage

If you omit the priority level in `priority`, the schedule reverts back to its original priority. If you omit the `cpu` in `follows`, `needs`, or `opens`, the default is the `cpu` of the schedule. To add prompt dependencies while running

conman on a fault tolerant agent, you must have access to the *maestrohome/mozart* directory on the master domain manager.

Examples

1. Add a **prompt** dependency to schedule sked3:
`adddep sched=sked3;prompt=msg103`
2. Add a **follows** dependency, and a job limit to schedule sked4:
`ads sked4;follows=sked3;limit=2`

altpass

Alter the password of a User object.

Security: You must have `altpass` access to the User.

```
altpass [cpu#]username["password"]
```

<i>cpu</i>	The Maestro cpu on which the User is defined. The default is this cpu.
<i>username</i>	The name of the User.
<i>password</i>	The new password for the User. This must be enclosed in quotes. To indicate no password for the user, use two consecutive quotes with no space between ("").

Usage

If the *password* field is omitted, 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 only in effect until the *Jnextday* job runs to turnover a new day. To make a permanent change see [User Definitions](#) on page 4-54, or [User Definitions](#) on page 8-43.

Examples

1. Change the password of User jim on cpu mis5 to "giraffe":

```
altpass mis5#jim;"giraffe"
```

2. Change the password of User jim on cpu mis5 to "giraffe" without showing it in clear text:

```
altpass mis5#jim
```

```
password: xxxxxxxx <enter password and hit Return>
```

```
confirm: xxxxxxxx <enter password again and hit Return>
```


altpri

Alter job and schedule priorities.

Security: You must have `altpri` access to the job or schedule.

```
{altpri }{jobselection }[;pri][;noask]
{ap }{schedselection }
```

`jobselection` See [Selecting and Qualifying Jobs on page 9-9](#).

`schedselection` See [Selecting and Qualifying Schedules on page 9-18](#).

`pri` The priority level: 0 through 99, `hi`, or `go`.

`noask` Do not prompt for confirmation before taking action against each qualifying job or schedule.

Usage for Jobs

If you do not specify a priority level, the job reverts back to its original priority. `hi` and `go` jobs are launched as soon as their dependencies are satisfied. They override the cpu job limit, but they do not override the schedule job limit nor Maestro's jobfence.

Usage for Schedules

If you do not specify a priority level, the schedule reverts back to its original priority. A schedule's priority affects all of its jobs. In the case of `hi` and `go`, all jobs are given `hi` or `go` priority. `hi` and `go` jobs are launched as soon as their dependencies are satisfied. They override the cpu job limit, but they do not override the schedule job limit nor Maestro's jobfence.

Examples

1. Change the priority of the balance job in schedule glmonth:

```
altpri glmonth.balance;55
```

2. Change the priority of all jobs in schedule mis5:

```
ap mis5.@;25
```

3. Change the priority of schedule glmonth:

```
altpri glmonth;10
```

4. Change the priority of schedule mis5:

```
ap mis5;30
```

cancel job

Cancel a job.

Security: You must have `cancel` access to the job.

```
{cancel} jobselection[;pend][;noask]
{cj      }
```

`jobselection` See [Selecting and Qualifying Jobs on page 9-9](#).

`pend` Cancel the job only after its dependencies are resolved.

`noask` Do not prompt for confirmation before taking action against each qualifying job.

Usage

If a job is cancelled before it is launched, it will not launch. If cancelled after it is launched, the job continues to execute. If an executing job is cancelled and it completes in the ABEND state, no automatic job recovery steps are attempted. If the `;pend` option is not used, jobs and schedules that are dependent on the cancelled job are released immediately.

If you include the `;pend` option, and the job has not been launched, the cancel is deferred until all of the dependencies, including an `at` time, for the job are resolved. Once all the dependencies are resolved, the job is cancelled and any jobs or schedules that are dependent on the cancelled job are released. During the period the cancel is deferred, the notation "[Cancel Pend]" is listed in the Dependencies column of the job in a `show job` display.

If the `;pend` option is used for a job that has already been launched, the option is ignored, and any jobs or schedules that are dependent on the cancelled job are immediately released.

Jobs that have been cancelled, or that are "[Cancel Pend]", can be rerun with the `rerun` command. You can also add and delete dependencies (`addep` and `deldep` commands) on jobs that are marked "[Cancel Pend]".

To immediately cancel a job that is marked "[Cancel Pend]", you can either issue a `release` command for the job, or issue another `cancel` command without the `;pend` option.

For jobs whose `until` times have expired, the notation "[Until]" is listed in the Dependencies column of the job in a `show job` 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 (`release`

or `deldep` command), or issue another `cancel` command without the `;pend` option.

The evaluation of dependencies can be stopped by setting the priority of a schedule to zero with an `altpri` command. To resume dependency evaluation, raise the priority above zero.

Note: Regarding internetwork dependencies, a `cancel` action on an EXTERNAL job releases all local jobs and schedules from the dependency (EXTERNAL jobs represent jobs and schedules that have been specified as internetwork dependencies). The status of an internetwork dependency ceases to be checked after a `cancel` is performed. For more information see appendix D, [Internetwork Dependencies](#).

Examples

1. Cancel the report job in schedule apwkly on cpu site3:

```
cancel site3#apwkly.report
```
2. Cancel the setup job in schedule mis5 if it is not in theabend state:

```
cj mis5.setup~state=abend
```
3. Cancel the job3 job in schedule sked3 only after its dependencies are resolved:

```
cj sked3.job3;pend
```

cancel sched

Cancel a schedule.

Security: You must have `cancel` access to the schedule.

```
{cancel} schedselection[;pend][;noask]
{cs }
```

<i>schedselection</i>	See Selecting and Qualifying Schedules on page 9-18.
pend	Cancel the schedule only after its dependencies are resolved.
noask	Do not prompt for confirmation before taking action against each qualifying schedule.

Usage

If a schedule is cancelled before it is launched, it will not launch. If cancelled after it is launched, its executing jobs are allowed to complete, but no other jobs are launched. If the `;pend` option is not used, jobs and schedules that are dependent on the cancelled schedule are released immediately.

If the `;pend` option is used, and the schedule has not been launched, the cancel is deferred until all of its dependencies, including an `at` time, are resolved. Once all dependencies are resolved, the schedule is cancelled and any dependent jobs or schedules are released. During the period the cancel is deferred, the notation "[Cancel Pend]" is listed in the Dependencies column of the schedule in a `show schedule` display.

If the `;pend` option is used for a schedule that has already been launched, any remaining jobs in the schedule are cancelled, and any dependent jobs and schedules are released.

To immediately cancel a schedule that is marked "[Cancel Pend]", issue a `release` command for the schedule, or issue another `cancel` command without `;pend`.

For schedules whose `until` times have expired, the notation "[Until]" appears in the Dependencies column of the `show schedule` display, and their dependencies are no longer evaluated. If such a schedule is also marked "[Cancel Pend]", it is not cancelled until you release or delete the `until` time (`release` or `deldep` command), or issue another `cancel` command without `;pend`.

Evaluation of dependencies can be stopped by setting a schedule's priority to zero with an `altpri` command. To resume dependency evaluation, raise the priority above zero.

Examples

1. Cancel schedule `sked1` on `cpu site2`:

```
cancel site2#sked1
```

2. Cancel schedule `mis2` if it is in the stuck state:

```
cs mis2+state=stuck
```

3. Cancel schedule `sked3` only after its dependencies are resolved:

```
cs sked3;pend
```

Command

Execute a system command.

```
[ : | ! ] command
```

command Any valid system command. The prefix (: or !) is required only when the command is spelled the same as a Conman command.

Example

Execute a **ps** command:

```
ps -ef
```

confirm

Confirm the completion of a job that was scheduled with the `confirmed` keyword— an exception is noted under *Usage* below.

Security: You must have `confirm` access to the job.

```
confirm jobselection;{succ|abend}[;noask]
```

<i>jobselection</i>	See Selecting and Qualifying Jobs on page 9-9 .
<code>succ</code>	Confirm that the job ended successfully.
<code>abend</code>	Confirm that the job ended unsuccessfully (abended).
<code>noask</code>	Do not prompt for confirmation before taking action against each qualifying job.

Usage

The following table shows the affect of `confirm` on jobs in various states.

Initial job state	State after <code>confirm job;succ</code>	State after <code>confirm job;abend</code>
<code>ready</code>	no affect	no affect
<code>hold</code>	no affect	no affect
<code>exec</code>	<code>succp</code>	<code>abenp</code>
<code>abenp</code>	<code>succp</code>	no affect
<code>succp</code>	no affect	no affect
<code>pend</code>	<code>succ</code>	<code>abend</code>
<code>done</code>	<code>succ</code>	<code>abend</code>
<code>succ</code>	no affect	no affect
<code>abend</code>	<code>succ</code>	no affect
<code>fail</code>	no affect	no affect
<code>susp</code>	no affect	no affect
<code>skel</code>	no affect	no affect
any EXTERNAL job	<code>succ</code>	<code>abend</code>

Changing a job from `abend` to `succ` does not require the `confirmed` keyword. For more information about job confirmation, see [confirmed](#) on page 8-56. For

more information about EXTERNAL jobs, see appendix D, *Internetwork Dependencies*.

Examples

1. Issue `succ` confirmation for job3 in schedule `misdly`:

```
confirm misdly.job3;succ
```

2. Issue `abend` confirmation to job number 234:

```
conf 234;abend
```

console

Assign Maestro's console and set its message level.

Security: You must have `console` access to the `cpu`.

```
console [sess|sys][;level=msglevel]
```

<code>sess</code>	Send Maestro's console messages and prompts to <code>stdout</code> (terminal).
<code>sys</code>	Stop sending Maestro's console messages and prompts to <code>stdout</code> . This occurs automatically when you exit Conman.
<code>msglevel</code>	The level of Maestro messages in the range 0-4. The levels are: <ul style="list-style-type: none">0 no messages (the default on fault-tolerant agent <code>cpus</code>).1 exception messages such as operator prompts, and job abends.2 level 1, plus schedule successful messages.3 level 2, plus job successful messages (the default on the master domain manager).4 level 3, plus job launched messages.

Usage

By default, Maestro's control processes write console messages and prompts to standard list files. You can also have them directed to the `syslog` daemon.

If the command is entered with no options, the current state of the Maestro console is displayed.

Examples

1. Begin writing console messages and prompts to `stdout`, and change the message level to 1:

```
console sess;level=1
```

2. Stop writing console messages and prompts to `stdout`, and change the message level to 4:

```
cons sys;l=4
```

3. Display the current state of the Maestro console:

cons

Console is #J675, level 2, session

675 is the pid of the user's shell.

continue

Ignore the next command error.

```
continue
```

Usage

This command is useful when commands are entered non-interactively. It instructs Conman to continue executing commands even if the next command, following `continue`, results in an error. This command is not needed when you enter commands interactively because Conman will not terminate on an error.

Example

```
conman "continue&cancel=176&rerun job=sked5.job3"
```

deldep job

Delete job dependencies.

Security: You must have `deldep` access to the job.

```
{deldep} jobselection;dependency[;...][;noask]
{ddj }
```

jobselection See [Selecting and Qualifying Jobs on page 9-9](#).

dependency See [Job Dependencies](#) below. At least one dependency must be specified in order for this command to function properly.

noask Do not prompt for confirmation before taking action against each job.

Job Dependencies

Following is a syntax summary of job dependencies. For a complete description of job dependencies refer to [The Scheduling Language](#) on page 8-50.

```
at
confirmed
every
follows=[cpu#]sched{.job|@}|job|net::net_dep][,...]
needs=[num] [cpu#]resource][,...]
opens=[cpu#"path"[(qualifier)]][,...]
priority
prompt=[cpu#]msgnum|promptname][,...]
until

Wildcards permitted in cpu, sched, job, resource, path,
promptname.
```

Usage

If `priority` is deleted from a job, it reverts back to its original scheduled priority. When deleting an `opens` dependency, you can include only the

filename (basename), and Conman performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are deleted.

Examples

1. Delete a resource dependency from job3 in sked5:

```
deldep sked5.job3;needs=2 tapes
```

2. Delete all `follows` dependencies from job4 in sked3:

```
ddj sked3.job4;follows
```

deldep sched

Delete schedule dependencies.

Security: You must have `deldep` access to the schedule.

```
{deldep} schedselection;dependency[;...][;noask]
{dds }
```

schedselection See [Selecting and Qualifying Schedules](#) on page 9-18.

dependency See [Schedule Dependencies](#) below. At least one dependency must be specified in order for this command to function properly.

noask Do not prompt for confirmation before taking action against each qualifying schedule.

Schedule Dependencies

Following is a syntax summary of schedule dependencies. For a complete description of schedule dependencies refer to [The Scheduling Language](#) on page 8-50.

```
at
carryforward
follows=[cpu#]sched[.job]|net::net_dep[[, ...]
limit
needs=[num] [cpu#]resource[[, ...]
opens=[cpu#]"path"[(qualifier)][, ...]
priority
prompt=[cpu#]msgnum|promptname[[, ...]
until
Wildcards permitted in cpu, sched, job, resource, path,
promptname.
```

Usage

If `priority` is deleted from a schedule, it reverts back to its original scheduled priority. When deleting an `opens` dependency, you can include only the filename (basename), and Conman performs a case-insensitive search for matching files, ignoring the directory names. Dependencies on all matching files are deleted.

Examples

1. Delete the carryforward flag from sked5:

```
deldep sked5;carryforward
```

2. Delete all prompt dependencies from sked3:

```
dds sked3;prompt
```

display file

Display the contents of a file.

Security: You must have read access to the file.

```
{display file= }path[;offline]
{df           }
```

- path* The path name of the file . The name must be enclosed in quotes (") if it contains characters other than the following: alphanumeric, dashes (-), slashes (/), and underscores (_). Wildcards are permitted. The file must be accessible from your login cpu.
- offline* Send the output of the command to the Conman output device. For information about this device, see [Offline Output](#) on page 9-3.

Examples

1. Display the file `/usr/lib/maestro/jclfiles/arjob3`:
`display file=/usr/lib/maestro/jclfiles/arjob3`
2. Display the file `/usr/lib/maestro/jclfiles/gljob` offline:
`df /usr/lib/maestro/jclfiles/gljob;off`

display job

Display the contents of a job's script file.

Security: You must have `display` access to the job.

```
{display job= }jobselection[;offline]
{dj          }
```

jobselection See [Selecting and Qualifying Jobs on page 9-9](#). The job's script file must be accessible from your login cpu.

offline Send the output of the command to the Conman output device. For information about this device, see [Offline Output](#) on page 9-3.

Examples

1. Display the script file for job4 in sked9:
`display job=sked9.job4`
2. Display the script file for job6 in sked 3 offline:
`dj sked3.job6;off`

display sched

Display a schedule.

Security: You must have `display` access to the schedule.

```
{display sched= }schedselection[;offline]
{ds }
```

schedselection See [Selecting and Qualifying Schedules](#) on page 9-18.

offline Send the output of the command to the Conman output device. For information about this device, see [Offline Output](#) on page 9-3.

Note: Maestro's `mozart` directory on the master domain manager must be accessible from your login cpu.

Examples

1. Display schedule sked9:
`display sched=sked9`
2. Display schedule sked3 offline:
`ds sked3;off`

exit

Exit the Conman program.

```
exit
```

Usage

When in `help` mode, Conman returns to command-input mode.

Examples

```
exit  
e
```

fence

Change the Maestro job fence on a cpu. Jobs will not be launched on the cpu if their priorities are less than or equal to the jobfence.

Security: You must have **fence** access to the cpu.

```
fence [cpu];pri[;noask]
```

<i>cpu</i>	The name of the cpu. If omitted, the default is your login cpu.
<i>pri</i>	A priority level in the range 0-99, or hi or go . Entering "system" sets the job fence to zero.
noask	Do not prompt for confirmation before changing the job fence on each qualifying cpu.

Usage

Maestro's job fence prevents low priority jobs from being launched, regardless of the priorities of their schedules. It is possible, therefore, to hold back low priority jobs in high priority schedules, while allowing high priority jobs in low priority schedules to be launched.

When you first start Maestro following installation, the cpu jobfence is set to zero. Once you change the jobfence, it is carried forward during pre-production processing to the next day's **symphony** file.

To display the current setting of Maestro's jobfence, use the **status** command.

Examples

1. Change Maestro's job fence on cpu site4:


```
fence site4;20
```
2. Change Maestro's job fence on this cpu:


```
f ;40
```
3. Prevent all jobs from being launched by Maestro on cpu tx3:


```
f tx3;go
```
4. Change Maestro's job fence to zero on this cpu:


```
f ;system
```

help

Display command help information. Not supported on Windows NT.

```
help command
```

command Any Conman or system command. For Conman commands, enter the full command name—abbreviations and short forms are not supported. In the case of command names consisting of two words, enter the first word, and all versions of the command will be displayed. For example, entering "help display" will display information about `display file`, `display job`, and `display sched`. You can also enter the following keywords:

<code>commands</code>	List of Conman commands.
<code>jobselect</code>	Information about selecting and qualifying jobs.
<code>jobstates</code>	List of job states.
<code>schedselect</code>	Information about selecting and qualifying schedules.
<code>schedstates</code>	List of schedule states.

Examples

1. Display a list of all Conman commands:
`help commands`
2. Display information about the `fence` command:
`help fence`
3. Display information about the `altpri job` and `altpri sched` commands:
`h altpri`

kill

Stop an executing job. On UNIX, this is accomplished with a UNIX `kill` command.

Security: You must have `kill` access to the job.

```
kill jobselection[ ;noask]
```

jobselection See [Selecting and Qualifying Jobs on page 9-9](#).

`noask` Do not prompt for confirmation before killing each qualifying job.

Usage

The kill operation is not performed by Conman, but is executed by Jobman, so there may be a short delay.

Killed jobs terminate in the `abend` state. Any jobs or schedules that are dependent on a killed job are not released. Killed jobs can be rerun.

Examples

1. Kill the report job in schedule apwkly on cpu site3:

```
kill site3#apwkly.report
```

2. Kill job number 456:

```
k 456
```

limit cpu

Change the job limit for Maestro on a cpu.

Security: You must have `limit` access to the cpu.

```
{limit cpu= }cpuid;limit[;noask]
{lc      }
```

cpuid The name of the cpu. Wildcards are permitted. The default is your login cpu. Cpu classes are not valid.

limit The job limit in the range 0-1024. Entering "system" sets the job limit to zero. If a limit of zero is set, no jobs, other than `hi` and `go` jobs, are launched by Maestro on the cpu.

noask Do not prompt for confirmation before changing the job limit on each qualifying cpu.

Usage

To display the current Maestro job limit on this cpu, use the `status` command.

When you first start Maestro following installation, the cpu job limit is set to zero, and must be raised before any jobs will be launched. Once you change the limit, it is carried forward during pre-production processing to the next day's `Symphony` file.

Maestro attempts to launch as many jobs as possible up to its job limit. There is a practical limit to the number of processes that can be started on a given system. If the limit is reached, the system responds with a message indicating that system resources are temporarily not available. When a Maestro job cannot be launched for this reason, it enters the `fail` state. You can try lowering Maestro's job limit to prevent this from occurring.

Examples

1. Change the job limit on cpu site3:
`limit cpu=site3;25`
2. Change the job limit on this cpu: `lc ;12`
3. Change the job limit on cpu rx12: `lc rx12;6`

limit sched

Change the job limit for a schedule.

Security: You must have `limit` access to the schedule.

```
{limit sched= }schedselection[;limit][;noask]
{ls }
```

<i>schedselection</i>	See Selecting and Qualifying Schedules on page 9-18.
<i>limit</i>	The job limit in the range 0-99. If <i>limit</i> is omitted, the schedule's job limit reverts back to the original, or 99 if an original job limit was not specified. If a limit of zero is specified, no further jobs are launched from the schedule.
<i>noask</i>	Do not prompt for confirmation before changing the job limit on each qualifying schedule.

Examples

1. Change the job limit on all sales schedules:

```
limit sched=sales@;4
```

2. Change the job limit on schedule apwkly:

```
ls apwkly;6
```

link

Open Maestro links for inter-cpu communications.

Security: You must have `link` access to the destination cpu.

```
{link} [domain!]cpu[;noask]
{lk }
```

- domain* The name of the domain in which the links are opened. Wildcards are permitted. The default is the domain of the target cpu. If the cpu is wildcarded (for example: `link @`), the default is the domain in which `conman` is running.
- cpu* The name of the destination cpu. Wildcards are permitted. This can be any cpu, except an extended agent.
- noask* Do not prompt for confirmation before linking to each qualifying cpu.

Usage

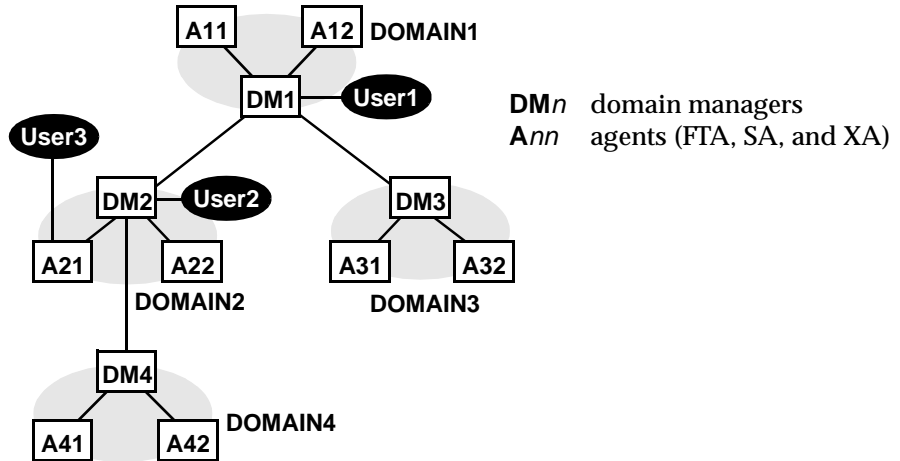
If the `autolink` flag was included in a cpu definition, the link is opened automatically each time you start Maestro. If `autolink` was omitted or `off`, you must use this command, along with `unlink`, to control the link. For information about `autolink` see [CPU Definition](#) on page 4-9.

Assuming that a user has link access to the cpus being linked, the following rules apply:

- A user running the Console Manager on the master domain manager can link any cpu in the network.
- A user running the Console Manager on a domain manager other than the master can link any cpu in its local domain or subordinate domain. The user cannot link cpus in a peer domain.
- A user running the Console Manager on an agent can link any cpu in its local domain.
- To link a subordinate domain, it is not necessary that the intervening links be open.

Examples

On the following page, the table shows the links opened by different **link** commands executed by users at different locations in the depicted Maestro network.



Links opened by:			
Command	User1	User2	User3
<code>link @!@</code>	All links are opened.	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4 DM4-A41 DM4-A42	DM2-A21 DM2-A22
<code>link @</code>	DM1-A11 DM1-A12 DM1-DM2 DM1-DM3	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4	DM2-A21 DM2-A22
<code>link DOMAIN3!@</code>	DM3-A31 DM3-A32	Not allowed.	Not allowed.
<code>link DOMAIN4!@</code>	DM4-A41 DM4-A42	DM4-A41 DM4-A42	Not allowed.
<code>link DM2</code>	DM1-DM2	na	DM2-A21
<code>link A42</code>	DM4-A42	DM4-A42	Not allowed.
<code>link A31</code>	DM3-A31	Not allowed.	Not allowed.

listsym

List **symphony** log files.

Security: You must have **display** access to the **symphony** file.

```
listsym [;offline]
```

offline Send the output of the command to the Conman output device.
For information about this device, see [Offline Output](#) on page 9-3.

Output Format

Schedule Date	The date used by the schedulr command to select schedules for execution.
Actual Date	The date batchman began executing the symphony file.
Start Time	The time batchman began executing the symphony file.
Log Date	The date the symphony file was logged by the stageman command.
Run Num	The run number assigned to the symphony file. These are used internally for Maestro network synchronization.
Size	The size of the log file in records.
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

```
listsym  
lis ;off
```

recall

Display pending prompts.

Security: You must have `display` access to the prompts.

```
{recall } [cpu][;offline]
{rc     }
```

<code>cpu</code>	The name of the <code>cpu</code> on which the prompt was issued. If omitted, only prompts for the login <code>cpu</code> , and global prompts are displayed.
<code>offline</code>	Send the output of the command to the Conman output device. For information about this device, see Offline Output on page 9-3.

Note: For information about Maestro-created prompts, see [Carry Forward Prompts](#) on page 3-21.

Output Format

State	The state of the prompt, always ASKED for pending prompts.
Message or Prompt	For global prompts, the message number, the name of the prompt, and the message text. For literal prompts, the message number, the name of the job or schedule, and the message text. Prompts are listed in the following format:

```
num (prompt-name) text
num (cpu#sched.job) text
```

Examples

1. Display pending prompts on this `cpu`:

```
recall
```

```
or:
```

```
rc
```

2. Display pending prompts on cpu site3:
`rc site3`
3. Display pending prompts on all cpus offline:
`rc @;offline`

redo

Edit and re-execute the previous command.

```
redo
```

Directives

When you issue redo, Conman displays the previous command, so that it can be edited and re-executed. Use the spacebar to move the cursor under the character to be modified, and enter one of the following directives.

d <i>[dir]</i>	Delete the character above the d . This can be followed by other directives.
i <i>text</i>	Insert text before the character above the i .
r <i>text</i>	Replace one or more characters with text, beginning with the character above the r . Replace is implied if no other directive is entered.
> <i>text</i>	Append text to the end of the line.
>d <i>[dir text]</i>	Delete characters at the end of the line. This can be followed by another directive or text.
>r <i>text</i>	Replace characters at the end of the line with text.

Directive Examples

ddd	Delete the three characters above the d 's.
iabc	Insert "abc" before the character above the i .
rabc	Replace the three characters, starting with the one above the r , with "abc".
abc	Replace the three characters above "abc" with "abc".
d diabc	Delete the character above the first d , skip one space, delete the character above the second d , and insert "abc" in its place.
>abc	Append "abc" to the end of the line.
>ddabc	Delete the last two characters in the line, and append "abc" in their place.
>rabc	Replace the last three characters in the line with "abc".

Examples

redo
setsm 4
 iy (insert a character, press the Return key)
setsym 4 (press the Return key to execute command)

redo
setsym 4
 5 (implicit replace, press the Return key)
setsym 5 (press the Return key to execute command)

release job

Release job dependencies.

Security: You must have **release** access to the job.

```
{release job= }jobselection[;dependency[;...]][;noask]
{rj          }
```

jobselection See [Selecting and Qualifying Jobs on page 9-9](#).

dependency See [Job Dependencies](#) below. If none are specified, all dependencies are released.

noask Do not prompt for confirmation before taking action against each qualifying job.

Job Dependencies

Following is a syntax summary of job dependencies. For a complete description of job dependencies refer to [The Scheduling Language](#) on page 8-50.

```
at
confirmed
every
follows=[cpu#]sched{.job|@}|job|net::net_dep[[, ...]]
needs=[num] [cpu#]resource[[, ...]]
opens=[cpu#]"path"[(qualifier)][, ...]]
priority
prompt=[cpu#]msgnum|promptname[[, ...]]
until

Wildcards permitted in cpu, sched, job, resource, path,
promptname.
```

Usage

When releasing an **opens** dependency, you can include only the filename (basename), and Conman will perform 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 may not be available. This may cause the Available units in a `showresources` to display a negative number.

Examples

1. Release job3 in schedule ap from all of its dependencies:

```
release job=ap.job3
```

2. Release job2 in schedule skedr from all of its `opens` dependencies:

```
rj skedr.job2;opens
```

3. Release all jobs on cpu site4 from their dependencies on a prompt named glprmt:

```
rj site4#@. @;prompt=glprmt
```

release sched

Release schedule dependencies.

Security: You must have **release** access to the schedule.

```
{release sched=} schedselection[;dependency[;...]][;noask]
{rs }
```

schedselection See [Selecting and Qualifying Schedules](#) on page 9-18.

dependency See [Schedule Dependencies](#) below. If none are specified, all dependencies are released.

noask Do not prompt for confirmation before taking action against each qualifying schedule.

Schedule Dependencies

Following is a syntax summary of schedule dependencies. For a complete description of schedule dependencies refer to [The Scheduling Language](#) on page 8-50.

```
at
carryforward
follows=[cpu#]sched[.job]|net::net_dep[[, ...]
limit
needs=[num] [cpu#]resource[[, ...]
opens=[cpu#] "path" [(qualifier)] [[, ...]
priority
prompt=[cpu#] msgnum | promptname [[, ...]
until
Wildcards permitted in cpu, sched, job, resource, path, promptname.
```

Usage

When releasing an **opens** dependency, you can include only the filename (basename), 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 schedule is given the required number of units of the resource, even though they may not be available. This may cause the Available units in a `showresources` to display a negative number.

Examples

1. Release schedule `ap` from all of its dependencies:

```
release sched=ap
```

2. Release schedule `sked5` from all of its `opens` dependencies:

```
rs sked5;opens
```

3. Release all schedules on `cpu site3` from their dependencies on schedule `main#sked23`:

```
rs site3#@;follows=main#sked23
```

reply

Reply to a Maestro prompt.

Security: You must have **reply** access to the named (global) prompt. To reply to a literal prompt by message number, you must have **reply** access to the prompt and the associated job or schedule.

```
reply {prompt          }{;reply}[;noask]
      { [cpu#]msgnum  }
```

<i>prompt</i>	The name of a global prompt. Wildcards are permitted.
<i>cpu</i>	The name of the cpu on which a literal prompt was issued.
<i>msgnum</i>	The message number of a literal prompt. Message numbers can be displayed with the recall and showprompts commands.
<i>reply</i>	The reply, either "y" (yes) or "n" (no).
noask	Do not prompt for confirmation before taking action against each qualifying prompt.

Usage

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 re-issued.

Prompts can be replied to before they are issued. The **showprompts** command can be used to display all prompts, whether issued or not.

Examples

1. Reply "yes" to global prompt arprmt:
`reply arprmt;y`
2. Reply "no" to message number 24 on cpu site4:
`rep site4#24;n`

rerun

Rerun a job.

Security: You must have **rerun** access to the job.

```
{rerun }jobselection [ ;from=[cpu#]job[ ;at=time][ ;pri=pri][ ;noask]
{rr    }                [ ;step=step                ]
```

<i>jobselection</i>	See Selecting and Qualifying Jobs on page 9-9 .
<i>cpu</i>	The name of the cpu on which the from job runs. If omitted, the default is your login cpu .
<i>job</i>	The name of the from job. The script file or command for this job is launched instead of the original job. If omitted, the original job is launched.
<i>time</i>	The time the job will be launched expressed as: <i>hhmm</i> [+ <i>n</i> days <i>date</i>] <i>hhmm</i> The hour and minute. <i>n</i> Offset in days from today. <i>date</i> A specific date. The default format is <i>mm/dd/yy</i> . If omitted, the job is launched as soon as possible based on its dependencies.
<i>pri</i>	The priority to be assigned to the rerun job. If omitted, the job is launched at the same priority as the original job.
noask	Do not prompt for confirmation before taking action against each qualifying job.
<i>step</i>	The job is rerun using this name in place of the original job name. See Usage below.

Usage

To rerun a job, it must be in the **succ**, **fail** or **abend** state. A rerun job is placed in the same schedule as the original job, and inherits the original'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: `rerun` can be used for EXTERNAL jobs in the `error` state. EXTERNAL jobs represent jobs and schedules that have been specified as internetwork dependencies. `rerun` instructs Conman to start checking the state of the EXTERNAL job. The job state is set to `extrn` immediately after a `rerun` is performed. For more information on internetwork dependencies see appendix D.

When the `;from` option is used, the Global Option `retain rerun job names` determines which name will be given to rerun jobs. If the option is set to "y", rerun jobs retain their original job names. If the option is set to "n", rerun jobs are given the `from` job names. See *Global Options* on page 2-1 for more information.

In Conman displays, `rerun` jobs appear with the notation ">>rerun as". To refer to a `rerun` job in another command, like `altpri`, you must use the original job name.

When a UNIX job is `rerun` with the `;step` option, the job runs with `step` in place of the original job name. Within a job script, you can use Maestro's `jobinfo` command to return the job name and make the script execute differently for each iteration. For example:

```
...
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 appear with the notation `>>rerun step`.

Examples

1. Rerun job4 from sked1 on cpu main:

```
rerun main#sked1.job4
```

2. Rerun job5 from sked2 using the script file for jobx. Give the job an at time of 6:30 p.m., and a priority of 25:

```
rr sked2.job5;from=jobx;at=1830;pri=25
```

3. Rerun job3 from sked4 using the job name jstep2:

```
rr sked4.job3;step=jstep2
```

resource

Change the number of total units of a resource.

Security: You must have **resource** access to the resource.

```
resource [cpu#]resource;num[;noask]
```

<i>cpu</i>	The name of the cpu on which the resource is defined. If omitted, the default is your login cpu.
<i>resource</i>	The name of the resource.
<i>num</i>	The number of available units to be assigned to the resource (0-1024).
noask	Do not prompt for confirmation before taking action against each qualifying resource.

Examples

1. Change the number of units of resource tapes to 5:

```
resource tapes;5
```

2. Change the number of units of resource jobslots on cpu site2 to 23:

```
res site2#jobslots;23
```

setsym

Set the log file pointer to a specific **symphony** log file. Subsequent display commands are executed against that log file. You cannot modify the information in a log file.

Security: Available to all users.

```
setsym [lognum]
```

lognum The number of the **symphony** log file. If omitted, the pointer is returned to the current **symphony** file. Use the **listsym** command to list log file numbers.

Examples

1. Set the log file pointer to 5:

```
setsym 5
```

2. Return the log file pointer to the current **symphony** file:

```
set
```

showcpus

Display cpu and cpu link information.

Security: Available to all users.

```
{showcpus } [[domain!]cpu][;info][;link][;offline]
{sc       }
```

<i>domain</i>	The name of the domain to be displayed. The default is the domain of the specified <i>cpu</i> .
<i>cpu</i>	The name of the <i>cpu</i> on to be displayed. The default is the <i>cpu</i> on which conman is running.
info	See Output Format (:info) below.
link	See Output Format (:link) below.
offline	Send the output of the command to the Conman output device. For more information, see Offline Output on page 9-3.

Output Format

CPUID	The name of the <i>cpu</i> to which this information applies.
RUN	The run number of the Production Control file (<i>symphony</i>).
NODE	The node type and <i>cpu</i> type. Node types are: UNIX, WINT, MPEV, MPIX, and OTHER. <i>Cpu</i> types are: MASTER, MANAGER, FTA, S-AGENT, and X-AGENT.
LIMIT	The Maestro job limit.
FENCE	The Maestro job fence.
DATE TIME	The date and time batchman started executing the <i>symphony</i> file.
STATE	The state of the <i>cpu</i> link between your login <i>cpu</i> and the <i>cpu</i> named as CPUID. Up to five characters are displayed as follows:

```
[L] [D] [I] [J] [W]
      [T]          [M]
      [H]          [H]
      [X]          [X]
      [?]
```

L The link is open (linked).

- D The link is DS.
- T The link is TCP/IP.
- I `jobman` has completed startup initialization.
- J `jobman` is running.
- W The cpu is linked via TCP/IP.
- M The cpu is linked via DS.
- X The cpu is linked as an extended agent.
- H The cpu is linked via the host.

Note: If the cpu running Conman is the x-agent's host , the state of the x-agent is `LXI JX`. If the cpu running Conman is not the x-agent's host, the state of the x-agent is `LHI JH`.

- METHOD For extended agents only. The name of the access method specified in the cpu definition.
- DOMAIN The name of the domain in which the cpu is a member.

Output Format (;info)

- CPUID The name of the cpu to which this information applies.
- VERSION The version of `jobman`.
- TIMEZONE The time zone (the value of the `tz` environment variable). For extended agents, the time zone of its host.
- INFO The Operating System version and cpu model. For extended agents, no information is returned.

Output Format (;link)

- CPUID The name of the cpu to which this information applies.
- HOST The name of the cpu on which `batchman` is running. For domain managers and fault-tolerant agents, this is the same as CPUID. For standard agent cpus, this is the name of the domain manager. For extended agents, this is the name of the host cpu.

FLAGS	<p>The attributes of the link from your login cpu (where Conman is running) to CPUID. Up to four characters are displayed as follows:</p> <p>AFS [D] [T] [B]</p> <p>A Auto-link is on. F Full Status mode on. S The mailman server id (a-z, 0-9). D The link definition contains only DS commands. T The link definition contains only TCP statements. B The link definition contains both DS commands and TCP statements.</p>
ADDR	The TCP port number of CPUID from its cpu definition.
NODE	The node name of CPUID from its cpu definition.

Examples

1. Display information about this cpu in the `info` format:

```
showcpus ;info
```

2. Display link information offline for all cpus:

```
sc @;link;off
```

showdomain

Show information about Maestro domains.

Security: Available to all users.

```
showdomain [[domain][;info][;offline]
```

- domain* The name of the domain to be displayed. Wildcards are permitted. The default is the domain in which **conman** is running.
- info* Display in the “info” format. See [Output Format—;info](#) below.
- offline* Send output to the conman output device. See [Offline Output](#) on page 9-3.

Output Format

- DOMAIN The domain name.
- MANAGER The name of the domain manager.
- PARENT The name of the parent domain.

Output Format—;info

- DOMAIN The domain name.
- MEMBER-CPUS The names of all cpus in the domain.
- CPU-TYPE The cpu type of each cpu: MASTER, MANAGER, FTA, S-AGENT, and X-AGENT.

Examples

1. Display information about the masterdm domain:
`showdomain masterdm`
2. Display the member cpus in all domains:
`showdomain @;info`

showfiles

Display the status of file dependencies.

Security: Available to all users.

```
{showfiles } [[cpu#]path][;state[;...]][;keys      ] [;offline]
{sf         }                               [;deps [;keys ] ]
                                                [;info  ]
                                                [;logon ]
```

- cpu* The name of the cpu on which the file exists. Wildcards are permitted. If omitted, it defaults to your login cpu.
- path* The path name of the file. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumerics, dashes ?(-), slashes (/), and underscores (_). Wildcards are permitted. If omitted, all file dependencies are displayed.
- state* If included, only file dependencies in these states are displayed. If omitted, file dependencies in all states are displayed. The states are:
- | | |
|---------|--|
| yes | File exists and is available. |
| no | File is unavailable, or does not exist. |
| ? | Availability is being checked. |
| <blank> | Not yet checked, or the file was available and used to satisfy a job or schedule dependency. |
- keys* See [Output Format \(:keys\)](#) and [Output Format \(:deps:keys\)](#) below.
- deps* See [Output Format \(:deps\)](#) below.
- info* See [Output Format \(:deps:info\)](#) below.
- logon* See [Output Format \(:deps:logon\)](#) below.
- offline* Send the output of the command to the Conman output device. For more information, see [Offline Output](#) on page 9-3.

Output Format

Exists The state of the file dependency (see *state* on [page 9-77](#)).

File Name The full path name of the file.

Output Format (;keys)

The files used in `opens` dependencies are listed one file per line. Directory names are not included. Each file is listed in the following format:

cpu#filename

Output Format (;deps)

The files used in `opens` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the basic `showjobs` format. Schedules are listed in the basic `showschedules` format.

Output Format (;deps;keys)

Jobs and schedules that have `opens` dependencies are listed one per line in the following format:

cpu#sched[.job]

Output Format (;deps;info)

The files used in `opens` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the `showjobs;info` format. Schedules are listed in the basic `showschedules` format.

Output Format (;deps;logon)

The files used in `opens` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the `showjobs;logon` format. Schedules are listed in the basic `showschedules` format.

Examples

1. Display the status of a file dependency for `/usr/lib/mis/data4`:

```
showfiles /usr/lib/mis/data4
```

```
Exists      File Name
NO          /usr/lib/mis/data4
```

2. Display offline the status of all file dependencies on all cpus in the `deps` format:

```
sf @#@;deps;offline
```

showjobs

Display job status.

Security: Available to all users.

```

{showjobs } [jobselection] [;keys          ][;short ][;offline]
{sj       }                [;info          ][;single ]
                                [;step          ]
                                [;logon         ]
                                [;stdlist[;keys] ]
                                [;deps[;keys  ] ]
                                [;info         ]
                                [;logon        ]

```

- jobselection* See [Selecting and Qualifying Jobs on page 9-9](#). If omitted, the default is all jobs on your login cpu.
- keys** See [Output Format \(:keys\)](#), [Output Format \(:stdlist:keys\)](#) and [Output Format \(:deps:keys\)](#) below.
- info** See [Output Format \(:info\)](#) and [Output Format \(:deps:info\)](#) below.
- step** See [Output Format \(:step\)](#) below. Not supported on Windows NT.
- logon** See [Output Format \(:logon\)](#) and [Output Format \(:deps:logon\)](#) below.
- stdlist** See [Output Format \(:stdlist\)](#) and [Output Format \(:stdlist:keys\)](#) below.
- deps** See [Output Format \(:deps\)](#) below.
- short** Shorten the display for **every** and **rerun** jobs to include only: 1) the first iteration, 2) jobs in different states, and 3) exactly matched jobs.
- single** Select only the parent job in a chain that can include reruns, repetitions, and recovery jobs. The job must be identified by job number in *jobselection*. This is particularly useful with the **;stdlist** option.
- offline** Send the output of the command to the Conman output device. For more information, see [Offline Output](#) on page 9-3.

Output Format

CPU	The cpu on which the schedule runs.
Schedule	The name of the schedule.
Job	The name of the job. The following notation may precede a job name: <ul style="list-style-type: none"> >> rerun as Job rerun with rerun command, or as a result of automatic recovery. >> rerun step Job rerun with 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 schedule or job. Job states are: <ul style="list-style-type: none"> abend Job terminated with a non-zero exit code. abemp "abend" confirmation received, but job not completed. add Job is being submitted. done (MPE only) Job completed in an unknown state. error For an internetwork dependency, an error occurred while checking for the remote status. exec Job is executing. extrn For an internetwork dependency, unknown status. An error occurred, a rerun action was just performed on the EXTERNAL job, or the remote job or schedule does not exist. fail Unable to launch job. fence Job 's priority is below the fence. hold Job awaiting dependency resolution. intro Job introduced for launching by the system. pend Job completed, awaiting confirmation. ready Job ready to launch, all dependencies resolved. sched Job's at time has not arrived. succ Job completed with zero exit code. succp "succ" confirmation received, but job not completed. susp (MPE only) Job suspended by breakjob command.

wait (X-agent jobs and MPE only) Job is being moved to **ready**.
waitd (MPE only) Job is in the **wait** state, and is deferred.

Schedule states are:

abend Schedule terminated unsuccessfully.
add Schedule has just been submitted.
cancel Schedule cancelled.
exec Schedule is executing.
hold Awaiting dependency resolution.
ready Dependencies resolved, ready to launch.
stuck Execution interrupted. No jobs will be launched without operator intervention.
succ Schedule completed successfully.

Pr The schedule or job priority. A plus sign (+) preceding the priority means the job has been launched.

(Est)Start Schedule or job start time. If in parentheses, it is the estimated start time (that is, the **at** time). If the time is more than 24 hours in the past or future, the date is listed instead of the time.

(Est)Elapse Schedule or job run time. If in parentheses, it is an estimation based on logged statistics.

dependencies A list of job dependencies and comments. Any combination of the following may be listed:

- For a **follows** dependency, a schedule or job name.
- For an **opens** dependency, the file name. If the file resides on an x-agent and its name is longer than 28 characters, it is displayed as follows:
...last_25_characters_of_filename
- For a **needs** dependency, a resource name enclosed in hyphens (-). If the number of units requested is greater than one, the number is displayed before the first hyphen.
- For an **every** rate, the repetition rate preceded by an ampersand (&).
- For an **until** time, the time preceded by a less than sign (<).
- For a **prompt** dependency, the prompt number

displayed as: `#num`. For global prompts, this is followed by the prompt name in parentheses.

- For executing jobs, the process id (PID). Shown as: `#Jnnnnn`.
- Jobs submitted using Maestro's `at` and `batch` commands are labeled: `[User jcl]`.
- Cancelled jobs are labeled: `[Cancelled]`.
- Jobs cancelled with `;pend` option are labeled: `[Cancel Pend]`.
- Jobs whose `until` times have expired and jobs cancelled with `;pend` option whose `until` times have expired 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.

Output Format (;keys)

The jobs are listed one per line. Each job is listed in the following format:

```
cpu#sched.job
```

Output Format (;info)

CPU	The cpu on which the schedule runs.
Schedule	The name of the schedule.
Job	The name of the job. The following notation may precede a job name:
>> rerun as	Job rerun by a <code>rerun</code> command, or as a result of automatic recovery.
>> every run	The second and subsequent runs of an <code>every</code> job.
>> recovery	The run of a recovery job.

Job File	The path name of the job's script file. Long path names may wrap the display causing incorrect paging. To avoid this, the output can be piped to more. For example: <code>conman "sj;info" more</code>
Opt	The recovery option, if any: RE Rerun CO Continue ST Stop
Job	The name of the recovery job, if any.
Prompt	The recovery prompt number, if any.

Output Format (;step)

Not supported on Windows NT.

CPU	The cpu on which the schedule runs.
Schedule	The name of the schedule.
Job	The name of the job. The following notation may precede a job name: >>rerun as Job rerun by a 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 schedule or job. See <i>Output Format</i> on page 9-81 for a description of states.
Job#	The process id (PID). Shown as: #Jnnnnn.
Step	A list of the descendent processes associated with the job. For extended agent jobs, only the descendent processes running on the host are listed.

Output Format (;logon)

CPU	The cpu on which the schedule runs.
Schedule	The name of the schedule.

Job	The name of the job. The following notation may precede a job name: >> rerun as Job rerun by a 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 schedule or job. See <u>Output Format</u> on page 9-81 for a description of states.
Job#	The process id (PID). Shown as: #Jnnnnn.
Logon	The user name under which the job executes.

Output Format (;stdlist)

The standard list files for the job selection are displayed in their standard format. For a description of standard list files refer to [Standard List Files](#) on page 3-7.

Output Format (;stdlist;keys)

The fully qualified path to the standard list files for the job selection are listed one per line. For a description of standard list files and their directories refer to [Standard List Files](#) on page 3-7.

Output Format (;deps)

The jobs used in **follows** dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the basic **showjobs** format. Schedules are listed in the basic **showschedules** format.

Output Format (;deps;keys)

Jobs and schedules that have **follows** dependencies are listed one per line. The jobs and schedules are listed in the following format:

```
cpu#sched[.job]
```

Output Format (;deps;info)

The jobs used in `follows` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the `showjobs;info` format. Schedules are listed in the basic `showschedules` format.

Output Format (;deps;logon)

The jobs used in `follows` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the `showjobs;logon` format. Schedules are listed in the basic `showschedules` format.

Examples

1. Display the status of all jobs in all `acctg` schedules on `cpu site3`:

```
showjobs site3#acctg@.@
```

2. Display the status of all jobs on this `cpu` in the `logon` format:

```
sj ;logon
```

3. Display the status of all jobs in the `hold` state on all `cpus` in the `deps` format:

```
sj @#@.@+state=hold;deps
```

4. Display all of the standard list files for the job `gl` in the schedule `arec` on `cpu site2`:

```
sj site2#arec.gl;stdlist
```

showprompts

Display the status of prompts.

Security: Available to all users.

```
{showprompts} [prompt ] [;state[;...]] [;keys ] [;offline]
{sp } [[cpu#]msgnum ] [;deps [;keys ] ]
[;info ]
[;logon ]
```

prompt The name of a global prompt. Wildcards are permitted.

cpu The name of the cpu on which a literal prompt was issued. The default is your login cpu.

msgnum The message number of a literal prompt.

state If included, only prompt dependencies in these states are displayed. If omitted, dependencies in all states are displayed. The states are:

YES	Replied to with "y".
NO	Replied to with "n".
ASKED	Issued, but no reply.
INACT	Not yet issued.

keys See [Output Format \(:keys\)](#) and [Output Format \(:deps:keys\)](#) below.

deps See [Output Format \(:deps\)](#) below.

info See [Output Format \(:deps:info\)](#) below.

logon See [Output Format \(:deps:logon\)](#) below.

offline Send the output of the command to the Conman output device. For more information, see [Offline Output](#) on page 9-3.

Note: For information about Maestro-created prompts, see [Carry Forward Prompts](#) on page 3-21.

Output Format

`State` The state of the prompt. See `state` on [page 9-87](#).

`Message or Prompt` For global prompts, the prompt number, the name of the prompt, and the message text. For literal prompts, the prompt number, the name of the job or schedule, and the message text. Prompts are listed in the following format:

```
num (prompt-name) text
num (cpu#sched.job) text
```

Output Format (;keys)

The prompts are listed one per line. Global prompts are listed with their prompt numbers, and prompt names. Literal prompts are listed with their prompt numbers, and the names of the jobs or schedules in which they appear as dependencies. Prompts are listed in the following format:

```
num (prompt-name)
num (cpu#sched.job)
```

Output Format (;deps)

The prompts used in `prompt` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the basic `showjobs` format. Schedules are listed in the basic `showschedules` format.

Output Format (;deps;keys)

The jobs and schedules that have `prompt` dependencies are listed one per line in the following format:

```
cpu#sched[.job]
```

Output Format (;deps;info)

The prompts used in `prompt` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the `showjobs;info` format. Schedules are listed in the basic `showschedules` format.

Output Format (;deps;logon)

The prompts used in `prompt` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the `showjobs;logon` format. Schedules are listed in the basic `showschedules` format.

Examples

1. Display the status of all prompts on this cpu:

```
showprompts
```

2. Display, in the `deps` format, the status of all mis prompts that have been asked:

```
sp mis@;asked;deps
```

3. Display the status of prompt number 34 on cpu main:

```
sp main#34
```

showresources

Display information about resources.

Security: Available to all users.

```
{showresources } [[cpu#]resource] [;keys          ] [;offline]
{sr             } [;deps [;keys  ] ]
                  [;info  ]
                  [;logon ]
```

- cpu* The name of the cpu on which the resource is defined. Wildcards are permitted. The default is your login cpu.
- resource* The name of the resource. Wildcards are permitted.
- keys* See Output Format (;keys) and Output Format (;deps:keys) below.
- deps* See Output Format (;deps) below.
- info* See Output Format (;deps:info) below.
- logon* See Output Format (;deps:logon) below.
- offline* Send the output of the command to the Conman output device. For more information, see Offline Output on page 9-3.

Output Format

- CPU The cpu 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 schedule.
- Used By The name of the job or schedule.

Output Format (;keys)

The resources are listed one per line in the following format:

```
cpu#resource-name
```

Output Format (;deps)

The resources used in **needs** dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the basic **showjobs** format. Schedules are listed in the basic **showschedules** format.

Output Format (;deps;keys)

The jobs and schedules that have **needs** dependencies are listed one per line. The jobs and schedules are listed in the following format:

```
cpu#sched[.job]
```

Output Format (;deps;info)

The resources used in **needs** dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the **showjobs;info** format. Schedules are listed in the basic **showschedules** format.

Output Format (;deps;logon)

The resources used in **needs** dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the **showjobs;logon** format. Schedules are listed in the basic **showschedules** format.

Examples

1. Display information about all resources on this cpu:

```
showresources
```

2. Display information about the dbase resource on cpu main in the **deps** format:

```
sr main#dbase;deps
```

showschedules

Display schedule status.

Security: Available to all users.

```
{showschedules } [schedselection][;keys      ] [;offline]
{ss             }           [;deps [;keys  ]
                               [;info  ]
                               [;logon ]
```

- schedselection* See [Selecting and Qualifying Schedules](#) on page 9-18. If omitted, the default is all schedules.
- keys** See [Output Format \(:keys\)](#) and [Output Format \(:deps;keys\)](#) below.
- deps** See [Output Format \(:deps\)](#) below.
- info** See [Output Format \(:deps;info\)](#) below.
- logon** See [Output Format \(:deps;logon\)](#) below.
- offline** Send the output of the command to the Conman output device. For more information, see [Offline Output](#) on page 9-3.

Output Format

- CPU** The cpu on which the schedule runs.
- Schedule** The name of the schedule.
- State** The state of the schedule. Schedule states are:
 - abend** Schedule terminated unsuccessfully.
 - add** Schedule has just been submitted.
 - exec** Schedule is executing.
 - hold** Awaiting dependency resolution.
 - ready** Dependencies resolved, ready to launch.
 - stuck** Execution interrupted. No jobs will be launched without operator intervention.
 - succ** Schedule completed successfully.
- Pr** The priority of the schedule.

(Est)Start	Schedule start time. If in parentheses, it is the estimated start time (that is, the at time). If the time is more than 24 hours in the past or future, the date is listed instead of the time.
(Est)Elapse	Schedule run time. If in parentheses, it is an estimation based on logged statistics.
Jobs #	The number of jobs in the schedule.
Jobs OK	The number of jobs that have completed successfully.
Sch Lim	The schedule's job limit. If blank, no limit is in effect.
<i>dependencies</i>	<p>A list of schedule dependencies and comments. Any combination of the following may be listed:</p> <ul style="list-style-type: none"> • For a follows dependency, a schedule or job name. • For an opens dependency, the file name. If the file resides on an x-agent and its name is longer than 28 characters, it is displayed as follows: <i>...last_25_characters_of_filename</i> • For a needs dependency, a resource name enclosed in hyphens (-). 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 a less than sign (<). • For a prompt dependency, the prompt number displayed as: #num. For global prompts, this is followed by the prompt name in parentheses. • Cancelled schedules are labeled: [Cancelled]. • Schedules cancelled with ;pend option are labeled: [Cancel Pend]. • Schedules whose until times have expired and schedules cancelled with ;pend option whose until times have expired are labeled: [Until]. • Schedules that are flagged as carryforward are labeled: [Carry]. • For schedules that were carried forward from the previous day, the original schedule name and original schedule date are displayed in brackets.

Output Format (;keys)

The schedules are listed one per line in the following format:

```
cpu#sched
```

Output Format (;deps)

The schedules used in `follows` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the basic `showjobs` format. Schedules are listed in the basic `showschedules` format.

Output Format (;deps;keys)

Schedules that have `follows` dependencies are listed one per line in the following format:

```
cpu#sched
```

Output Format (;deps;info)

The schedules used in `follows` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the `showjobs;info` format. Schedules are listed in the basic `showschedules` format.

Output Format (;deps;logon)

The schedules used in `follows` dependencies are listed followed by the dependent jobs and schedules. Jobs are listed in the `showjobs;logon` format. Schedules are listed in the basic `showschedules` format.

Examples

1. Display the status of all schedules in the `hold` state on this cpu:

```
showschedules @+state=hold
```

2. Display the status of all corp schedules on cpu site2 in the `deps;info` format:

```
ss site2#corp@;deps;info
```

3. Display offline the status of all schedules in the `abend` state on all cpus:

```
ss @#@+state=abend;off
```

shutdown

Unconditionally stop all of Maestro's production processes. On a stand-alone cpu, Batchman and Jobman are stopped. On network cpus, Netman, Mailman, all Mailman servers, and all `writers` are also stopped.

Security: You must have `shutdown` access to the cpu.

```
shutdown [;wait]
```

`wait` Wait until all processes have stopped before prompting for another command.

Usage

`shutdown` will stop only locally executing processes. To restart Netman only, execute the `startup` command (see [10-32](#)). To restart the entire process tree, execute a `start` command (described on [9-96](#)).

Note: If you are running Maestro in a network configuration, do an `unlink @; noask` command (described on [9-116](#)) before you issue a `shutdown`.

Examples

1. Shut down production on this cpu:
`shutdown`
2. Shut down production on this cpu and wait for all processes to stop:
`shut ;wait`

start

Start Maestro's production processes. On a UNIX cpu, **start** must be executed locally by the **maestro** user to start Netman.

Security: You must have **start** access to the cpu.

```
start [[domain!]cpu][;mgr][;noask]
```

domain The name of the domain in which production is started. The default is the domain in which the **conman** is running.

cpu The name of the cpu on which production is started. The default is the cpu on which the **conman** is running.

mgr Applies only to the local cpu on which **conman** is running. Start as the domain manager. The local cpu becomes the new domain manager and the current domain manager becomes a fault-tolerant agent.

This form of the command logically follows a **stop** command. Note that the preferred method of switching a domain manager is to execute a **switchmgr** command. See [switchmgr](#) on page 9-114 for more information.

noask Do not prompt for confirmation before starting each qualifying cpu.

Usage

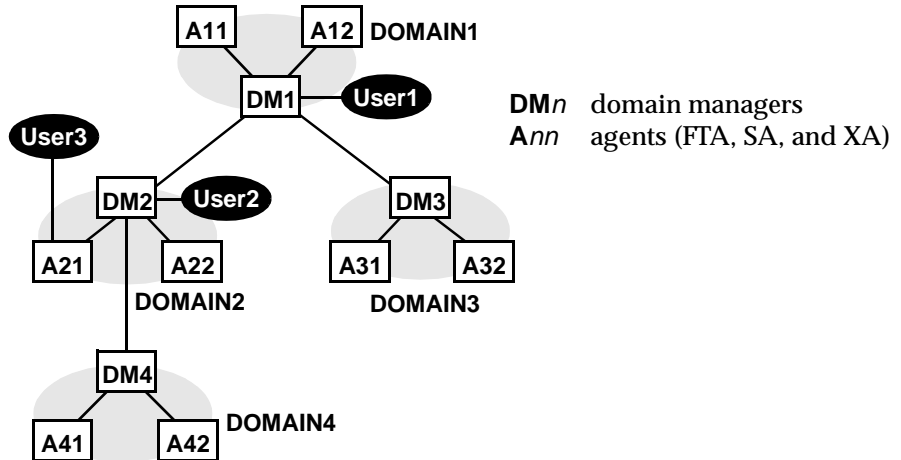
Following pre-production processing, the command should be executed on the master domain manager without specifying any cpus. This causes the **autolinked** fault-tolerant agent and standard agent cpus to be initialized and started automatically. Cpus that are not **autolinked** are initialized and started when you execute a **link** command following **start**.

Assuming that a user has **start** access to the cpus being started, the following rules apply:

- A user running the **conman** on the master domain manager can start any cpu in the network.
- A user running **conman** on a domain manager other than the master can start any cpu in its local domain or subordinate domain. The user cannot start cpus in a peer domain.
- A user running the **conman** on an agent can start any cpu in its local domain.

Examples

The table below shows the cpus started by different **start** commands executed by users at different locations in the depicted Maestro network.



Command	Cpus started by:		
	User1	User2	User3
<code>start @!@</code>	All cpus are started.	DM2 A21 A22 DM4 A41 A42	DM2 A21 A22
<code>start @</code>	DM1 A11 A12	DM2 A21 A22	DM2 A21 A22
<code>start DOMAIN3!@</code>	DM3 A31 A32	Not allowed.	Not allowed.
<code>start DOMAIN4!@</code>	DM4 A41 A42	DM4 A41 A42	Not allowed.
<code>start DM2</code>	DM2	DM2	DM2
<code>start A42</code>	A42	A42	Not allowed.
<code>start A31</code>	A31	Not allowed.	Not allowed.

status

Display Maestro's production status.

Security: Available to all users.

```
status
```

Symphony Mode

Following the word "Schedule" on the second line of the Conman banner, the Symphony mode is shown in parentheses. **Def** indicates that the Symphony file contains non-expanded object names mode, and **Exp** indicates expanded names mode. Symphony files are created with expanded or non-expanded names based on the Global option **expanded version**-- see [Global Options](#) on page 2-1.

Example

```
%status
```

```
MAESTRO for UNIX (HPUX)/CONMAN 6.0 (3.34) (C) Tivoli Systems 1998  
Schedule (Exp) 11/30/98 (#7) on DEMOCPU. Batchman down. Limit: 6, Fence: 0
```

```
%setsym 2
```

```
Schedule 11/29/98 (#5) on DEMOCPU. Symphony switched.
```

```
(2)%status
```

```
MAESTRO for UNIX (HPUX)/CONMAN 6.0 (3.34) (C) Tivoli Systems 1998  
Schedule (Exp) 11/29/98 (#5) on DEMOCPU. Symphony switched.
```

stop

Stop Maestro's production processes. To stop Netman, use the `shutdown` command.

Security: You must have `stop` access to the `cpu`.

```
stop [[domain!]cpu][;wait][;noask]
```

<i>domain</i>	The name of the domain in which production is stopped. The default is the domain in which conman is running.
<i>cpu</i>	The name of the <code>cpu</code> on which production is stopped. The default is the <code>cpu</code> on which conman is running.
<i>wait</i>	Do not accept another conman command until all production processes have stopped.
<i>noask</i>	Do not ask for confirmation before stopping each qualifying <code>cpu</code> .

Usage

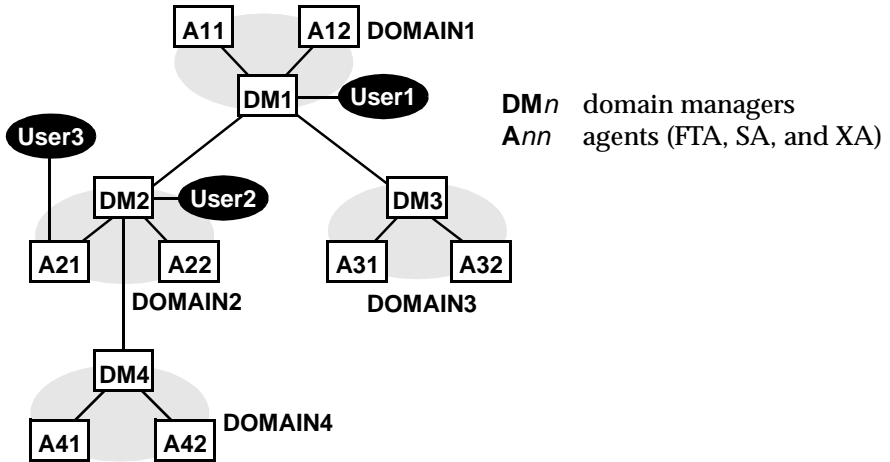
If `stop` cannot be delivered to a distant `cpu` (TCP/IP path not available, for example), the command is stored locally in a `pobox` file, and is mailed to the distant `cpu` when it becomes linked.

Assuming that a user has `start` access to the `cpus` being stopped, the following rules apply:

- A user running the **conman** on the master domain manager can stop any `cpu` in the network.
- A user running **conman** on a domain manager other than the master can stop any `cpu` in its local domain or subordinate domain. The user cannot stop `cpus` in a peer domain.
- A user running the **conman** on an agent can stop any `cpu` in its local domain.

Examples

On the following page, the table shows the `cpus` stopped by different `stop` commands executed by users at different locations in the depicted Maestro network.



Command	Cpus stopped by:		
	User1	User2	User3
stop @!@	All cpus 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.

submit docommand

Submit a command to be launched as a job by Maestro.

Security: You must have `submit` access to the command. To include `needs` and `prompt` dependencies, you must have `use` access to the resources and prompts.

```
{submit docommand= } [cpu#]"cmd" [ ; joboption [ ; ... ]
{sbd                } }
```

cpu The name of the cpu on which the job will be launched. Wildcards are permitted, in which case, the job is launched on all qualifying cpus. The default is your login cpu. You cannot submit a job to a cpu class.

cmd A valid system command (up to 255 characters). The entire command must be enclosed in quotes ("). The command is treated the same as a job, and all job rules apply.

joboption A combination of dependencies, recovery, and other job options. See [Job Dependencies](#), [Recovery Options](#) and [Other Options](#) below.

Job Dependencies

Following is a syntax summary of job dependencies. Wildcards are not permitted. For a complete description refer to [The Scheduling Language](#) on page 8-50.

```
at=hhmm[+n days |mm/dd/yy]
confirmed
every=rate
follows={ [cpu#]sched{.job|@} | job|net::net_dep} [ , ... ]
needs=[num] [cpu#]resource [ , ... ]
opens=[cpu#]"path" [ (qualifier) ] [ , ... ]
priority=pri | hi | go
prompt={"[:|!]text" | promptname} [ , ... ]
until=hhmm[+n days |mm/dd/yy]
```

Recovery Options

Following is a syntax summary of recovery options. For a complete description refer to [Jobs](#) on page 8-39.

```
recovery=stop|continue|rerun
after|recoveryjob=rcpu#rjob
abendprompt|recoveryprompt="rtext"
```

Other Options

Following is a syntax summary of other job options.

```
alias[=name]
interactive
into=[cpu#]sched
[stream]logon=user
```

<i>name</i>	A unique name to be assigned to the job. See Alias Names below.
interactive	Identifies the job as one that runs interactively on the users desktop.
<i>cpu</i>	The name of the cpu on which the into schedule is launched. If omitted, the default is your login cpu. Cpu classes are not valid.
<i>sched</i>	The name of the schedule into which the job will be placed for launching. If omitted, the job is added to a schedule named JOBS.
<i>user</i>	The user name under which the job will execute. The default is your user name.

Usage

If you omit the **cpu** in **follows**, **needs**, or **opens**, the default is the cpu of the job.

submit file

Submit a script file to be launched as a job by Maestro.

Security: You must have **submit** access to the script file. To include **needs** and **prompt** dependencies, you must have **use** access to the resources and prompts.

```
{submit file= }path[;joboption[;...]][;noask]
{sbff          }
```

path The path name of the script file (up to 255 characters). Wildcards are permitted. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumeric, dashes (-), slashes (/), and underscores (_).

joboption A combination of dependencies, recovery, and other job options. See [Job Dependencies](#), [Recovery Options](#) and [Other Options](#) below.

noask Do not prompt for confirmation before taking action against each qualifying file.

Job Dependencies

Following is a syntax summary of job dependencies. Wildcards are not permitted. For a complete description refer to [The Scheduling Language](#) on page 8-50.

```
at=hhmm[+n days |mm/dd/yy]
confirmed
every=rate
follows={ [cpu#]sched{.job|@} | job|net::net_dep}[ ,... ]
needs=[num] [cpu#]resource[ ,... ]
opens=[cpu#]"path"[(qualifier)][ ,... ]
priority=pri |hi |go
prompt={"[:|!]text" |promptname}[ ,... ]
until=hhmm[+n days |mm/dd/yy]
```

Recovery Options

Following is a syntax summary of recovery options. For a complete description refer to [Jobs](#) on page 8-39.

```
recovery=stop|continue|rerun
after|recoveryjob=rcpu#rjob
abendprompt|recoveryprompt="rtext"
```

Other Options

Following is a syntax summary of other job options.

```
alias[=name]
interactive
into=[cpu#]sched
[stream]logon=user
```

<i>name</i>	A unique name to be assigned to the job. See Alias Names below.
interactive	Identifies the job as one that runs interactively on the users desktop.
<i>cpu</i>	The name of the cpu on which the into schedule is launched. If omitted, the default is your login cpu. Cpu classes are not valid.
<i>sched</i>	The name of the schedule into which the job will be placed for launching. If omitted, the job is added to a schedule named JOBS.
<i>user</i>	The user name under which the job will execute. The default is your user name.

Usage

If you omit the **cpu** in **follows**, **needs**, or **opens**, the default is your login cpu. If executed on a fault-tolerant agent, and you wish to submit a job with a **prompt** dependency or an **abendprompt**, the **m Mozart** directory

(*maestrohome/mozart*) on the master domain manager must be accessible via a mount or share.

Alias Names

The `alias` keyword is used to assign a unique name to the job. The name can contain up to eight alphanumeric characters starting with a letter. The name is always upshifted. If you enter the `alias` keyword without a name, a job name is constructed using the first two alphanumeric characters of the filename (basename) followed by a six digit random number. The name is always upshifted. If the filename does not begin with a letter, you are prompted to use the `alias` option to supply a valid name. Some examples follow.

If filename is:	Job name is (the last six digits are random):
<code>jc1ttx5</code>	<code>JC342454</code>
<code>B56yu9</code>	<code>B5515241</code>
<code>j-123</code>	<code>J543235</code>
<code>9-jc1</code>	invalid, must use <code>alias=name</code>

If you omit the `alias` option entirely, a job name is constructed using the first eight alphanumeric characters of the filename (basename). The name is always upshifted. If the filename does not begin with a letter, you are prompted to use the `alias` option to supply a valid name. Some examples:

If filename is:	Job name is:
<code>jc1ttx5</code>	<code>JCLTTX5</code>
<code>B56yu9</code>	<code>56YU9</code>
<code>j-123</code>	<code>J</code>
<code>9-jc1</code>	invalid, must use <code>alias=name</code>

Examples

1. Submit a script file into the JOBS schedule. The job name is myjcl.

```
submit file=/usr/lib/john/myjcl
```

2. Submit a script file, with a job name of misjob4, into a schedule named missed. The job needs two units of the slots resource.

```
sbf /usr/lib/mis/misjcl/jcl4;alias=misjob4;into=missked;needs=2 slots
```

3. Submit all script files that have file names beginning with "back" into a schedule named bkup.

```
sbf "/usr/lib/backup/back@";into=bkup
```

submit job

Submit a job to be launched by Maestro. To submit a job, you must be running Console Manager on the master domain manager, or have access to the Maestro databases on the master domain manager.

Security: You must have `submit` access to the job. To include `needs` and `prompt` dependencies, you must have `use` access to the resources and prompts.

```
{submit job=} [cpu#] job[; joboption[;...]] [noask]
{sbj          }
```

- cpu* The name of the cpu on which the job will be launched. Wildcards are permitted, in which case, the job is launched on all qualifying cpus. The default is your login cpu. You cannot submit a job to a cpu class.
- job* The name of the job. Wildcards are permitted, in which case, all qualifying jobs are submitted. If the job is already in production, and is being submitted into the same schedule, you must use the `alias` option to assign a unique name.
- joboption* A combination of dependencies, recovery, and other job options. See [Job Dependencies](#), [Recovery Options](#) and [Other Options](#) below.
- noask* Do not prompt for confirmation before taking action against each qualifying job.

Job Dependencies

Following is a syntax summary of job dependencies. Wildcards are not permitted. For a complete description refer to [The Scheduling Language](#) on page 8-50.

```
at=hhmm[+n days | mm/dd/yy]
confirmed
every=rate
follows={ [cpu#] sched{. job|@} | job | net::net_dep } [ , ... ]
needs=[num] [cpu#] resource [ , ... ]
opens=[cpu#] "path" [ (qualifier) ] [ , ... ]
priority=pri | hi | go
prompt="{ [:|!] text" | promptname } [ , ... ]
until=hhmm[+n days | mm/dd/yy]
```

Recovery Options

Following is a syntax summary of recovery options. For a complete description refer to [Jobs](#) on page 8-39.

```
recovery=stop | continue | rerun
after | recoveryjob=rcpu#rjob
abendprompt | recoveryprompt="rtext"
```

Other Options

Following is a syntax summary of other job options.

```
alias[=name]
into=[cpu#] sched
```

name A unique name to be assigned to the job. See [Alias Names](#) below.

<i>cpu</i>	The name of the <code>cpu</code> on which the <code>into</code> schedule is launched. If omitted, the default is the <code>cpu</code> of the job. <code>Cpu</code> classes are not valid.
<i>sched</i>	The name of the schedule into which the job will be placed for launching. If omitted, the job is added to a schedule named <code>JOBS</code> .

Usage

If you omit the `cpu` in `follows`, `needs`, or `opens`, the default is the `cpu` of the job. If executed on a fault-tolerant agent, and you wish to submit a job with a `prompt` dependency or an `abendprompt`, the `mozart` directory (`maestrohome/mozart`) on the master domain manager must be accessible via a mount or share.

Alias Names

The `alias` keyword is used to assign a unique name to the job. The name can contain up to eight alphanumeric characters starting with a letter. The name is always upshifted. If you enter the `alias` keyword without a name, a job name is constructed using the first two alphanumeric characters of the submitted job name followed by a six digit random number. The name is always upshifted. Some examples:

If submitted job name is:	Resulting job name is (the last six digits are random):
<code>job55</code>	<code>JO342454</code>
<code>apjob3</code>	<code>AP515241</code>

Examples

1. Submit the test jobs into the JOBS schedule.

```
submit job=test@
```

2. Submit a job, with an alias of rptx4. Place the job in the reports schedule with an at time of 5:30 p.m.

```
sbj rjob4;alias=rptx4;into=reports;at=1730
```

3. Submit job txjob3 on all cpus whose names begin with "site":

```
sbj site#@txjob3;alias
```

If the qualifying cpus are site1 and site2, the jobs are added to their respective JOBS schedules as follows:

```
SITE1#JOBS.TX122231
```

```
SITE2#JOBS.TX345623
```


submit sched

Submit a schedule to be launched by Maestro. To submit a schedule, you must be running Console Manager on the master domain manager, or have access to the Maestro databases on the master domain manager.

Security: You must have `submit` access to the schedule. To include `needs` and `prompt` dependencies, you must have `use` access to the resources and prompts.

```
{submit sched= }[cpu#]sched[;schedoption[;...]][;noask]
{sbs          }
```

<i>cpu</i>	The name of the <code>cpu</code> or <code>cpu</code> class on which the schedule will be launched. Wildcards are permitted, in which case, the schedule is launched on all qualifying <code>cpus</code> . The default is your login <code>cpu</code> .
<i>sched</i>	The name of the schedule. If this schedule is already in production, you must use the <code>alias</code> option to assign a unique name. Wildcards are permitted.
<i>schedoption</i>	A combination of dependencies, and other schedule options. See Schedule Dependencies , and Other Options below.
noask	Do not prompt for confirmation before taking action against each qualifying schedule.

Schedule Dependencies

Following is a syntax summary of schedule dependencies. For a complete description refer to *T [The Scheduling Language](#)* on page 8-50.

```
at=hhmm[+n days |mm/dd/yy]
carryforward
follows={ [cpu#]sched[. job|@]] | job|net::net_dep}[,...]
limit=joblimit
needs=[num] [cpu#]resource[,...]
opens=[cpu#]"path"[(qualifier)][,...]
priority=pri|hi|go
prompt={"[:|!]text" |promptname}[,...]
until=hhmm[+n days |mm/dd/yy]
```

Other Options

Following is a syntax summary of other schedule options.

```
alias[=name]
```

name A unique name to be assigned to the schedule. See [Alias Names](#) below.

Usage

If you omit the `cpu` in `follows`, `needs`, or `opens`, the default is the `cpu` of the schedule. If executed on a fault-tolerant agent, and you wish to submit a job with a `prompt` dependency or an `abendprompt`, the `mozart` directory (`maestrohome/mozart`) on the master domain manager must be accessible via a mount or share.

Alias Names

The `alias` keyword is used to assign a unique name to the schedule. The name can contain up to eight alphanumeric characters starting with a letter. The name is always upshifted. If you enter the `alias` keyword without a

name, a schedule name is constructed using the first two alphanumeric characters of the submitted schedule name followed by a six digit random number. The name is always upshifted. Some examples:

If submitted schedule name is:	Resulting name is (the last six digits are random):
sked99	SK342454
arsked3	AR515241

Examples

1. Submit the adhoc schedule on cpu site1, and flag it as a `carryforward` schedule.

```
submit sched=site1#adhoc;carryforward
```
2. Submit schedule fox4 with a job limit of 2, a priority of 23, and an `until` time of midnight.

```
sbs fox4;limit=2;pri=23;until=0000
```
3. Submit schedule sched3 on all qualifying cpus.

```
sbs site@#sched3
```

Assuming cpus site1 and site2 qualify, sched3 is submitted on both cpus as follows:

```
SITE1#SCHED3  
SITE2#SCHED3
```

switchmgr

The **switchmgr** command is used to switch domain management from the current domain manager to a backup domain manager. The backup domain manager must be running Maestro version 6.0 or later.

Security: The user issuing **switchmgr** must have **start** and **stop** access to the cpu being designated as the new domain manager.

```
switchmgr domain,newdm
```

domain The domain in which the manager will be switched.

newdm The Maestro cpu name of the new domain manager. This should be defined beforehand as a backup domain manager (a fault-tolerant agent with Resolve Dependencies and Full Status enabled). It must be in the same domain as the current domain manager.

Operation

The command stops the specified cpu, and then restarts it as the current domain manager. All domain members are informed of the new domain manager, and the old domain manager converts to a fault-tolerant agent.

The identification of domain managers is carried forward to each new day's Symphony file, so that any switch remains in effect until a subsequent **switchmgr** command is executed. However, if **Jnextday** processing is performed on the old domain manager, it will take back domain management responsibilities.

Examples

1. Switch the domain manager to orca in the masterdm domain:

```
switchmgr masterdm,orca
```

2. Switch the domain manager to ruby in the bldg2 domain:

```
switchmgr bldg2,ruby
```

tellop

Send a message to the Maestro console.

Security: Available to all users.

```
{tellop } [text]
{to      }
```

text The character string to be sent.

Usage

The message is sent only if the console message level is greater than zero. See [console](#) on page 9-40.

In a network, the message is sent to the Maestro console on each cpu to which your login cpu is currently linked.

If the command is entered without options, Conman prompts for the message text. Following the prompt "TELLOP>", type in each line and press Return. To signify the end of your message, type two slashes (//) or a period (.), and press Return. Typing Control-c at any time will abort the command without sending the message.

The overall length of a message cannot exceed 900 characters. Due to the addition of a label before each line, longer lines may be split when displayed. You can use the new line sequence (\n) to format messages.

Examples

1. Send command line text:

```
tellop Maestro will be stopped at\n4:30 for 15 minutes.
```

2. Enter text interactively:

```
to
TELLOP>*****
TELLOP>*  Maestro will be stopped at   *
TELLOP>*  4:30 for 15 minutes.       *
TELLOP>*****
TELLOP>// <Return>
```

unlink

Close Maestro links to disable inter-cpu communications.

Security: You must have `unlink` access to the destination cpu.

```
unlink [domain!]cpu[;noask]
```

<i>domain</i>	The name of the domain in which links are closed. Wildcards are permitted. The default is the domain of the target cpu. If the cpu is wildcarded, the default is the domain in which conman is running.
<i>cpu</i>	The name of the target cpu-- any cpu except an extended agent. Wildcards are permitted.
noask	Do not prompt for confirmation before unlinking from each qualifying cpu.

Usage

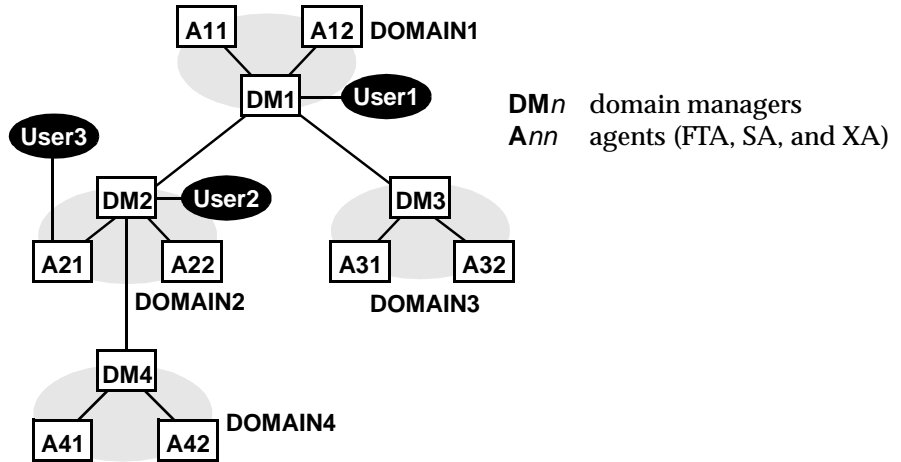
Assuming that a user has `unlink` access to the cpus being unlinked, the following rules apply:

- A user running the Console Manager on the master domain manager can unlink any cpu in the network.
- A user running the Console Manager on a domain manager other than the master can unlink any cpu in its local domain or subordinate domain. The user cannot unlink cpus in a peer domain.
- A user running the Console Manager on an agent can unlink any cpu in its local domain.

`unlink` remains in effect until a `link` command is executed to re-open the link.

Examples

On the following page, the table shows the links closed by different `unlink` commands executed by users at different locations in the depicted Maestro network.



Command	Links closed by:		
	User1	User2	User3
<code>unlink !@</code>	All links are closed.	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4 DM4-A41 DM4-A42	DM2-A21 DM2-A22
<code>unlink @</code>	DM1-A11 DM1-A12 DM1-DM2 DM1-DM3	DM1-DM2 DM2-A21 DM2-A22 DM2-DM4	DM2-A21 DM2-A22
<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	na	DM2-A21
<code>unlink A42</code>	DM4-A42	DM4-A42	Not allowed.
<code>unlink A31</code>	DM3-A31	Not allowed.	Not allowed.

version

Display Conman's program banner.

```
version
```

Example

```
%version  
MAESTRO for UNIX (HPUX)/CONMAN 6.0 (3.34) (C) Tivoli Systems 1998  
Schedule 5/16/98 (#7) on DEMOCPU. Batchman down. Limit: 6, Fence: 0
```


10

Utility Commands

This section describes Maestro's utility commands. The commands, except `startUp` and `version`, are installed in the `bin` subdirectory of the `maestro` user's home directory (`maestrohome/bin`). `startUp` is installed in the `maestrohome` directory, and `version` is installed in the `maestrohome/version` directory.

Command Descriptions

Command	Description	Page
<code>at mat</code>	Submit a job to be executed at a specific time.	10-2
<code>batch mbatch</code>	Submit a job to be executed as soon as possible.	10-2
<code>cpuinfo</code>	Return information from a <code>cpu</code> definition.	10-6
<code>datecalc</code>	Convert date and time to a desired format	10-8
<code>dbexpand</code>	Expand Maestro's databases.	10-13
<code>delete</code>	Remove script files and standard list files by name.	10-14
<code>evtsize</code>	Define the maximum size of Maestro's event files.	10-15
<code>jobinfo</code>	Return information about the current job.	10-16
<code>jobstdl</code>	Return the pathnames of standard list files.	10-18
<code>maestro</code>	Return the home directory of the <code>maestro</code> user.	10-20
<code>makecal</code>	Create custom calendars.	10-21
<code>morestdl</code>	Display the contents of standard list files.	10-23
<code>parms</code>	Display, change, and add parameters.	10-25
<code>release</code>	Release units of a resource.	10-27
<code>rmstdlist</code>	Remove standard list files based on age.	10-29
<code>showexec</code>	Display information about executing jobs.	10-30
<code>startUp</code>	Start the Netman process.	10-32
<code>version</code>	Display Maestro file information.	10-33

at and batch

The `at` and `batch` commands are used to submit ad hoc commands and jobs that are launched by Maestro. Maestro's `customize` script creates the following links by default:

```
/usr/bin/mat -> maestrohome/bin/at
/usr/bin/mbatch -> maestrohome/bin/batch
```

Security: See [at.allow and at.deny](#) below.

```
{at } {-v|-u| [-s sched ] time-spec}
{mat }          [-qqueue ]

{batch } [-v|-u|-s sched]
{mbatch}
```

- `-v|-v` Display the command version and exit.
- `-u|-u` Display command usage information and exit.
- `-s sched` The name of a schedule into which the job will be submitted. The name can contain up to eight alphanumeric characters starting with a letter, and may contain non-leading dashes (-).

If this option and the `qqueue` option are omitted, a schedule name is selected based on the value of the environment variable `ATSCRIPT`. If `ATSCRIPT` contains the word "maestro", the schedule name 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 schedule name will be "at" (for jobs submitted with `at`), or "batch" (for jobs submitted with `batch`).

See [Other Considerations](#) below for more information about schedules.
- `-qqueue` The job is submitted into a schedule with the name `queue`, which can be a single letter (a-z). See [Other Considerations](#) below for more information about schedules.
- `time-spec` For `at` jobs only. The time at which the job will be launched by Maestro. The syntax is the same as that used with the UNIX `at` command.

Usage

After entering `at` or `batch`, enter the commands that constitute the job. Each command line is ended with 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 the redirection symbol "<" to cause `at` or `batch` to read commands from a file. See [Examples](#) below.

Information about `at` and `batch` jobs is sent to the master domain manager, where the jobs are added to schedules in the `symphony` file. The jobs are launched based on the dependencies included in the schedules.

The shell used for jobs submitted with Maestro's `at` and `batch` commands is determined by the `SHELL_TYPE` variable in the `jobmanrc` configuration script. (Note: Do not use the C shell.) See [Standard Configuration Script - jobmanrc](#) on page 3-4 for more information.

Once submitted, jobs are launched in the same manner as scheduled jobs. Each job executes in the submitting user's environment. To ensure that the environment is complete, set variable commands are inserted into the script for the variables set in the user's environment.

Replacing the UNIX Commands

If you wish, the standard UNIX `at` and `batch` commands can be replaced with the Maestro commands. This can be done as follows:

```
$ mv /usr/bin/at /usr/bin/uat
$ mv /usr/bin/batch /usr/bin/ubatch
$ ln -s maestrohme/bin/at /usr/bin/at
$ ln -s maestrohme/bin/batch /usr/bin/batch
```

Note that Maestro's `customize` script installs links to its `at` and `batch` commands by default. This permits the commands to be run as: `/usr/bin/mat` and `/usr/bin/mbatch`.

at.allow and at.deny

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 the use of `at` and `batch`. If neither of these files exists, only the `root` user is permitted to use `at` and `batch`. If the commands are executed as `mat` and `mbatch`, `at.allow` and `at.deny` are ignored, and no restrictions apply.

Script Files (jcl)

The commands entered with `at` or `batch` are stored in a script file. The file names are created as follows:

```
maestrohome/atjobs/epoch.sss
```

Where: *epoch* The number of seconds since 00:00, 1/1/70.
 sss The first three characters of the schedule name.

Note: Maestro removes script files for jobs that are not carried forward. However, it is a good practice to monitor the disc space in the `atjobs` directory and remove older files if necessary.

Job Names

All `at` and `batch` jobs are given unique Maestro job names 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. For example:

```
User's name=           fredrick  
User's pid=            5678  
Maestro job name=     FRED5678
```

Other Considerations

Tivoli recommends that the schedules into which `at` and `batch` jobs are submitted be created beforehand using Composer. The schedules can contain dependencies that determine when the jobs will be launched. At a minimum, the schedules should contain the `carryforward` keyword. This will ensure that jobs that do not complete, or are not launched, during the current day will be carried forward to the next day's production schedule. Some other suggestions regarding these schedules:

1. Include the expression "on `everyday`" to have the schedules selected everyday.
2. Use the `limit` keyword to limit the number of submitted jobs that can be run concurrently.
3. Use the `priority` keyword to set the priority of submitted jobs relative to other jobs.

Examples

1. Submit a job into schedule `sched8` to be launched as soon as possible:

```
mbatch -s sched8  
shell-command-line <Return>  
...  
<Control d>
```

2. Submit a job into schedule `k` to be launched at 9:45 pm:

```
mat -qk 21h45  
shell-command-line <Return>  
...  
<Control d>
```

3. Submit a job to be launched two hours from now:

```
maestrohome/bin/at now + 2 hours  
shell-command-line <Return>  
...  
<Control d>
```

If the variable `ATSCRIPT` contains " ", the job is submitted into a schedule having the same name of the user's group. Otherwise, it is submitted into a schedule named "at".

4. Submit a job into schedule `sked-mis` to be launched at 5:30 pm:

```
mat -s sked-mis 17h30  
shell-command-line <Return>  
...  
<Control d>
```

5. Same as example 4 above, except that the job's commands are read from a file:

```
mat -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 `./myjob` into a script file created by the `at` command.

cpuinfo

This utility command can be used within an access method to return information from a cpu definition.

```
cpuinfo -v|-U|cpu_name [cpu_option] [...]
```

- v|-v** Display the command version and exit.
- U** Display command usage information and exit.
- cpu_name* The name of the cpu.
- cpu_option* One or more of the following:
 - os_type** The value of the **os** field: **MPEV**, **MPIX**, **UNIX**, **WNT**, or **OTHER**.
 - node** The value of the **node** field.
 - port** The value of the **port** field.
 - autolink** The value of the **autolink** field: **ON** or **OFF**.
 - fullstatus** The value of the **fullstatus** field: **ON** or **OFF**.
 - resolvedep** The the value of the **resolvedep** field: **ON** or **OFF**.
 - host** The value of the **host** field.
 - method** The value of the **access** field.
 - server** The value of the **server** field.
 - type** The type of cpu: **MASTER**, **MANAGER**, **FTA**, **S-AGENT**, and **X-AGENT**.
 - time_zone** The time zone of the cpu. For an x-agent, the field is blank.
 - version** The Maestro version the cpu is running. For an x-agent, the field is blank.
 - info** The operating system version and cpu model of the cpu. For an x-agent, no information is printed.

Usage

The values are returned, separated by new lines, in the same order that the cpu options were entered on the command line. If no options are specified, all applicable cpu information is returned with labels, separated by new lines.

Examples

The examples are based on the following cpu definition:

```

cpuname    oak
os         UNIX
node       oak.tivoli.com
tcpaddr    51111
    for maestro
        host        maple
        access      nt
        autolink    off
        fullstatus  off
    end

```

1. Print the os type for cpu oak:

```

>cpuinfo oak os_type
UNIX

```

2. Print the node, access method, and host, in that order, for cpu oak:

```

>cpuinfo oak node host
oak.tivoli.com
maple

```

3. Print all information for cpu oak:

```

>cpuinfo oak
OS TYPE:UNIX
NODE:oak.tivoli.com
PORT:31111
AUTOLINK:ON
FULLSTATUS:OFF
RESOLVEDEP:OFF
HOST:maple
METHOD:
SERVER:
TYPE: S-AGENT
TIME_ZONE:US/Pacif
VERSION:4.5
INFO:SunOS 5.3 Generic 1016 sun4m

```

datecalc

The `datecalc` command resolves a date expression and returns the date in a desired format. Maestro's `customize` script creates the following link by default:

```
/usr/bin/datecalc -> maestrohome/bin/datecalc
```

```
datecalc -v|-u
datecalc {base-date          } [offset] [pic format]
         {-t time [base-date]}
         {yyyymmddhhtt      }
```

- `-v|-V` Display the command version and exit.
 - `-u|-U` Display command usage information and exit.
 - `base-date` One of: `day|date|today|tomorrow|scheddate`
- Where:
- `day` Next occurrence of a day, expressed as:
 su mo tu we th fr sa
 - `date` The next occurrence of a date, expressed as:
 element/element[/element]
- Where *element* is:
- `d[d]` day number
 - `m[m]` month number
 - `yy[yy]` year number. If two digits are used, a number greater than 70 is a 20th century date, and a number less than 70 is a 21st century date.
- The month number (`m[m]`) can be expressed as:
- jan may sep
 - feb jun oct
 - mar jul nov
 - apr aug dec

The delimiting slashes (/) can be replaced by dashes (-), periods (.), commas (,), or spaces.

For example, any of the following can be entered for March 28, 1997:

03/28/97	3-28-1997	28.mar.97
97,28,3	mar 28 1997	93 3 97

If numbers are used, it is possible to enter an ambiguous date- for example, 2,7,98. In this case, **datecalc** uses the date format defined in Maestro's message catalog to interpret the date. If the date does not match the format, **datecalc** outputs an error message. For example, if the message catalog defines the format as "mm/dd/YY", the following will occur:

2,7,98	accepted by datecalc
98,7,2	error message

today	The current system date.
tomorrow	The current system date plus one day (or plus 24 hours in the case of time calculations).
scheddate	Maestro's current production date.

-t *time*

The time expressed as:

```
[now ]
[noon ]
[midnight ]
[h[h][[:]tt] [am|pm][zulu] ]
```

Where:

now	The current system date and time.
noon	12:00 p.m. (or 1200).
midnight	12:00 a.m. (or 0000).

`h[h][[:]tt]` Hour and minute, expressed 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 (lowercase), 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` The time you entered is Greenwich Mean Time (Universal Coordinated Time). `datecalc` will convert it to local time.

If a *base-date* is not specified, the default is `today` if the time is greater than the current system time, or `tomorrow` if the time is less than the current system time.

`yyyymmddhhtt`

The year, month, day, hour, and minute expressed in exactly twelve digits. For example: 199705070915 (1997, May 7, 9:15 a.m.).

`offset`

An offset from *base-date*, expressed as:

```
{ {+|>|-|<} {units|nearest } } {day[s] }
{next } {weekday[s] }
{workday[s] }
{week[s] }
{month[s] }
{year[s] }
{hour[s] }
{minute[s] }
{day }
{calendar }
```

Where:

- `next` The next occurrence of the unit type.
- `+|>` (Plus) Offset to a later date or time. Be sure to escape the greater than sign (i.e., "`\>`").
- `-|<` (Minus) Offset to an earlier date or time. Be sure to escape the less than sign (i.e., "`\<`").

<i>units</i>	The number of units of the specified type.
nearest	Offset to the nearest occurrence of the unit type (earlier or later).
weekday[s]	Every day but Saturday and Sunday.
workday[s]	Same as weekday[s] , excluding any dates on the holidays calendar.
<i>day</i>	Same as <i>day</i> in <i>base-date</i> above.
<i>calendar</i>	Entries in a calendar by this name. For example: <code>datecalc today next monthend</code> will return the next date, from today, on the calendar <code>monthend</code> .

pic The format in which the date and time are returned. The *format* characters are:

- m** Month number
- d** Day number
- y** Year number
- j** Julian day number
- h** Hour number
- t** Minute number
- ^** One space. This character must be escaped (\) in the Bourne shell.

Punctuation characters can also be included. These are the same as the delimiters used in *date* and *time* above.

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.

Examples

- In the following examples, the current system date is Friday, April, 1997.

```
>datecalc today +2 days pic mm/dd/yy
04/13/97
>datecalc today next tu pic yy\^mm\^dd
97 04 15
```

```
>LANG=american;export LANG
>datecalc -t 14:30 tomorrow
Sat, Apr 12, 1997 02:30:00 PM
>LANG=french;datecalc -t 14:30 tomorrow
Samedi 12 avril 1997 14:30:00
```

- 2.** In the following example, the current system time is 10:24.

```
>datecalc -t now \> 4 hours pic hh:tt
14:24
```

dbexpand

Convert the databases on the master domain manager from non-expanded to expanded versions. The command sets the `expanded version` global option to `yes`, and makes backup copies of your old database files that can be used to return to the pre-expanded versions if necessary

If you update your network from an earlier version to Maestro 6.0 in stages, you must use non-expanded databases until all of your computers have been updated. When all of the computers are running Maestro 6.0, run **dbexpand** on the master domain manager to convert the databases to expanded versions.

```
dbexpand [-v|-U|-n|-b backup_dir]
```

- `-v` Display the version and exit.
- `-U` Display command usage information and exit.
- `-n` Run non-interactive. Do not prompt for a backup directory name.
If `-b` is included, the named directory is used for backup. If `-b` is not included, the default directory is used. In either case, if the directory exists, it is overwritten.
- `-b backup_dir` Use this directory to backup the database files. If omitted, the default directory is: `maestrohome/mozart/mozart.old`. If `-n` is omitted, and the backup directory already exists, you are prompted for a backup directory name.

Usage

Dbexpand can be run without stopping Maestro. However, Console Manager (conman) **submit jobs** and **submit schedule** commands cannot be executed until after the new day turnover (**Jnextday**). For this reason, it is recommended that **dbexpand** be run shortly before the **Jnextday** job.

Example

Expand the databases, and backup the current files in the `/usr/lib/maestro/temp` directory. If the directory already exists, overwrite its contents.

```
dbexpand -n -b /usr/lib/maestro/temp
```

delete

Remove files. The command is intended to remove script files (created by Maestro's `at` and `batch`) and standard list files.

Security: The `maestro` and `root` users can remove any file. Other users can only remove files associated with their own jobs.

```
delete -v|-u|path [...]
```

`-v|-v` Display the command version and exit.

`-u|-u` Display command usage information and exit.

`path` A script or standard list file pathname. The name must be enclosed in quotes (") if it contains characters other than the following: alphanumeric, dashes (-), slashes (/), and underscores (_). Wildcards are permitted.

Warning: Use this command carefully. Improper use of wildcards can result in removing files accidentally.

Example

The following is the script for a Maestro scheduled job that removes its 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
...
```

Note that the standard configuration script, `jobmanrc`, sets the variable `UNISON_STDLIST` to the pathname of the job's standard list file. For more information about `jobmanrc` refer to [Job Execution](#) on page 3-1.

evtsize

Define the size of Maestro's event files. This command is used to increase the size of an event file after receiving the Maestro message, "End of file on events file."

Security: Must be the `maestro` or `root` user.

```
evtsize -v|-u|filename filesize
```

`-v|-v` Display the command version and exit.

`-u|-u` Display command usage information and exit.

filename The name of the event file. One of the following:

```
Courier.msg  
Intercom.msg  
Mailbox.msg  
maestro.msg  
netman.msg  
unison.msg
```

filesize The maximum size of the event file in number of bytes. When first built by Maestro, the maximum size is set to 1 Mbyte.

Example

Set the maximum size of the Intercom file to 2 million bytes:

```
evtsize Intercom.msg 2000000
```

jobinfo

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

```
jobinfo -v|-u|job-option [...]
```

<code>-v -v</code>	Display the command version and exit.
<code>-u -u</code>	Display command usage information and exit.
<i>job-option</i>	One or more of the following:
<code>confirm_job</code>	Returns "YES" if the job requires confirmation.
<code>is_command</code>	Returns "YES" if the job was scheduled or submitted using the <code>docommand</code> construct.
<code>job_name</code>	Returns the job's name without the <code>cpu</code> and <code>schedule</code> names.
<code>job_pri</code>	Returns the job's priority level.
<code>programmatic_job</code>	Returns "YES" if the job was submitted with Maestro's <code>at</code> or <code>batch</code> command.
<code>re_job</code>	Returns "YES" if the job is being rerun as the result of a <code>conman rerun</code> command, or the <code>rerun</code> recovery option.
<code>re_type</code>	Returns the job's recovery option (<code>stop</code> , <code>continue</code> , or <code>rerun</code>).
<code>rstrt_flag</code>	Returns "YES" if the job is being run as the recovery job.
<code>time_started</code>	Returns the time the job started executing.

Usage

Job option values are returned, separated by new lines, in the same order they were requested.

Example

The script file `/jcl/backup` is documented twice, giving it the Maestro 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 determination:

```
#Determine partial (1) or full (2):
if [ "`maestro`/bin/jobinfo job_name`" = "PARTBACK" ]
then
    bkup=1
else
    bkup=2
fi
```

jobstdl

Return the pathnames of standard list files.

```
jobstdl [-v|-u]
jobstdl [-day days][-first ] [-name sched.job]
                [-last ] [job-number ]
                [-num n ]
                [-all ]
```

<code>-v -V</code>	Display the command version and exit.
<code>-u -U</code>	Display command usage information and exit.
<code>-day</code>	Return the pathnames of standard list files that are a specific number of days old (1 for yesterday, 2 for the day before yesterday, etc.). The default is zero (today).
<code>-first</code>	Return the pathname of the first qualifying standard list file.
<code>-last</code>	Return the pathname of the last qualifying standard list file.
<code>-num</code>	Return the pathname of the standard list file for a specific run of a job.
<code>-all</code>	Return the pathname of all qualifying standard list files.
<code>-name</code>	The name of the schedule and job for which standard list file pathnames are returned.
<i>job-number</i>	The job number for which standard list file pathnames are returned.

Output Format

For a description of standard list files and their directories refer to [Standard List Files](#) on page 3-7.

Pathnames are returned in a format suitable for input to other commands. Multiple pathnames are returned separated by one space.

Examples

1. Return the pathnames of all standard list files for today:

```
jobstdl
```

2. Return the standard list file pathname for the first run of job mailxhg1.getmail on the current day:
`jobstdl -first -name mailxhg1.getmail`
3. Return the standard list file pathname for the second run of job mailxhg1.getmail on the current day:
`jobstdl -num 2 -name mailxhg1.getmail`
4. Return the standard list file pathnames for all runs of job mailxhg1.getmail from three days ago:
`jobstdl -day 3 -name mailxhg1.getmail`
5. Return the standard list file pathname for the last run of job mailxhg1.getmail from four days ago:
`jobstdl -day 4 -last -name mailxhg1.getmail`
6. Return the pathname of the standard list file for job number 455:
`jobstdl 455`
7. Print the contents of the standard list file for job number 455:
`cd `maestro`/bin
lp -p 6 `jobstdl 455``

maestro

Return the pathname of the Maestro user's home directory, that is, *maestrohome*. Maestro's `customize` script creates the following link by default:

```
/usr/bin/maestro -> maestrohome/bin/maestro
```

```
maestro [-v|-u]
```

- `-v|-v` Display the command version and exit.
- `-u|-u` Display command usage information and exit.

Examples

1. Display the Maestro user's home directory:

```
$ maestro  
/usr/lib/maestro
```

2. Change directory to the Maestro user's home directory:

```
$ cd `maestro`
```

makecal

Creates custom Maestro calendars. Unless the `-x` or `-z` option is specified, calendars are added with the `composer replace` command.

On UNIX, the Korn shell is required to execute this command.

```
makecal [-v|-u]
makecal [-c name] {-d n } [-i n] [-x|z]
                {-e }
                {-f 1|2|3 {-s date} }
                {-l }
                {-m }
                {-p n }
                {-r n {-s date} }
                {-w n }
```

- `-v|-V` Display the command version and exit.
- `-u|-U` Display the command usage information and quit.
- `-c name` Create calendar with the name *name*, where *name* is eight or less alphanumeric characters (starting with a letter). If omitted, 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 last day of every month.
- `-f 1|2|3` Creates a fiscal month-end calendar selecting the last day of the fiscal month, where:
 - 1 specifies 4-4-5 week format.
 - 2 specifies 4-5-4 week format.
 - 3 specifies 5-4-4 weeks format.
 This option requires the `-s` option also be used.
- `-i n` Put *n* dates in the calendar. This is used to specify the range.
- `-l` Specifies last workday of every month. To work properly, the `holidays` calendar and the `symphony` file must exist.
- `-m` Specifies the first and fifteenth of every month.
- `-p n` Specifies the workday before the *n*th of every month. To work properly, the `holidays` calendar and the `symphony` file must exist.

- `-r n` Specifies every *n*th day. This option requires the `-s` option also be used.
- `-s date` Specifies the starting date for `-f` or `-r` option. Where *date* must be a valid, unambiguous date— see *base-date* for [datecalc](#) on page 10-8. For example, use `JAN 10 1999`, not `1/10/99`.
- `-w n` Specifies the workday after the *n*th of the month. To work properly, the `holidays` calendar and the `symphony` file must exist.
- `-x` Sends the calendar output to `stdout` rather than adding it to Composer.
- `-z` Performs a `composer replace` command for the new calendar and a `compile` of the `symphony` file. WARNING: This option re-submits jobs and schedules from the current day's production. Jobs and schedules may need to be cancelled.

Examples

1. Make a two-year calendar with the last day of every month selected:

```
makecal -e -i24
```

2. Make a 30-date calendar that starts on May 30, 1997 with every third day selected:

```
makecal -r 3 -s "30 MAY 1997" -i30
```

morestdl

Display the contents of standard list files.

```
morestdl [-v|-u]
morestdl [-day days] [-first ] [-name sched.job ]
                [-last ] [job-number ]
                [-num n ]
                [-all ]
```

-v	Display the command version and exit.
-u	Display command usage information and exit.
-day	Display standard list files that are a specific number of days old (1 for yesterday, 2 for the day before yesterday, etc.). The default is zero (today).
-first	Display the first qualifying standard list file.
-last	Display the last qualifying standard list file.
-num	Display the standard list file for a specific run of a job.
-all	Display all of the qualifying standard list files.
-name	The name of the schedule and job whose standard list file is displayed.
<i>job-number</i>	The job number whose standard list file is displayed.

Output Format

For a description of standard list files and their directories refer to [Standard List Files](#) on page 3-7.

Examples

1. Display the standard list file for the first run of job mailxhg1.getmail on the current day:

```
morestdl -first -name mailxhg1.getmail
```

2. Display the standard list file for the second run of job mailxhg1.getmail on the current day:

```
morestdl -num 2 -name mailxhg1.getmail
```

3. Display the standard list files for all runs of job mailxhg1.getmail from three days ago:

```
morestdl -day 3 -name mailxhg1.getmail
```

4. Display the standard list file for the last run of job mailxhg1.getmail from four days ago:

```
morestdl -day 4 -last -name mailxhg1.getmail
```

5. Print the standard list file for job number 455:

```
morestdl 455 | lp -p 6
```

parms

The **parms** command can be used to return the current value of a parameter, change the value of a parameter, or add a new parameter. The command syntax is:

```
parms -v|-u
parms parmname
parms -c parmname ["]value["]
```

-v -V	Display command version information only.
-u -U	Display command usage information only.
<i>parmname</i>	The name of the parameter.
-c	Change the value of an existing parameter. If the parameter does not exist it is created.
<i>value</i>	The value of the parameter (up to 72 characters). Quotation marks are required if the value contains special characters.

Operation

If run without command line options, **parms** enters interactive mode, prompting for parameter names and values. See [Examples](#) below.

Examples

1. Return the value of myparm:

```
parms myparm
```

2. Change the value of myparm:

```
parms -c myparm "item 123"
```

3. Create a new parameter named hisparm:

```
parms -c hisparm "item 789"
```

4. Change the value of myparm, and add herparm:

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

Release units of a resource at either the schedule or job level.

```
release -v|-u|[-s] [cpu#]resource
```

- `-v|-v` Display the command version and exit.
- `-u|-u` Display command usage information and exit.
- `-s` Release resource units only at the schedule level. If omitted, resource units are released at the job level, or at the schedule level if the resource is not found at the job level.
- `cpu` The name of the cpu or cpu class on which the resource is defined. If omitted, the default is this cpu.
- `resource` The name of the resource.

Usage

Units of a resource acquired by a job or schedule at the time it is launched are released automatically when the job or schedule completes. The **release** command is used in a job script to release resources before the job or schedule completes. **release** will release units of a resource in the same order they were acquired. See example 2 [below](#).

Examples

1. In the following schedule, two units of the dbase resource are reserved at the schedule level:

```
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 that the `-s` option can be omitted, since no resources were reserved at the job level.

2. In the following schedule, eight units of the discio resource are reserved at the job level. This is done in two blocks of 5 and 3 so that they can be released incrementally in the same ordered they were acquired.

```
schedule ux1#sked7 on weekdays
:
  job1
  job2 follows job1 needs 5 discio,3 discio
  job3 follows job2
end
```

To release the discio resource incrementally, while job2 is executing, the script for job2 contains the following command lines:

```
...
# Release 5 units of discio:
`maestro`/bin/release discio
...
# Release 3 units of discio:
`maestro`/bin/release discio
...
```

rmstdlist

Display or remove standard list files based on age.

```
rmstdlist [-v|-u|[-p][age]]
```

- v|-v** Display the command version and exit.
- u|-u** Display command usage information and exit.
- p** Display only. The pathnames of qualifying standard list file directories are returned, and no directories or files are removed.
- age* The age, in days, of standard list file directories to be displayed or removed. The default is 10 days, that is, files older than 10 days.

Examples

1. Display the pathnames of standard list file directories that are more than 14 days old:

```
rmstdlist -p 14
```

2. Remove all standard list files (and their directories) that are more than 7 days old:

```
rmstdlist 7
```

showexec

Display the status of executing jobs. This command is intended for standard agent cpus. On domain managers and fault-tolerant agents, the `conman showjobs` command should be used (see [showjobs](#) on page 9-80).

```
showexec [-v|-u|-info]
```

- `-v|-v` Display the command version and exit.
- `-u|-u` Display command usage information and exit.
- `info` Display the script file name instead of user, date, and time.

Output Format

CPU	The cpu on which the job is executing.
Schedule	The name of the schedule in which the job is executing.
Job	The Maestro job name.
Job#	The job number.
User	The job's user name.
Start Date	The date the job started executing.
Start Time	The time the job started executing.
(Est) Elapse	The estimated time, in minutes, the job will execute.

Output Format (-info)

CPU	The cpu on which the job is executing.
Schedule	The name of the schedule in which the job is executing.
Job	The Maestro job name.
Job#	The job number.
JCL	The job's script file name.

Examples

1. Display executing jobs in the standard format:

```
showexec
```

CPU	Schedule	Job	Job#	User	Start Date	Start Time	(Est) Elapse
UX2	#FULLBACK.FULLBACK		#J8299	root	6/05	12:32	80

2. Display executing jobs in the `-info` format:

```
showexec -info
```

CPU	Schedule	Job	Job#	JCL
UX2	#FULLBACK.FULLBACK		#J8299	/users/root/fullback

StartUp

Start Maestro's network management process, Netman.

Security: The user must have `start` access on this cpu.

```
StartUp [-v|-u]
```

`-v|-v` Display the command version and exit.

`-u|-u` Display command usage information and exit.

Usage

The `startUp` command is normally installed in the `/etc/rc` file on each system in a Maestro network, so that Netman is started each time the systems are booted. `startUp` can be used to restart Netman if it is stopped for any reason. The remainder of the process tree can be restarted with a `command start` command (see [page 9-96](#)).

Examples

1. Display the command name and version only:

```
startUp -v
```

2. Start Maestro's network control process, Netman:

```
startUp
```


version

Display Maestro version information. The information is extracted from a product version file.

```
version/version [-Vauh] [-f vfile] [-p product] [file...]
```

-v	Display the command version and exit.
-a	Display information about all product files. The default is to display information about the files specified in the last argument (see <i>file</i> below).
-u	Display command usage information and exit.
-h	Display command help information and exit.
-f vfile	Specify the name of the product version file. The default is a file named <code>version.info</code> in the current working directory, or the product directory specified with -p .
-p product	Specify the Tivoli product name whose directory is directly below the current working directory, and contains a <code>version.info</code> file. If omitted, -f , or its default, is used.
<i>file</i>	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.

Output Format

The output header contains the product name, version, platform, patch level, and install 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 number of the file (if any).
Size(bytes)	The size of the file in bytes.
Checksum	The checksum for the file. Checksums are calculated using the UNIX <code>sum</code> command (the AIX platform uses <code>sum</code> with the <code>-o</code> option).

Usage

Maestro file information is contained in the `version.info` file. This file is placed in the `maestrohome/version` directory during installation. The `version.info` file is in a specific format and should not be altered.

The `version.info` file can be moved to another directory if desired. However, the `-f` option must then be included to locate the file.

The `-p` option is useful if you are in a directory containing the directories of multiple Tivoli products. This allows you to access version information by specifying the product name; for example, `maestro`.

Examples

1. Display information about all Maestro files:

```
version/version -a -f version/version.info
```

2. Display information about the file `customize`:

```
cd version
./version customize
```

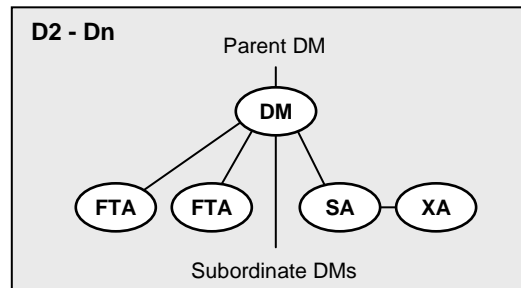
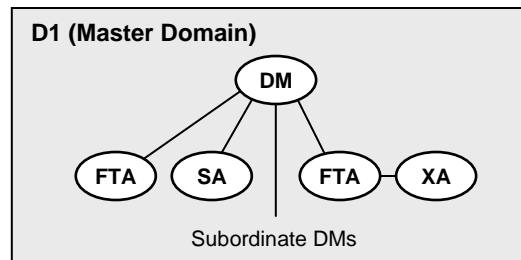
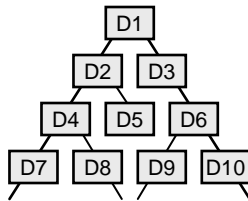
3. Display information about the file `customize`, when `version.info` is located in `/apps/maestro`:

```
cd version
./version -f /apps/maestro/version.info customize
```

A

Maestro Networks

A Maestro network consists of one or more Maestro domains arranged hierarchically. A Maestro domain is a logical grouping of computers, consisting of an domain manager and a number of agents.



Definitions

- | | |
|-----------------------|--|
| Backup Domain Manager | A fault-tolerant agent capable of assuming the responsibilities of its domain manager. |
| Domain | A named group of Maestro cpus, consisting of one or more agents, and a domain manager. All domains have a parent domain, except the master domain. |

Domain Manager (DM)	The management hub in a domain. All communications to and from the agents in a domain is routed through the domain manager. See also <i>Master Domain Manager</i> below .
Extended Agent (xa)	An agent cpu that launches jobs only under the direction of its host. Extended agents can be used to interface Maestro with non-Maestro systems and applications.
Fault-tolerant Agent (fta)	An agent cpu capable of resolving local dependencies and launching its jobs in the absence of a domain manager.
Host (x-host)	The scheduling function required by extended agents. It can be performed by any Maestro cpu, except another extended agent.
Master Domain Manager	The domain manager in the topmost domain of a Maestro network. It contains the centralized master files used to document scheduling objects. It creates the Production Control file at the start of each day, and performs all logging and reporting for the network. See also <i>Domain Manager</i> above .
Master Domain	The topmost domain in a Maestro network.
Parent Domain	The domain directly above the current domain. All domains, except the master domain, have a parent domain. All communications to/from a domain is routed through the parent domain manager.
Standard Agent (sa)	An agent cpu that launches jobs only under the direction of its domain manager.

Maestro for MPE

Maestro networks can contain a mix of MPE (Hewlett-Packard proprietary operating system), Windows NT, UNIX, and other computers and agents. For information about mixed networks refer to [appendix B, *Networking with MPE*](#). For information about Maestro for MPE refer to the *Maestro for MPE User Guide*.

Network Communications

In a Maestro network, agents communicate with their domain managers, and domain managers communicate with their parent domain managers. There are basically two types of communications that take place: 1) start-of-day initialization, and 2) scheduling events in the form of change-of-state messages during the processing day.

Before the start of each new day, the master domain manager creates a production control file called Symphony. Then, Maestro is restarted in the network, and the master domain manager sends a copy of the new Symphony file to each of its automatically linked agents and subordinate domain managers. The domain managers, in turn, send copies to their automatically linked agents and subordinate domain managers. Agents and domain managers that are not set up to link automatically, are initialized with a copy of Symphony as soon as a link operation is executed in Maestro.

Once the network is started, scheduling messages, like job starts and completions, are passed from the agents to their domain managers, through parent domain managers to the master domain manager. The master domain manager then broadcasts the messages throughout the hierarchical tree to update the Symphony files of all domain managers and all fault-tolerant agents running in "full status" mode.

Network Links

Links provide bi-directional communications between Maestro cpus in a network. Links are controlled by the AUTO Link flag, and the Console Manager **Link** and **Unlink** commands. When a link is open, messages are passed between two cpus. When a link is closed, the sending cpu stores messages in a local pobox file, and sends them to the destination cpu when the link is re-opened.

Note: Extended agents do not have links. They communicate with their domain managers via their hosts.

To have a cpu link opened automatically, turn on the AUTO Link flag in the cpu's definition. The link is opened when Maestro is started on the cpu. If the AUTO Link flag is turned off, the link is opened only by executing a **Link** command after the cpu is started.

If you Stop a cpu, the paths from it to other cpus are closed. However, the paths from the other cpus to it remain open until:

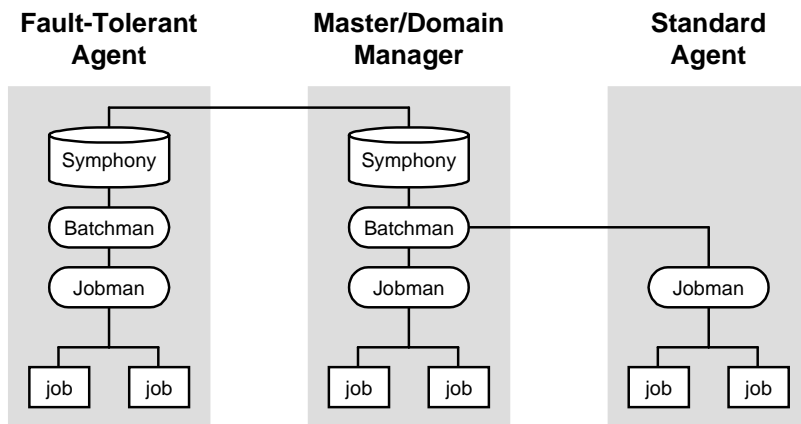
- 1) the stopped cpu is restarted and a **Link** command is issued, or
- 2) until the other cpus' **Mailman** processes time out (see *Local Options* starting on page 2-8 for a description of **Mailman** timeout values).

To be certain that inter-cpu communications is properly restored, you can issue a **Link** command after restarting a cpu.

Network Operation

The Batchman processes on domain managers and fault-tolerant agent cpus operate autonomously, scanning their *symphony* files to resolve dependencies and launch jobs. Batchman launches jobs via the Jobman process. On a standard agent, the Jobman process responds to launch requests from the domain manager's Batchman.

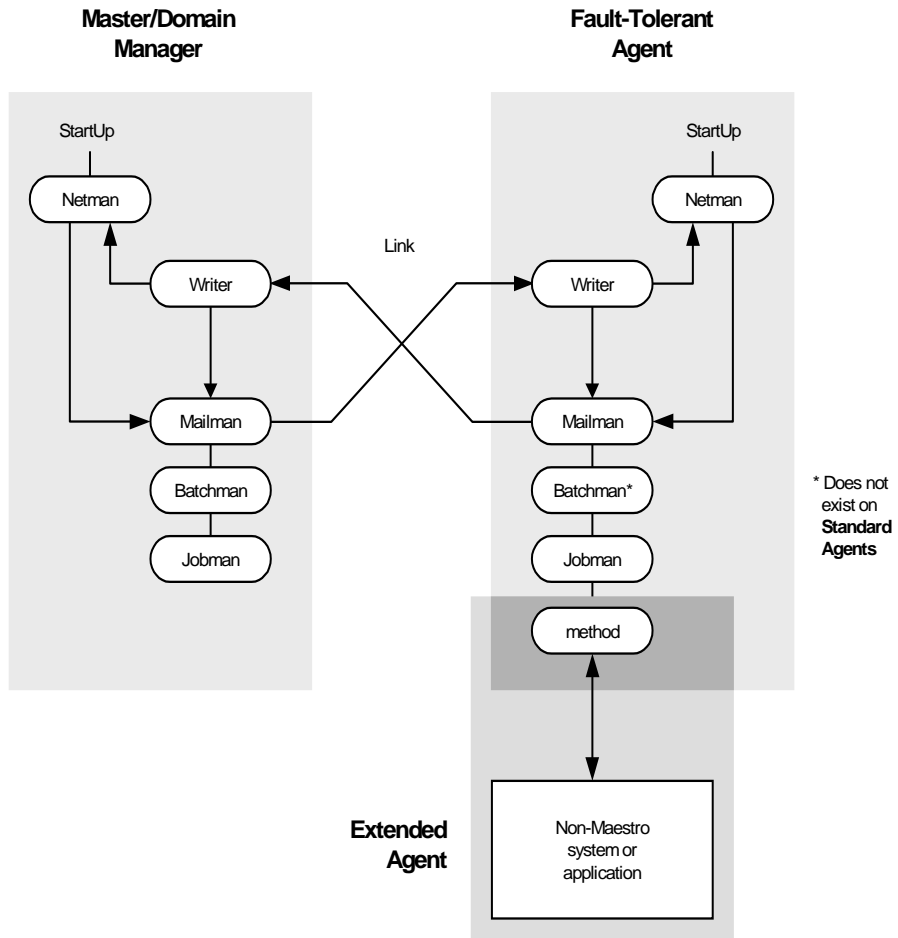
The master domain manager is continuously informed of job launches and completions, and is responsible for broadcasting the information to domain managers and fault-tolerant agents so that they can resolve any inter-cpu dependencies.



The degree of synchronization among the *symphony* files depends on the setting of Full Status and Resolve Dependencies modes in a cpu's definition. Assuming that these modes are turned on, a fault-tolerant agent's *symphony* file contains the same information as the master domain manager's. For more information about these modes, see [CPU Definition](#) on page 4-9.

Network Processes

Netman is started by the `startup` script. The order of process creation is Netman, Mailman, Batchman, and Jobman. On standard agent cpus, Batchman does not exist. All processes, except Jobman, run as the `maestro` user. Jobman runs as `root`.



As network activity begins, Netman receives requests from remote Mailman processes. Upon receiving a request, Netman spawns a Writer process and passes the connection off to it. Writer receives the message and passes it to the local Mailman. The Writer processes (there may be more than one on a domain manager) are started by link requests, and are stopped by unlink requests (or when the communicating Mailman terminates).

Domain managers, including the master domain manager, can communicate with a large number of agents and subordinate domain managers. For improved efficiency, you can define of Mailman servers on a domain manager to distribute the communications load-- see [CPU Definition](#) on page 4-9.

Extended Agents

An extended agent (xa or x-agent) serves as an interface to an external, non-Maestro system or application. It is defined as a Maestro cpu with an access method and a host. The access method communicates with the external system or application to launch and monitor jobs, and test Opens file dependencies. The host is another Maestro cpu (except another xa) that resolves dependencies and issues job launch requests via the method.

Jobs are defined for an x-agent in the same manner as for other Maestro cpus, except that job attributes are dictated by the external system or application.

Extended agent software is available for several systems and applications. The UNIX xa's, included with Maestro, are described below. Please contact your Tivoli Systems sales representative for information about other extended agents.

For information on defining Maestro cpus, see [CPU Definition](#) on page 4-9. For information on writing access methods, see appendix C, [Extended Agent Reference](#).

UNIX Extended Agents

Maestro includes access methods for two types of UNIX extended agents. The Local UNIX method allows a single UNIX computer to operate as two Maestro cpus, both of which can execute Maestro-scheduled jobs. The Remote UNIX access method allows you to designate a remote UNIX computer to run Maestro-scheduled jobs without having Maestro installed on it.

Information about a job's execution is sent to Maestro from an extended agent via the job's **stdlist** file. A Method Options file (described on [page C-6](#)) can specify alternate logons to launch jobs and check Opens file dependencies.

Local UNIX Access Method

The Local UNIX method can be used to define multiple Maestro cpus on one computer: the host cpu and one or more extended agents. When Maestro sends a job to a local UNIX xa, the access method, `unixloc1`, is invoked by the host to execute the job. The method starts by executing the standard configuration script on the host cpu (`maestrohome/jobmanrc`). If the job's logon user is permitted to use a local configuration script, and the script exists as `$HOME/.jobmanrc`, the local configuration script is also executed. The job itself is then executed either by the standard configuration script, or the local configuration script. If neither configuration script exists, the method simply starts the job.

The launching of the configuration scripts, `jobmanrc` and `.jobmanrc`, is configurable in the method script. The method will execute the configuration scripts by default, if they exist. To disable this feature, you must comment out a set of lines in the method script. For more information, examine the script file `maestrohome/methods/unixloc1` on the x-agent's host.

Remote UNIX Access Method

The Remote UNIX access method can be used to designate a non-Maestro UNIX computer to run Maestro-scheduled jobs. When Maestro sends a job to a remote UNIX extended agent, the access method, `unixrsh`, creates a `/tmp/maestro` directory on the non-Maestro computer. It then transfers a wrapper script to the directory and executes it. The wrapper then executes the scheduled job. The wrapper is created only once, unless it is deleted, moved, or is out-dated.

To execute jobs via the x-agent, the job logon users must be given appropriate access on the non-Maestro UNIX computer. To do this, a `.rhost`, `/etc/host.equiv`, or equivalent file should be set up on the computer. If Opens file dependencies are to be checked, `root` access must also be permitted. Contact your system administrator for more information and assistance.

For more information about the access method, examine the script file `maestrohome/methods/unixrsh` on an x-agent's host.

Managing Production for Extended Agents

In general, jobs that run on x-agents behave like other Maestro jobs. Maestro tracks a job's status and records output in the job's `stdlist` file. See section 6, [Managing Production](#) for information information about managing jobs.

Failure Launching Jobs on an X-Agent

If the access method is not located in the proper directory on the x-agent's host or the method cannot be accessed by Maestro, jobs will fail to launch or a file dependency will not be checked. For a job, the Maestro job's logon, or the logon specified in the Method Options file (described [page C-6](#)), must have read and execute permissions for the access method.

When checking a file to satisfy an Opens dependency, `root` is used as the login, unless another login is specified in the Method Options file.

Netman Configuration File

The Netman configuration file exists on all Maestro CPUs. If Netman is installed in Maestro's home directory (the default), the name of the file is *maestrohome/Netconf*. If Netman is installed in a separate directory, the name of the file is *netmanhome/Netconf*. It defines the services provided by Netman. The NetConf file supplied by Tivoli includes comments describing each service. The services are:

- 2001 Start a Writer process to handle incoming messages from a remote Mailman.
- 2002 Start the Mailman process. Mailman, in turn, starts the rest of Maestro's process tree (Batchman, Jobman).
- 2003 Stop Maestro's process tree (Mailman, Batchman, Jobman).
- 2004 Find and return a stdlist file to the requesting Conman process.
- 2005 Switch the domain manager in a domain.
- 2501 Check the status of a remote job.
- 2502 Start the Console Manager-- a service requested by the client side of the Remote Console. See the *Tivoli Remote Console User Guide* for more information.

The Mailman service (2002) can include a parameter that determines the size of Maestro's internal *symphony* table. The table should contain enough space for all the records in the *symphony* file, plus additional space for work submitted after Maestro has started its production run. The syntax for the *NetConf* entry is:

```
2002        son            bin/mailman [-parm value]
```

In the *-parm* option, *value* can be one of the following:

- number* The *symphony* table is built with space for exactly this many records. For example, "*-parm -6000*" builds a table with space for exactly 6000 records. The maximum permitted is 65,535 records.
- number* The *symphony* table is built with space for all records in the *symphony* file, plus this many additional records.

If you receive a message indicating that there are too many jobs scheduled for Batchman to handle, it may be necessary to increase the size of the *symphony* table. Before doing so, contact your Tivoli support representative for help in determining an appropriate size.

Network IP Address Validation

When a TCP/IP connection is established, Netman reads the requester's nodename and IP address from the socket. The IP address and nodename are used to search the `symphony` file for a known Maestro cpu with the following possible results:

- 1) if an IP address match is found, the validation is considered successful; or
- 2) if a nodename match is found, the `gethostbyname()` system call is used to get the associated IP address, and, if it matches the IP address read from the socket, the validation is considered successful; or
- 3) if no match is found in **Symphony**, or the IP address returned by `gethostbyname()` does not match the one read from the socket, the validation is considered unsuccessful.

The Local Option, `nm ipvalidate`, determines the action to be taken if IP validation is unsuccessful. If the option is set to `full`, unsuccessful validation causes Maestro to close the connection and generate an error message. If the option is set to `none`, Maestro permits all connections, but generates a warning message for unsuccessful validation checks.

System Configuration (UNIX only)

IP validation depends on the system call `gethostbyname()` to lookup all the valid addresses for a host. The behavior of this routine varies depending on the system configuration. When `gethostbyname()` uses the file `/etc/hosts`, it returns the first matching entry. If the connection is initiated on an address that appears after the first matching entry, IP validation will fail. To resolve the problem, place the entry used to initiate the connection before any other matching entries in the `/etc/hosts` file.

If `gethostbyname()` uses the "named" name server or the Network Information Service server, and `gethostbyname()` fails, contact your system administrator for assistance.

Error/Warning Messages

Following is a list of the messages for IP validation. If the Local Option `nm ipvalidate` is set to `none` the errors appear as warnings.

1. Maestro `cpu` name is not found in the `symphony` file:

```
IP address validation failed for request: Service num for
program on cpu(os_type). Connection received from IP
address: c_ipaddr. Maestro CPU cpu not found in Symphony
file.
```

2. Call to `gethostbyname()` fails:

```
IP address validation failed for request: Service num for
program on cpu(os_type). Connection received from IP
address: c_ipaddr. gethostbyname() failed, unable to
retrieve IP address of connecting node: node
```

3. IP Addresses returned by `gethostbyname()` do not match the IP address of connecting `cpu`:

```
IP address validation failed for request: Service num for
program on cpu(os_type). Connection received from IP
address: c_ipaddr. System known IP addresses for node name
node: k_ipaddr.
```

4. The IP address specified in the `cpu` definition for the Maestro `cpu` specified in service request packet does not match the IP address of connecting `cpu`:

```
IP address validation failed for request: Service num for
program on cpu(os_type). Connection received from IP
address: c_ipaddr. Maestro known IP address for cpu:
k_ipaddr.
```

5. Regardless of the state of `nm ipvalidate`, the following information message is displayed when IP validation cannot be performed because the `symphony` file does not exist or an error occurs when reading it:

```
IP address validation not performed for request: Service num
for program on cpu(os_type). Connection received from IP
address: c_ipaddr. Cannot open or read Symphony file.
Service request accepted.
```

Where:

<code>num</code>	service number (2001-writer, 2002-mailman,...)
<code>program</code>	program requesting service
<code>cpu</code>	Maestro <code>cpu</code> name of connecting <code>cpu</code>
<code>os_type</code>	operating system of connecting <code>cpu</code>
<code>node</code>	node name or IP address of connecting <code>cpu</code>

Error/Warning Messages

<i>c_ipaddr</i>	IP address of connecting cpu
<i>k_ipaddr</i>	known IP address for connecting cpu

IP validation is always successful in the absence of a `symphony` file. In a Maestro network, the initial link step (AUTO Link option or manual **Link** command) from a domain manager to an agent is normally successful because a `symphony` file does not yet exist. However, if the agent has a `symphony` file from a previous Maestro run, the initial link request may fail if the `symphony` file does not include the name of the domain manager.

Network Recovery

Several types of problems may make it necessary to follow network recovery procedures. These include:

- Initialization problems that prevent agents and domain managers from starting properly at the start of a new day.
- Network link problems that prevent agents from communicating with their domain managers.
- Loss of the a domain manager, which requires a switch to a backup.

Note: In all cases, a problem with a domain manager will affect all of its agents and subordinate domain managers.

Initialization Problems

Initialization problems can occur when Maestro is started for a new day. This can be caused by having Maestro processes running on an agent or domain manager from the previous day or a previous Maestro run. To initialize the agent or domain manager in this situation, do the following:

1. For a domain manager, log into the parent domain manager or the master domain manager. For an agent, log into the agent's domain manager, the parent domain manager or the master domain manager.
2. Run the Console Manager, and execute a **Stop** command for the affected agent.
3. Execute a **Link** command for the affected agent. This will initialize and start the agent.

If the above actions fail to work, a fault-tolerant agent or subordinate domain manager can be run as a stand-alone system. To do this, stop the agent or domain manager, and copy the file *maestrohome/sinfonia* from the master domain manager. Rename the copied file *maestrohome/symphony*, and then start the agent or domain manager. Any inter-cpu dependencies must be resolved locally using appropriate Console Manager commands—**Delete Dependency** and **Release**, for example.

Network Link Problems

Maestro has a high degree of fault tolerance in the event of a communications problem. Each fault-tolerant agent has its own copy of the Symphony file, containing the day's processing. When link failures occur, they continue processing using their own copies of Symphony. Any inter-cpu dependencies, however, must be resolved locally using appropriate Console Manager commands—**Delete Dependency** and **Release**, for example.

While a link is down, any messages destined for a non-communicating cpus are stored by the sending cpus in the *maestrohome/pobox* directory, in files named *cpu_{name}.msg*. When the links are restored, the cpus begin sending their stored messages.

If the links to a domain manager will be down for an extended period, it may be necessary to switch to a standby.

Notes

1. The Console Manager **Submit Job** and **Submit Schedule** commands cannot be used on an agent that cannot communicate with its domain manager.
2. If the link to a standard agent cpu is lost, there is no temporary recovery option available, because standard agents are hosted by their domain managers. In networks with a large number of standard agents, you can choose to switch to a standby domain manager.

Setting Up a Standby Domain Manager

Being prepared for network problems will make recovery easier. In particular, the following steps should be taken.

1. Designate a fault-tolerant agent in the domain to be a standby domain manager.
2. Make certain that the Full Status and Resolve Dependencies modes are selected in the standby's cpu definition.

For a Standby Master Domain Manager:

It may be necessary to transfer files between the master domain manager and its standby. For this reason, the computers must have compatible operating systems. Do not combine UNIX with Windows NT computers; and, in UNIX, do not combine big-endian with little-endian computers.

On a daily basis, following start-of-day processing on the master domain manager, make copies of the `maestrohome/mozart` and `maestrohome/./unison/network` directories, and the `maestrohome/Sinfonia` file. The copies can then be moved to the standby master domain manager if necessary.

Note: For a UNIX master domain manager, if the `maestrohome/mozart` and `./unison/network` directories on the current master domain manager are reasonably static, they can be copied to the standby beforehand. During normal operation, they are hidden when you mount the current master domain manager's directories on the standby. If it becomes necessary to switch to the standby, simply unmounting the current master domain manager's directories will make the standby's copies accessible.

A Note About Network Security

Network security is enforced using IP address validation. As a consequence, cpu linking (AUTO Link option or Link command) may fail if an agent has an old Symphony file that does not contain the new domain manager. If a connection fails, remove the old Symphony file on the agent, and retry the connection.

Losing a Domain Manager

Loss of a domain manager can occur as the result of network linking problems or the failure of the domain manager computer itself. Running without a domain manager has the following effects:

- Inability of agents and subordinate domain managers to resolve inter-cpu dependencies, because activity records broadcasted by the master domain manager are not being received.
- Inability of standard agents that are hosted by the failed domain manager to perform any processing, since they depend on the domain manager for all scheduling and job launching.

If the problem is expected to be of short duration, you can handle it as described in *Network Link Problems* above. If you are uncertain about the duration, or if you want to restore normal agent operations, it will be necessary to switch to a standby. This is described below for domain managers, and extended loss of the master domain manager.

Switching a Domain Manager

Use this procedure to switch to a standby domain manager, including a short term loss of the master domain manager. If you do not expect the master domain manager to be available to handle the next new day turnover (**final** schedule and **Jnextday** job), then use the procedure in [Extended Loss of Master Domain Manager](#) below.

1. Run the graphical Console Manager.
2. Click Domains, or select Domains... from the Objects menu.
3. Select the domain in the SHOWDOMAINS list, and then choose Switch Manager from the Actions menu.
4. In the Switch Manager dialog, enter the name of the backup domain manager in the Domain Manager field, or select the name from the list of cpus produced by clicking the CPUs... button
5. Click OK.

Domain managers remain switched until you execute another switch manager operation. To return to the original domain manager, repeat the above procedure. For a switched master domain manager, you must do this before the new day turnover, unless you followed the [Extended Loss of Master Domain Manager](#) procedure below. For a switched domain manager, other than the master, you can do this at any time without regard to new day processing.

Extended Loss of Master Domain Manager

Use the following procedure to switch to the standby if the original master domain manager is not expected to return to service before the next new day turnover (**final** schedule and **Jnextday** job). For UNIX, use forward slashes in pathnames.

1. Use the Console Manager's **Stop** function to stop Maestro on the master domain manager and its standby.
2. On UNIX, if the master's directories are mounted on any of the agents or domain managers, unmount them.
3. On the standby, install the most recent copies of the master domain manager's *maestrohome/mozart* and *maestrohome/./unison/network* directories. Note that this is unnecessary if you copied the directories beforehand.
4. On the standby, edit the file *maestrohome/mozart/globalopts*, and change the Global Option **master** to the cpu name of the standby.

5. On the standby, use Composer to modify any important schedules that run on the master domain manager-- for example, the **final** schedule. For each of these, change the `cpu` name to the name of the standby.
6. Copy any important job scripts and programs from the master domain manager to the standby-- for example, the **Jnextday** job that runs in the **final** schedule.
7. For UNIX, set up the file system on the standby master domain manager. This discusses HP-UX; for other UNIX platforms, use comparable procedures. On the standby, make certain that the file system containing the `maestrohome/mozart` and `maestrohome/./unison/network` directories has been included in the `/etc/exports` file. If you choose to control the availability of the file system, make the appropriate entries in the `/etc/hosts` file, or the `/etc/netgroup` file on the standby master `cpu`.
8. If necessary, on agents and domain managers, mount the directories from the standby master domain manager.
9. Use the Console Manager's **Switch Manager** function to switch to the backup master. Refer to the procedure in *Switching a Domain Manager* above.

B

Networking with MPE

This appendix describes the configuration requirements and operation of Maestro networks containing a mix of MPE, UNIX, and Windows NT computers. These networks conform to Maestro's standard hierarchy of master, fault-tolerant agent (slave) and standard agent cpus, which are characterized as follows:

- The master cpu can be either an MPE iX, UNIX, or Windows NT computer. It is the administrative hub in a network. It contains Maestro's master scheduling files, and performs all post-production (end-of-day) and pre-production (start-of-day) processing for the network. When a new production day begins, the master processes its own schedules, as well as those of standard agent cpus. It launches its own jobs and issues launch requests to execute jobs on standard agents.
- MPE slave cpus are the equivalent of UNIX and Windows NT fault-tolerant agents. For administrative functions, they rely on the master cpu. When a new production day begins, they process their own schedules and launch their own jobs.
- Standard agent cpus are UNIX or Windows NT computers. For administrative purposes, they rely on the master cpu. When a new production day begins, they execute jobs only in response to launch requests from the master.

Software Versions

In a mixed network containing MPE computers, the following software versions are applicable. All commands documented in this appendix are command line interface (CLI) commands.

Maestro for MPE:

- D.01.10 (MPE V and iX) Can be a slave with an MPE master only.
- D.01.27 and later (MPE iX) Can be the master or a slave.

Maestro for UNIX:

4.03 and later Can be the master, a fault-tolerant agent, or a standard agent.

Maestro for Windows NT:

5.0 and later Can be the master, a fault-tolerant agent, or a standard agent.

To determine the version of your Maestro software, execute the following commands:

On MPE: `run conman.maestro.ccc;info="v"`

On UNIX: `maestrohome/bin/conman -v`

On Windows NT: `maestrohome\bin\conman -v`

Network Considerations

You should review this appendix in its entirety before deciding how to configure a mixed Maestro network. Following is a summary of basic considerations.

- Object definitions and scheduling for slaves and fault-tolerant agents must be done on the master domain manager if the slaves and fault-tolerant agents are not the same platform as the master.
- The standby master domain manager, if required, must be the same platform as the master domain manager.
- Security considerations are different on each platform, MPE, UNIX and Windows NT.

Why choose an MPE master?

Note: Maestro MPE does not support the feature set in Maestro version 6.0 for UNIX and Windows NT. Do not use an MPE computer as the master if you intend to use long object names (greater than eight characters), or Maestro network domains. MPE systems can be used as slaves (fault-tolerant agents) in any domain.

- An MPE master cpu can link to MPE slaves using either NS or TCP/IP. A UNIX or Windows NT master cpu uses only TCP/IP.
- Because a UNIX or Windows NT fault-tolerant agent's security is independent of the MPE master, it can be set to `full status mode on`, and be used for central console management of the network. However, the fault-tolerant agent is still subject to certain limitations on `Conman` commands.

Why choose a UNIX or Windows NT master?

- Maestro for UNIX and Windows NT has graphical interfaces, and more powerful and flexible command sets.
- The `docommand` feature is fully-functional in UNIX and Windows NT schedules, and in the `conman submit` command.
- UNIX and Windows NT Composer can add multiple schedules from a single edit file.

Installation

Installation instructions for Maestro for UNIX and Windows NT are found in the *Maestro Installation Guide*.

To install Maestro for MPE for the first time, refer to section 2 of the *Maestro User Guide For the MPE System User*.

Set Up and Configuration

This section outlines the steps needed to set up Maestro networks with MPE, UNIX, or Windows NT master cpus.

Note: For details about specific commands and transactions see the *Maestro User Guide for the MPE System User*, the *Maestro for UNIX User Guide*, or the *Maestro for Windows NT User Guide*.

Setting Up a Network with MPE Master

After installing the software, follow the steps below to configure each cpu in the network.

On MPE Master:

Use the ARRANGER program to do the following:

1. Use the ACPU transaction to create a cpu definition for each cpu in the network.
2. Use the CSYS transaction to define this cpu as the master.
3. Use the ALNK transaction to define the following links:
 - a. Master-to-master: define a link with TCP/IP commands.
 - b. Master-to-MPE slave running software version D.01.10: define a link with both NS (with `remote hello`) and TCP/IP commands.
 - c. Master-to-MPE slave running software version D.01.20 or later: define a link with TCP/IP commands. Although an NS link (with `remote hello`) is optional, it is recommended. This permits the slave to be initialized and started if the TCP/IP link fails.
 - d. Master-to-UNIX or Windows NT fault-tolerant agent: define a link with TCP/IP commands.
 - e. Master-to-UNIX or Windows NT standard agent: define a link with TCP/IP commands.
4. Use the CTP1, CTP2 and CTP3 transactions to define Maestro's start up, CONMAN, and BATCHMAN parameters.

On Each MPE Slave:

Use the ARRANGER program to do the following:

1. Use the ACPU transaction to create cpu definitions for this cpu and the master.
2. Use the CSYS transaction to identify this cpu and the master.
3. Use the ALNK transaction to define a slave-to-master link containing TCP/IP commands, and, optionally, NS commands. The NS link is required only if you want to do any of the following on the slave:
 - Run CHORUS, COMPOSER, or ARRANGER.

- Submit jobs or schedules using the CONMAN SUBMIT command.
- Rerun jobs using the CONMAN RERUN ;FROM command.
- Use the CONMAN ADDDEP command to add prompt dependencies.

The NS links do not require `remote hello (dsline ;logon= is sufficient)`.

4. Use the CTP2 and CTP3 transactions to define CONMAN and BATCHMAN parameters.

On Each UNIX and Windows NT Fault-tolerant Agent and Standard Agent:

1. Modify the Global and Local Options files to meet your requirements. Default values, suitable for most installations, are inserted by the customize and set up programs.
2. Modify and install the Security file. If you wish, you can do this on one computer and then copy the file to each of the others.

Setting Up a Network with UNIX or Windows NT Master

Note: A Maestro network containing MPE computers must be installed with pre-6.0 compatible scheduling object names.

After installing the software, follow the steps below to configure each cpu in the network.

On UNIX or Windows NT Master:

1. Use Composer to create cpu definitions for all cpus. The definitions also include link information.
2. Modify the Global and Local Options files to meet your requirements. Default values, suitable for most installations, are inserted by the customize and set up programs.

The following Global Options take the place of ARRANGER CTP1 parameters for MPE slaves.

Global Option	Description
<code>rules mode={yes no}</code>	Replaces CTP1-Complete Control Mode. If set to <code>yes</code> , you must also set <code>batchman schedule</code> to <code>yes</code> .
<code>all userjobs in userjobs schedule={yes no}</code>	Replaces CTP1-Place all userjobs in USERJOBS schedule. You must set this to <code>no</code> if <code>rules mode</code> is set to <code>yes</code> .
<code>set mpe job pri to zero={yes no}</code>	Replaces CTP1-Force MPE priority to 0 for all userjobs. You must set this to <code>no</code> if all <code>userjobs in userjobs schedule</code> is set to <code>yes</code> .
<code>batchman schedule={yes no}</code>	Replaces CTP1-Assign priority 10 to Batchman-created schedules. This also affects UNIX and Windows NT cpus.

On UNIX and Windows NT Fault-tolerant Standard Agents:

Modify the Local Options file to meet your requirements. Default values, suitable for most installations, are inserted by the `customize` and `set up` programs.

UNIX and Windows NT Security:

Modify and install the Security file. If you wish, you can do this on one cpu (the master, for example), and then copy the file to each of the others.

On Each MPE Slave:

Use the ARRANGER program to do the following:

1. Use the ACPU transaction to create definitions for this cpu and the master.
2. Use the CSYS transaction to identify this cpu and the master.
3. Use the ALNK transaction to define a slave-to-master link containing TCP/IP commands.
4. Use the CTP2 and CTP3 transactions to define CONMAN and BATCHMAN parameters.

Operation with MPE Master

This section describes the exceptions that apply in a network of MPE, UNIX, and Windows NT cpus with an MPE cpu defined as the master. For additional information about network operation see [Network Operation](#) on page B-11.

Note: For detailed descriptions of operations refer to the *Maestro User Guide for the MPE System User*, the *Maestro for UNIX User Guide*, and the *Maestro for Windows NT User Guide*.

On the MPE Master

Arranger and Composer

- All object definitions and scheduling for UNIX fault-tolerant agents and standard agents must be done with COMPOSER and ARRANGER on the MPE master.
- The MPE COMPOSER does not support the `docommand` feature in job statements for schedules that run on UNIX and Windows NT fault-tolerant agents and standard agents.
- Recovery options for UNIX and Windows NT jobs must be documented with the ARRANGER XJOB transaction.
- In the MPE scheduling language, `scriptname` is not a valid keyword. Instead, use `jobfilename` to define UNIX and Windows NT job scripts. For example:

```
jobfilename "pathname" streamlogon user
```

Conman

- The `docommand` option is not available in the `submit` command to submit commands for execution on a UNIX or Windows NT cpu.

On UNIX and Windows NT Fault-Tolerant Agents

Composer

- You cannot use Composer, except to document Maestro parameters (`parms`), which are local to each cpu.

Conman

The following Conman operations are not permitted:

- Submit jobs and schedules with the `submit` command.
- Add prompt dependencies with the `adddep` command.
- Use the `;from` and `;file` options in the `rerun` command.
- Display jobs and schedules using the `display` command.

On the MPE Slaves

To be successfully started and initialized via TCP/IP, the NETMAN process must already be running on the MPE slave. See *Network Operation* on page B-11.

Operation with UNIX or Windows NT Master

This section describes the exceptions that apply in a network of MPE, UNIX, and Windows NT computers with a UNIX master. For additional information about network operation see [Network Operation](#) on page B-11.

Note: For detailed descriptions of operations refer to the *Maestro User Guide for the MPE System User*, the *Maestro for UNIX User Guide*, and the *Maestro for Windows NT User Guide*.

On the UNIX or Windows NT Master

Composer

- All object definitions and scheduling for MPE slaves must be done with Composer on the UNIX or Windows NT master.
- For automatic job documentation, the UNIX and Windows NT scheduling language job statement accepts MPE-specific keywords, and MPE file and user names. For example:

```
jobfilename file.grp.acct [streamlogon user.acct[,grp]]
```

or:

```
isuserjob [=jobname] streamlogon user.acct[,grp]
```

- The UNIX and Windows NT scheduling language **opens** keyword accepts MPE file names. For example:

```
opens [cpu#]"pathname"|file.grp.acct[(qualifiers)] [...]
```

Conman

- The **showjob** command displays MPE-specific information:

```
[userjob]
>> alias is
>> run again as
SKEL
```

- The `submit docommand` command is not valid when directed to an MPE cpu, even though the command can be executed on the UNIX or Windows NT master.

On the MPE Slaves

ARRANGER

- The only valid ARRANGER transactions are: CTP2, CTP3, XCPU, XLNK, XPRM, XPAS, XSYS.

COMPOSER

- You cannot run COMPOSER.

CONMAN

The following CONMAN operations are not permitted:

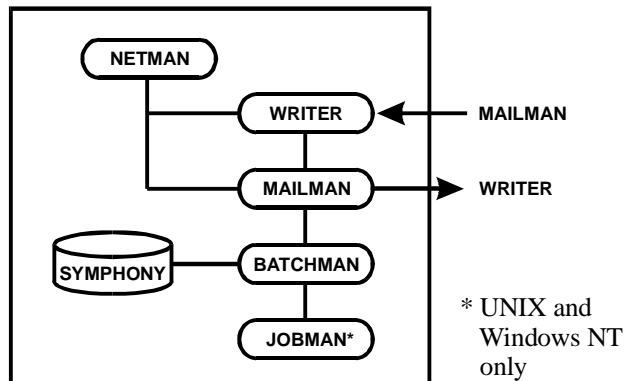
- Submit jobs and schedules with the SUBMIT command.
- Add prompt dependencies with the ADDDEP command.
- Use the ;FROM and ;FILE options in the RERUN command.
- Display jobs and schedules using the DISPLAY command.

Starting and Initialization

To be successfully started and initialized via TCP/IP, the Netman process must already be running on the MPE slave.

Network Operation

This section describes the operation of Maestro processes in a network, and how they are controlled with Conman commands.



Processes

The Network Processes are:

Netman The network manager. It establishes connections between distant Mailman processes and local Writer processes. Netman creates writer processes as required.

Writer The network writer. It passes incoming messages from a distant Mailman to its local Mailman. Writer is created by Netman when a connection is established with a distant Mailman.

The Production Processes are:

Mailman The mail manager. It sends messages to distant Writer processes, and receives messages from its local Writer process.

Batchman The production control process. It resolves dependencies and either launches jobs or directs Jobman to launch jobs.

Jobman The job manager on UNIX and Windows NT cpus only. It launches and tracks jobs under the control of Batchman.

Operational Overview

In normal operation, Netman is started and runs continuously on each cpu in the network. To start a new production day, pre-production processing is first performed on the master cpu. This involves the following steps:

1. Select schedules for the new day, and create a new production control file, `symphony`.
2. Stop production throughout the network while the new `symphony` file is installed on the master cpu.
3. Initialize and start production in the network. The new `symphony` file is distributed to slave and fault-tolerant agent cpus, and their production processes are started.

During the day, processing status on each slave and fault-tolerant agent cpu is sent to the master to update its `symphony` file. The master cpu, in turn, distributes the information to slaves and fault-tolerant agents to update their `symphony` files.

The `symphony` file is given a unique run number when it is created. As a means of maintaining production integrity, the cpus insert the run number in all messages they send to other cpus. This ensures that a cpu that was improperly initialized does not corrupt the processing on other cpus.

Cpus can be stopped and restarted during the production day without adversely affecting Maestro's operation. A restarted cpu will continue processing from the point it was stopped.

Starting Netman

Following installation, Netman is started on each MPE cpu by streaming (or MSTREAMing) the `JBATCHMN` job. On a UNIX cpu, Netman is started by executing the `startup` script. On a Windows NT cpu, Netman is a started service. The production processes are started when pre-production processing is completed and the first `conman start` command is executed on the master cpu.

Netman is also affected by `conman start`, `stop`, and `shutdown` commands as described in the following paragraphs.

The conman start Command

The `conman start` command starts Maestro's production processes. It can be executed locally on any cpu, or on the master cpu to start slave, fault-tolerant agent, and standard agent cpus.

When executed locally, it starts the entire process tree, including Netman if it is not running.

When executed on the master to start a slave, fault-tolerant agent, or standard agent cpu, Netman must be running on the slave, fault-tolerant agent, or standard agent. The only exception is in the case of an MPE master and an MPE slave when an NS master-to-slave link has been defined. In this case, a start is attempted over the TCP/IP link, and, if that fails, the MPE master uses the NS link to run Conman on the MPE slave and issue a `start` command.

The conman stop Command

The `conman stop` command stops Maestro's production processes. It can be executed locally on any cpu, or on the master cpu to stop slave, fault-tolerant agent, and standard agent cpus.

When executed on an MPE cpu, whether locally or from the master, it stops the entire process tree, including Netman. Netman waits for all other processes to stop, including Writers, before it stops. When Netman stops, the JBATCHMN job restreams itself before logging off. When the new iteration of JBATCHMN starts up, it restarts Netman.

When executed on a UNIX or Windows NT cpu, whether locally or from the master, it stops the production processes, but not Netman.

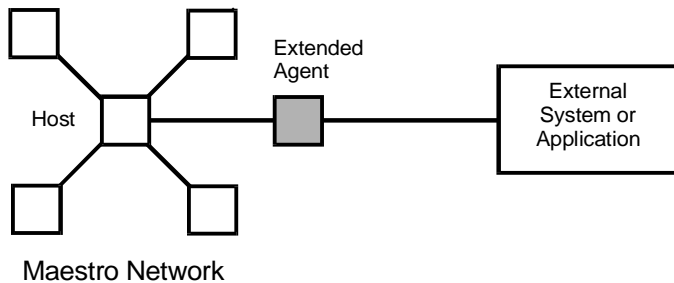
The conman shutdown Command

The `conman shutdown` command is used to unconditionally stop the entire process tree, including Netman. It can only be executed locally. In the case of MPE cpus, the JBATCHMN job does not restream itself, and Netman is not restarted.

C

Extended Agent Reference

This appendix explains the extended agent interface, and provides information intended for programmers creating custom access methods.



Extended Agents

Extended agents are used to integrate Maestro's scheduling and job control with other systems and applications. This is accomplished with Tivoli-supplied and user-supplied scripts or programs called *access methods*. In operation, the access method is executed on a Maestro *host* whenever the extended agent is referenced in the production schedule. For example, to launch a job on an extended agent, Maestro executes the access method, passing in job details as command line options. The access method communicates with the external system or application to launch the job and return job status to Maestro.

Cpu Definition

The name of the access method and the host cpu that executes it are included in the extended agent's cpu definition. For an example of defining an extended agent, see appendix D, *Internetwork Dependencies*.

Method Interface

The interface between Maestro and an access method consists of information passed to the method on the command line, and messages returned to Maestro in **stdout**.

Method Command Line Syntax

Maestro runs an access method using the following command line syntax:

```
method_name -t task other_options -- task_string
```

where:

- | | |
|-----------------------|---|
| <i>method_name</i> | The filename of the access method. All access methods must be stored in the directory path:
<i>maestrohome/methods</i> |
| -t <i>task</i> | The task being requested by Maestro. The tasks are:
LJ Launch a job on an external system.
MJ Manage a previously launched job on an external system. Used only to synchronize if a prior LJ task terminated unexpectedly.
CF Check the availability of a file on an external system. Used to check file "opens" dependencies.
GS Get the status of a job on an external system. Used to check job "follows" dependencies. |
| <i>other_options</i> | The options associated with the task. See <u>Options</u> below. |
| <i>task_string</i> | A string of up to 255 characters associated with the task. See <u>Options</u> below. |

Options

Task	Other Options											Task String	
	-t	-c	-n	-p	-r	-s	-d	-l	-o	-j	-q		-w
LJ	X	X	X	X	X	X	X	X	X	X			<i>lj_string</i>
MJ	X	X	X	X	X	X	X	X	X	X			<i>mj_string</i>
CF	X	X	X							X			<i>cf_string</i>
GS	X	X	X	X		X					X		<i>gs_string</i>

-c *xagent,host,master*

The Maestro names of the extended agent cpu, the host cpu, and the master cpu separated by commas.

-n *node*

The node name of the computer associated with the extended agent, if any. This is defined in the extended agent's cpu definition Node field.

-p *port*

The TCP port number associated with the extended agent, if any. This is defined in the extended agent's cpu definition TCP Address field.

-r *current,specific*

The current Maestro run number and the specific run number associated with the job separated by a comma. The current and specific run numbers may be different if the job was carried forward from an earlier run.

-s *sched*

The name of the job's schedule.

-d *sched_date,epoch*

The schedule date (*yyymmdd*) and the epoch equivalent separated by a comma.

-l *user*

The job's user name. This is defined in the job definition Logon field.

-o *stdlist*

The full pathname of the job's standard list file. Any output from the launched job must be written to this file.

-j *job_name,id*

The job's name and a unique identifier assigned by Maestro separated by a comma. The name is defined in the job definition Job Name field.

- `-q qualifier` The qualifier to be used in a **test** command issued by the method against the file. See [Specifying a Test Qualifier for a File Dependency](#) on page 5-43.
- `-w timeout` The amount of time, in seconds, Maestro waits to get a reply on an external job prior to sending a SIGTERM signal to the access method. If none is specified the default is 300.
- `-- lj_string` Used with the **LJ** task. For the job to be launched, the string from the Script File or Command field of the job definition. See [Defining Jobs](#) on page 4-32.
- `-- mj_string` Used with the **MJ** task. The information provided to Maestro by the method in a **%CJ** response to an **LJ** task. Usually, this identifies the job that was launched. For example, a UNIX method can provide the **pid** number of the job it launched, which is then sent by Maestro as part of a Manage Job (**MJ**) task.
- `-- cf_string` Used with the **CF** task. For a file "opens" dependency, the string from the Opens Files field of the schedule definition. See [Opens Files Panel: Selecting Schedule File Dependencies](#) on page 5-21 or [Opens Files Panel: Selecting Job File Dependencies](#) on page 5-41.
- `-- gs_string` Used with the **GS** task. The format is:
follows_job[, *job_id*]
where:
follows_job The string from the Follows Sched/Job list of the schedule definition. See [Follows Sched/Job Panel: Select Processing for a Schedule to Follow](#) on page 5-17 or [Follows Sched/Job Panel: Selecting Processing for a Job to Follow](#) on page 5-36.

job_id An optional job identifier received by Maestro in a %CJ response to a previous GS task.

Method Response Messages

Methods return information to Maestro in messages written to **stdout**. Each line starting with a percent sign (%) and ending with a new line is interpreted as a message by Maestro. The messages have the following format:

```
%CJ state [mj_string|job_id]
%JS [cpu_time]
%UT [error_message]
```

where:

CJ Change the job state.

state The state of the job. All Maestro job states are valid except HOLD and READY. See [SHOWJOBS Window](#) on page 6-41 for job states. For the GS task, the following states are also valid:

ERROR An error occurred.

EXTRN Status is unknown.

mj_string A string of up to 255 characters that Maestro will include in any MJ task associated with the job. See [mj_string](#) on page C-4.

job_id A string of up to 64 characters that Maestro will include in any GS task associated with the job. See [gs_string](#) on page C-4.

JS Indicate successful completion of a job, and provide its elapsed run time.

cpu_time The job's elapsed time in seconds.

UT Indicate that the requested task is not supported by the method.

error_message A string of up to 255 characters that Maestro will include in its error message.

Method Options File

Maestro will read the method options file, if it exists, before executing a method. Use the file to specify login and any other options. If, after starting Maestro, the method options file is edited, the changes do not take effect until Maestro is stopped and restarted.

Syntax

The file can contain additional options for a method, as needed. However, Maestro recognizes only the following options:

```
LJuser=user_name
CFuser=user_name
GSuser=user_name
GStimeout=seconds
```

LJuser	The login to use for the LJ and MJ tasks. The default is the logon from the job definition. See Defining Jobs on page 4-32.
CFuser	The login to use for the CF task. The default is root for UNIX, and for Windows NT it is the user name of the account in which Maestro was installed.
GSuser	The login to use for the GS tasks. The default is root for UNIX, and for Windows NT it is the user name of the account in which Maestro was installed.

Note: If the extended agent's host is a Windows NT computer, these users must be defined as Maestro user objects. See [User Definitions](#) on page 4-54.

GStimeout The amount of time, in seconds, Maestro waits for a response before killing the access method. The default is 300 seconds.

The options file must have the same path name as its access method, with an **.opts** extension. For example, the pathname of the options file for method **netmth** is:

```
maestrohome/methods/netmth.opts
```


Method Execution

Launch Job (LJ) Task

The **LJ** task is used to instruct the extended agent's method to launch a job on an external system or application. Before running the method, Maestro establishes an execution environment. The **LJuser** parameter is read from the method options file to determine the user to run the method. If the parameter is not present, or the options file does not exist, the user name from 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: <code>/bin:/usr/bin</code> . For Windows NT: <code>%SYSTEM%\SYSTEM32</code> .
TZ	The timezone.

If for any reason the method is not executable, the job is placed in the **FAIL** state by Maestro.

Once it is running, the method writes messages to its **stdout** indicating the state of the job on the external system. These are summarized in the following table.

Task	Method Response	Maestro Action
LJ	<code>%CJ state [mj_string]</code>	Set job state to <i>state</i> . Include <i>mj_string</i> in any subsequent MJ task.
and		
MJ	<code>%JS [cpu_time]</code>	Set job state to SUCC .
	Exit code=non-zero	Set job state to ABEND .
	<code>%UT [error_message]</code> and Exit code=2	Set job state to ABEND and display error message.

The normal sequence consists of one or more `%CJ` messages indicating job state changes, and then a `%JS` message before the method exits to indicate that the job ended successfully. If the job is unsuccessful on the external system, 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 (MJ) Task

The **MJ** task is used to synchronize with a previously launched job if Maestro determines that the **LJ** task terminated unexpectedly. Maestro sets up the environment in the same manner as for the **LJ** task— see [Launch Job \(LJ\) Task](#) on page C-7— and passes in the *mj_string*.

If the method locates the specified job on the external system, it responds with the same messages as an **LJ** task— see [Launch Job \(LJ\) Task](#) on page C-7. If the method is unable to locate the job, it exits with a non-zero exit code, causing the job to be placed in the ABEND state by Maestro.

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 through the Maestro console manager. On receiving the signal, the method must attempt to stop (kill) the job on the external system and then exit without writing a *%JS* message.

Check File (CF) Task

The **CF** task is used to request the extended agent's method to check the availability of a file on the external system. Before running the method, Maestro establishes an execution environment. The **CFuser** parameter is read from the method options file to determine the user to run the method. If the parameter is not present, or the options file does not exist, the **root** user is used on UNIX, and for Windows NT the user name of the account in which Maestro was installed is used. If for any reason the method is not executable, Maestro indicates that the file "opens" dependency failed— that is, the file status is set to "NO" and any dependent job or schedule is not released for execution.

Once it is running, the method executes 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 non-zero exit code. This is summarized in the following table.

Task	Method Response	Maestro Action
CF	Exit code=0	Set file state to YES
	Exit code=non-zero	Set file state to NO
	%UT [<i>error_message</i>] 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 (GS) Task

The GS task is used to request the extended agent's method to check the status of a job on the external system. This is necessary when another Maestro 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 to run the method. If the parameter is not present, or the options file does not exist, the `root` user is used on UNIX, and for Windows NT the user name of the account in which Maestro was installed is used. If for any reason the method is not executable, Maestro does not release the dependent job or schedule for execution. If a `job_id` is available from a prior GS task, it is passed to the method.

Once it is running, 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, Maestro re-executes the method with a GS task until one of the following job states is returned in the %CJ message:

ABEND	The job ended abnormally.
SUCC	The job completed successfully.
CANCL	The job was cancelled.
DONE	The job is done, 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 determines how long Maestro will wait for a response before killing the method. See [Method Options File](#) on page C-6. For information about the `bm check status` local option, see [Local Options](#) on page 2-8.

Method responses are summarized in the following table.

Task	Method Response	Maestro Action
GS	%CJ <i>state</i> [<i>job_id</i>]	Set job state to <i>state</i> , include <i>job_id</i> in any subsequent GS task.
	%UT [<i>error_message</i>] 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.

The `cpuinfo` Command

The `cpuinfo` command can be used within an access method to return information from a cpu definition. See [cpuinfo](#) on page 10-6 for complete command information.

Trouble Shooting

Job Standard List Error Messages

All output messages from a method, except those starting with "%", are written to the job's standard list (`stdlist`) file. In the case of `GS` and `CF` tasks, which are not associated with Maestro-launched jobs, method messages are written to Maestro's standard list file. For insight into a problem of any kind, be sure to examine these files.

Method Not Executable

If a method is not executable, the following will occur:

- For `LJ` and `MJ` tasks, the job is placed in the `FAIL` state.
- For `CF` task, the file dependency remains unresolved and the dependent job remains in the `HOLD` state.
- For `GS` task, the job dependency remains 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 Maestro.

Other Error Messages

Console Manager

This error message appears if you issue a Start/Stop or a Link/Unlink for an extended agent:

```
Error executing command: Not implemented for extended agents. [2202.58]
```

This error does not appear if an extended agent using wildcards.

Composer and Compiler

These error messages are generated when Composer encounters invalid syntax in a cpu definition:

```
ACCESS METHOD is syntactically invalid [1116.45]
```

```
Duplicate ACCESS keyword [1116.46]
```

```
Missing or invalid ACCESS METHOD [1116.47]
```

If an extended agent is defined with an access method, but without a host, the following is displayed:

```
"Method needs a Host CPU".
```

Jobman

Error, warning, and information messages are written to Jobman's **stdlist** file for an extended agent. A successful job launch generates the following message:

```
Launched job job_name for cpu_name, #Jjob_id for user logon.
```

Failure to launch a job generates the following message:

```
Error launching job_name for cpu_name: error_text
```

Failure of a check file task generates the following message:

```
Error invoking method_name for cpu_name: error_text
```

Failure of a manage job task generates the following message:

```
Error managing job_name for cpu_name using method_name: error_text
```

When a method sends a message to Jobman that is not recognized, the following message is generated:

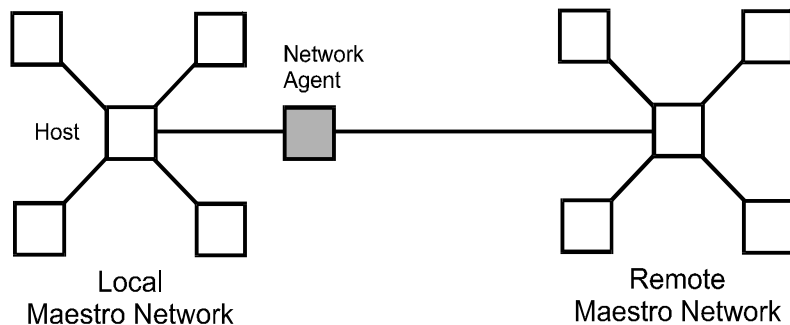
```
Error: message invalid for job_name, #jjob_number for cpu_name using  
method_name.
```

```
"first 64 characters of the offending message"
```

D

Internetwork Dependencies

Maestro's internetwork dependencies permit jobs and schedules in the local network to use jobs and schedules in a remote network as **follows** dependencies. This appendix describes how to use internetwork dependencies.



Network Agents

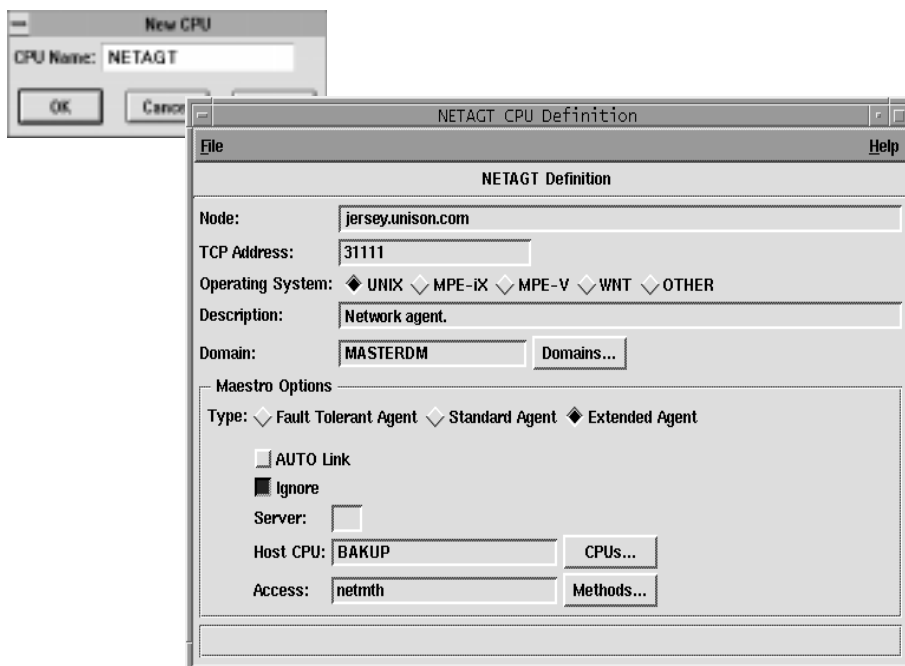
A network agent is a Maestro cpu that handles **follows** dependencies between its local network and a remote Maestro network. Remote **follows** dependencies are assigned to jobs and schedules in the same manner as local **follows** dependencies, except that the network agent's name is included to identify the followed job or schedule. A special schedule named EXTERNAL is created for a network agent in the local network. It contains placeholder jobs to represent each remote **follows** dependency.

The cpu definition for a network agent contains the name of the network access method, **netmth**. The access method is invoked by Maestro each time it needs to check the status of a remote job or schedule. The access method queries the remote network. Maestro continues checking until the remote job or schedule reaches the SUCC, CANCL, or ERROR state.

You can monitor the status of internetwork dependencies with the Console Manager by displaying the network agent's EXTERNAL schedule.

Configuring a Network Agent

Before you can specify an internetwork dependency, you must create a Maestro cpu definition for the remote network. The cpu definition for a remote network is called a *network agent*. Network agent cpu definitions are defined in the standard manner and include a Host CPU name and a Method name.



The fields are:

Node

Enter the node name or IP address of the domain manager or fault-tolerant agent cpu in the remote network to which you want the network agent to connect. If the remote network has an MPE master, enter its node name or IP address.

TCP Address

Enter the TCP port number of the Node specified above. If none is specified, 31111 is assumed.

Operating System	Click OTHER.
Description	Optional free-form textual description of the cpu (up to 40 characters).
Resolve Dependencies	Not used.
Full Status	Not used.
AUTO Link	Not used
Server	Not used.
Ignore	Not used.
Host CPU	Enter the Maestro cpu name of the network agent's host. This can be a domain manager, fault-tolerant agent, or standard agent.
Access	Enter the name of the internetwork dependency method, <code>netmth</code> , that is executed by the Maestro host.

The following shows a command line cpu definition for the network agent NETAGT.

```
cpuname netagt    description "network agent"  
os other  
node  manu       tcpaddr 31111  
for maestro  
    host main     access netmth  
end
```

Method Options File

A method options file can be used to specify the user under which the access method runs, and how often a remote job or schedule dependency is checked. Changes to this file do not take effect until you stop and start Maestro.

Syntax

```
GSuser=login_name
GStimeout=seconds
```

login_name The login used to run the method. If the network agent's host is a Windows NT computer, this user must be defined in Maestro— see [User Definitions](#) on page 4-54. The default is **maestro**.

seconds The amount of time, in seconds, Maestro waits for a response before killing the access method. The default is 300 seconds.

The options file must have the same path name as its access method, with an **.opts** extension. For example, the pathname of the options file for method **netmth** is:

```
maestrohome/methods/netmth.opts
```

Example Entries

```
GSuser=bunyon
GStimeout=400
```

Specifying Internetwork Dependencies

Using the local Composer, internetwork dependencies are specified in the same manner as other Follows dependencies. The Follows Sched/Job tab, in both the Schedule Definition and Job Dependencies windows, provides an optional Internetwork entry box in which to specify the internetwork dependencies. The dependencies are entered in the form:

```
net::net_dep
```

Where *net* is the name of the x-agent cpu, and *net_dep* is the identity of the job or schedule in the remote network. For example:

Internetwork:

When a value is entered in the Internetwork field, click Add to insert it in the Follows Sched/Job list.

Note: Remote jobs and schedules are defined and run on their local network in the standard manner. Their use as internetwork dependencies has no effect on their local behavior.

When remote jobs and schedules are specified as Follows dependencies in local schedules, they are tracked by Conman in a specially created EXTERNAL schedule. Names are generated for the dependencies and they are treated as jobs in the EXTERNAL schedule. For more information on the EXTERNAL schedule and Maestro name generation see [EXTERNAL Schedule and SHOWJOBS Window](#) on page D-6.

Using the Command Line

Internetwork dependencies can be included in schedules composed with the command line **Composer**. For example, use the **follows** keyword as in the following examples:

```
sked6 follows cluster4::site4#skedx.@"
```

and:

```
apjob follows engg::ahab#qa5.jobc1
```

Internetwork Dependencies and Conman

Internetwork dependencies are displayed and manipulated in several ways in Conman.

Ad Hoc Scheduling and Internetwork Dependencies

Internetwork dependencies can be used as Follows dependencies for production created with the Submit actions. The dependencies are specified as they are for other Follows dependencies. See [Specifying Internetwork Dependencies](#) on page D-5 for general information.

EXTERNAL Schedule and SHOWJOBS Window

SHOWJOBS windows display internetwork dependencies in schedules named EXTERNAL. The dependencies are listed as jobs regardless of whether they are Maestro jobs or schedules. There is an EXTERNAL schedule for each network agent. The network agent name is found in the CPU column.

Unique job names are generated as follows:

E*nnnmmss*

where:

nnn is a random number
mm is current minutes
ss is current seconds

The actual name of the job or schedule is stored in the JCL portion of the job record.

EXTERNAL Job States

The state of the jobs is determined by the access method and listed in the State column of the SHOWJOBS table. The states are only as current as the last time the remote network was polled. Therefore, jobs may appear to skip states that occur between polls.

All states for jobs and schedules are listed (except FENCE). In addition, there are two states that are unique to EXTERNAL jobs:

ERROR An error occurred while checking for the remote status.

EXTRN Unknown status. An error occurred, a Rerun action was just performed on the EXTERNAL job, or the remote job or schedule does not exist.

In the event either of the above states occurs, you should check the local Jobman's standard list file for messages and error information.

Taking Action on EXTERNAL Jobs

You can take three actions on remote jobs in an EXTERNAL schedule: Cancel, Rerun, and Confirm.

Note: None of these commands have any affect on the remote job or schedule on the remote network; they simply manipulate the dependency for the local network.

Cancel	Cancels the EXTERNAL job, releasing the dependency for all local jobs and schedules. The status of the dependency ceases to be checked.
Confirm SUCC/ABEND	Sets the status of the EXTERNAL job to SUCC or ABEND, releasing the dependency for all local jobs and schedules. The status of the dependency ceases to be 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's state until you perform a Rerun action.

Taking Action on Internetwork Dependencies for Jobs and Schedules

Internetwork dependencies are listed in the Dependencies column of SHOWJOBS and SHOWSCHEDULES windows in the following format:

net::net_dep

where:

net The cpu name of the network agent. The two colons (::) are a required delimiter. Wildcards are valid.

net_dep The internetwork dependency in the following format:

[cpu#]sched[.job]

If no cpu is specified, the default is the Maestro cpu to which the network agent is connected. This is determined by the Node and TCP Address fields of the network agent's cpu definition. Wildcards are valid.

The Release, Add Dependency, and Delete Dependency actions work the same for internetwork dependencies as they do for other dependencies.

Conman Command Line Specification

Conman commands that can specify internetwork dependencies are listed below with an example based on:

- A local Maestro cpu named local1,
- a schedule defined for local1 named sched1,
- a job in local1#sched1 named job1.
- A Maestro network agent named netagt,
- a Maestro cpu in the netagt remote network named remote1,
- a schedule defined for remote1 named rsched, and
- a job in remote1#rsched named rjob.

adddep job

Add a remote job as a Follows dependency to a job:

```
adj local1#sched1.job1;follows=netagt::remote1#rsched.rjob
```

adddep sched

Add a remote schedule as a Follows dependency to a schedule:

```
ads local1#sched1;follows=netagt::remote1#rsched
```

cancel job

Cancel all EXTERNAL jobs for a network agent (the two commands are equivalent):

```
cj netagt#EXTERNAL.@
```

```
cj netagt::@
```

confirm

Confirm that an EXTERNAL job has finished successfully:

```
confirm netagt::remote1#rsched.rjob;succ
```

deldep job

Delete a remote job dependency from a job:

```
ddj local1#sched1.job1;follows=netagt::remote1#rsched.rjob
```

deldep sched

Delete a remote job dependency from a schedule:

```
dds local1#sched1;follows=netagt::remotel#rsched.rjob
```

release job

Release a job from an internetwork dependency:

```
rj local1#sched1.job1;follows=netagt::remotel#rsched.rjob
```

release sched

Release a schedule from an internetwork dependency:

```
rs local1#sched1;follows=netagt::remotel#rsched.rjob
```

rerun

Rerun an EXTERNAL job (the two commands are equivalent):

```
rr netagt#EXTERNAL.rjob  
rr netagt::remotel#rsched.rjob
```

showjobs;info

Display all the remote dependencies for a network agent with their original names and their Maestro-generated names:

```
sj netagt#EXTERNAL.@;info
```

The original name is displayed in the JOB FILE heading of the display.

submit

Submit an `rm` command into the JOBS schedule with a remote schedule as a Follows dependency:

```
sbd "rm apfile";follows=netagt::remotel#rsched
```



Index

A

- access methods **C-1 to C-12**
 - cpuinfo command **C-10**
 - Local UNIX **A-7**
 - Remote UNIX **A-8**
- Access, cpu definition option **4-12**
- Actions menu, SHOW windows **6-22**
- Actual Production report
 - Detail report sample **7-20**
 - reptr command **7-8**
- ad hoc processing, see *Submit dialogs*
- add command (Composer CLI) **8-7**
- Add Job Dependencies window **6-77 to 6-88**
 - basic operation **6-77**
 - Follows Sched/Job panel **6-81**
 - Needs Resources panel **6-87**
 - Opens Files panel **6-85**
 - Options panel **6-79**
- Add Local Prompt dialog
 - job dependency **5-40**
 - schedule definition **5-20**
- Add Schedule Dependencies window **6-64 to 6-76**
 - basic operation **6-64**
 - Follows Sched/Job panel **6-68**
 - Needs Resources panel **6-75**
 - Opens Files panel **6-72**
 - Options panel **6-66**
 - Prompts panel **6-70**
- adddep job command (Conman CLI) **9-26**
- adddep sched command (Conman CLI) **9-28**
- adding dependencies
 - jobs, Conman **6-52**
 - schedules, Conman **6-39**
- agents

- extended, defined **A-2**
- fault-tolerant, defined **A-2**
- standard, defined **A-2**
- alias names
 - Submit Command dialog **6-20**
 - submit docommand command (Conman CLI) **9-103**
 - submit file command (Conman CLI) **9-106**
 - Submit File dialog **6-19**
 - submit job command (Conman CLI) **9-109**
 - Submit Job dialog **6-18**
 - submit sched command (Conman CLI) **9-112**
 - Submit Schedule dialog **6-17**
- all userjobs in userjobs schedule Option **2-5**
- Allow Command Edit
 - and Add Job Dependencies window **6-78**
 - and Add Schedule Dependencies window **6-65**
 - SHOW windows **6-23**
- altpass command (Conman CLI) **9-30**
- altpri command (Conman CLI) **9-31**
- Arranger CTP1 transaction equivalents **2-5**
- at command (utility) **10-2** to **10-5**
- at keyword (Composer CLI) **8-53**
- At option, Options panel
 - Job Dependencies window **5-33**
 - Schedule Definition window **5-13**
- at schedule **10-2**
- AUTO Link, cpu definition option **4-10**
- Automatic Refresh action, Conman main window **6-5**
- automatically grant logon as batch **2-2**
- Available Calendars dialog, On/Except panel **5-11**
- available lists, using **4-4**

B

- backup domain manager, defined **A-1**
- batch command (utility) **10-2** to **10-5**
- batch schedule **10-2**
- Batchman
 - defined **3-11**
 - Local Options **2-9**
 - operation **3-2**, **A-5** to **A-6**
 - shutdown command (Conman CLI) **9-95**
 - start command (Conman CLI) **9-96**
 - stop command (Conman CLI) **9-99**
- Batchman production process **1-8**

batchman schedule **2-2**
batchman schedule Option
 2-5
Batchman schedules
 at schedule **10-2**
 batch schedule **10-2**
 groupname schedule **10-2**
 JOBS schedule **6-15, 9-102, 9-105, 9-109**
 priority **2-2**
bm check file **2-9**
bm check status **2-9**
browser, for files **4-5**
build command (Composer CLI) **8-8**

C

Calendar Filter, Composer **4-41**
calendars
 access capabilities **2-27**
 Color Chooser dialog **4-44**
 Day of Month dialog **4-43**
 defining **4-39 to 4-44**
 defining with Composer CLI **8-45**
 Definition window **4-42**
 Display window **4-40**
 Filter, Composer **4-41**
 ignore Option **2-3**
 rep3 command **7-3**
 security attributes **2-22**
 User Calendar Listing **7-14**
calendars, using in On/Except panel **5-11**
cancel job command (Conman CLI) **9-33**
Cancel job, SHOWJOBS **6-49**
cancel sched command (Conman CLI) **9-35, 9-38**
Cancel schedule, SHOWSCHEDULES **6-39**
Carry Forward
 prompts **3-21**
 schedules **3-20 to 3-21**
 stageman command **3-22**
Carry Forward option **5-15**
carry job states **2-2**
carry job states Option **2-6**
carryforward Global Option **2-6**
carryforward global option **2-2**
carryforward keyword (Composer CLI) **8-54**

case sensitivity **1-18**
cextract program **7-26**
cf schedules **3-20 to 3-21**
Change Resource dialog, SHOWRESOURCES **6-57**
CLI vs. GUI interfaces **1-6**
Color Chooser dialog, Calendar Definition window **4-44**
colors, setting Conman display of **6-9**
command input and output, Console Manager **6-3**
Command Window
 and Allow Command Edit **6-23**
 Console Manager input and output **6-3**
Command, job definition option **4-33**
commands
 submit docommand command (Conman CLI) **9-101 to 9-103**
 submitting during production **6-20**
 utility see *utility commands*
comments
 in scheduling language (Composer CLI) **8-55**
Comments box
 Job Dependencies window **5-31**
 Schedule Definition window **5-7**
company name **2-3**
compiler command **3-14**
Composer
 Calendar Filter **4-41**
 calendars, defining **4-39 to 4-44**
 common elements **4-2**
 cpu classes, defining **4-18 to 4-23**
 Cpu Filter **4-8**
 cpus, defining **4-6 to 4-17**
 domains, defining **4-24 to 4-26**
 exiting **4-2**
 global prompts, defining **4-48 to 4-50**
 Job Filter **4-31**
 jobs, defining **4-28 to 4-38**
 local prompt definition
 jobs **5-40**
 schedules **5-20**
 main window **4-2**
 messages **4-3**
 parameters, defining **4-45 to 4-47**
 resources, defining **4-51 to 4-53**
 running **4-2**
 schedule Filter **5-4**
 security **4-3**

- status bar **4-3**
- user interfaces **1-6**
- User objects, defining **4-54 to 4-56**
- Composer CLI **8-1 to 8-30**
 - add **8-7**
 - build **8-8**
 - changing prompt **8-3**
 - continue **8-10**
 - cpu classes, defining **8-36**
 - cpus, defining **8-32 to 8-35**
 - create **8-11**
 - delete **8-13**
 - display **8-15 to 8-18**
 - edit **8-19**
 - editor **8-3**
 - exit **8-20**
 - help **8-21**
 - job statement **8-65**
 - jobs, defining **8-39 to 8-42**
 - list of commands **8-6**
 - modify **8-22**
 - new **8-24**
 - redo **8-25**
 - replace **8-27**
 - running **8-1**
 - scheduling language **8-50 to 8-78**
 - special characters **8-4**
 - system commands **8-9**
 - User objects, defining **8-43**
 - validate **8-28**
 - version **8-30**
- confirmed keyword (Composer CLI) **8-56**
- confirming jobs, SHOWJOBS Confirm command **6-50**
- Conman **6-1 to 6-88**
 - adding dependencies to job **6-77**
 - adding dependencies to schedules **6-64**
 - Auto Refresh option **6-5**
 - cancelling, jobs **6-49**
 - cancelling, schedules **6-39**
 - Carry Forward effects **3-19**
 - changing resource units **6-57**
 - Command Window **6-3**
 - confirming jobs **6-50**
 - Console window **6-4**
 - Cpu Filter **6-25**

job details **6-53**
job fence **6-27**
Job Filter **6-45**
job limit for cpus **6-26**
job limit for schedules **6-38**
job states **6-42**
killing jobs **6-49**
Link/Unlink cpus action **6-28**
Link/Unlink domains action **6-30**
main window **6-1**
object filters **6-12 to 6-14**
Preferences dialog **6-9**
priority, job **6-48**
priority, schedule **6-38**
releasing dependencies, jobs **6-49**
releasing dependencies, schedules **6-39**
replying to prompts **6-60**
rerunning jobs **6-49**
Resource Filter **6-57**
running **6-1**
Schedule Filter **6-34**
schedule states **6-32**
SHOWCPUS window **6-24 to 6-28**
SHOWDOMAINS window **6-29 to 6-30**
SHOWFILES window **6-62 to 6-63**
SHOWJOBS window **6-41 to 6-55**
SHOWPROMPTS window **6-59 to 6-61**
SHOWRESOURCES window **6-56 to 6-58**
SHOWSCHEDULES window **6-32 to 6-40**
stageman effects **3-19**
standard list file **6-51**
Start action **6-27, 6-30**
states
 file dependencies **6-62**
 jobs **6-42**
 prompts **6-59**
 schedules **6-32**
Stop action **6-27, 6-30**
submitting ad hoc production **6-15 to 6-20**
Symphony file, changing **6-6**
Conman CLI **9-1 to 9-118**
 adddep job **9-26**
 adddep sched **9-28**
 altpass **9-30**
 altpri **9-31**

cancel job **9-33**
cancel sched **9-35, 9-38**
changing prompts **9-4**
console **9-40**
continue **9-42**
deldep job **9-43**
deldep sched **9-45**
display file **9-47**
display job **9-48**
display sched **9-49**
exit **9-50**
fence **9-51**
help **9-52**
kill **9-53**
limit cpu **9-54**
limit sched **9-55**
link **9-56**
list of commands **9-7 to 9-8**
listsym **9-58**
recall **9-59**
redo **9-61**
release job **9-63 to 9-64**
release sched **9-65 to 9-66**
reply **9-67**
rerun **9-68 to 9-70**
resource **9-71**
running **9-1 to 9-3**
selecting jobs **9-9 to 9-17**
selecting schedules **9-18 to 9-25**
setsym **9-72**
showcpus **9-73 to 9-75**
showdomain **9-76**
showfiles **9-77 to 9-79**
showjobs **9-80 to 9-86**
showprompts **9-87 to 9-89**
showresources **9-90 to 9-91**
showschedules **9-92 to 9-94**
shutdown **9-95**
special characters **9-6**
start **9-96**
status **9-98**
stop **9-99**
submit docommand **9-101 to 9-103**
submit file **9-104 to 9-106**
submit job **9-107 to 9-110**

- submit sched **9-111 to 9-113**
- system commands **9-37**
- tellop **9-115**
- unlink **9-116**
- version **9-118**
- Connect... menu item **6-10**
- console command **2-14**
- console command (Conman CLI) **9-40**
- Console Manager
 - user interfaces **1-6**
- Console Manager. See Conman
- Console window **6-4**
- continue command (Composer CLI) **8-10**
- continue command (Conman CLI) **9-42**
- Continue, job recovery option **4-34**
- Count option, Needs Resources panel
 - Job Dependencies window **5-46**
 - Schedule Definition window **5-27**
- cpu
 - access capabilities **2-27**
 - security attributes **2-23**
- cpu class
 - access capabilities **2-27**
- CPU Class Display window **4-20, 4-24**
- cpu classes **4-18 to 4-23**
 - and Opens file dependencies **5-22, 5-42**
 - defining **4-18 to 4-23**
 - defining with Composer CLI **8-36**
 - Definition window **4-21**
 - Display window **4-20, 4-24**
 - List of window **4-18**
- CPU Definition window **4-9**
- CPU Display window **4-9**
- Cpu Filter
 - Composer **4-8**
 - Conman **6-25**
- cpu types, defined **A-1 to A-2**
- cpuinfo command **10-6 to 10-7**
 - and access methods **C-10**
- cpus **A-1 to A-2**
 - classes see *cpu classes*
 - cpuinfo command **10-6 to 10-7**
 - defining **4-6 to 4-17**
 - defining with Composer CLI **8-32 to 8-35**
 - Definition window **4-9**

display window **4-9**
domains see *domains*
extended agents, defined **A-2**
fault-tolerant agent defined **A-2**
Filters
 Composer **4-8**
 Conman **6-25**
host defined **A-2**
job fence, Conman **6-27**
Link/Unlink actions **6-28, 6-30**
List of window **4-6**
master defined **A-2**
showcpus command (Conman CLI) **9-73 to 9-75**
SHOWCPUS window **6-24 to 6-28**
showdomain command (Conman CLI) **9-76**
SHOWDOMAINS window **6-29 to 6-30**
standard agent defined **A-2**
Start actions **6-27, 6-30**
Stop actions **6-27**
create command (Composer CLI) **8-11**
Cross Reference Report **7-10, 7-22**

D

data panels
 Job Dependencies window **5-30**
 Schedule Definition window **5-6**
database expansion **10-13**
date format **1-10**
datecalc command **10-8 to 10-12**
Day of Month dialog **4-43**
dbexpand command **10-13**
decentralized administration **2-12**
Define menu, Schedule Definition window **5-8**
Definition windows
 calendar **4-42**
 cpu **4-9**
 cpu class **4-21**
 domain **4-26**
 global prompt **4-48**
 job **4-32**
 MPE job **4-38**
 parameters **4-45**
 resource **4-51**
 User **4-54**

deldep job command (Conman CLI) **9-43**
deldep sched command (Conman CLI) **9-45**
delete command **10-14**
delete command (Composer CLI) **8-13**
deleting dependencies
 jobs, Conman **6-52**
 schedules, Conman **6-39**
dependencies
 adding to jobs in Conman **6-77**
 adding to schedules in Conman **6-64**
 for jobs **5-29**
 for schedules **5-1**
 for Submit dialogs **6-16**
 maximum number of **5-1**
 order resolved **5-9, 5-32, 8-52**
dependencies (Composer CLI) **8-52**
Dependent Objects for File dialog, SHOWFILES **6-63**
Dependent Objects for Job window, SHOWJOBS **6-52**
Dependent Objects for Prompt dialog, SHOWPROMPTS **6-61**
Dependent Objects for Resource dialog, SHOWRESOURCES **6-58**
details, for a job **6-53**
disaster command **1-16**
display command (Composer CLI)Composer CLI
 list **8-15 to 8-18**
display file command (Conman CLI) **9-47**
display job command (Conman CLI) **9-48**
display sched command (Conman CLI) **9-49**
Display windows
 calendar **4-40**
 cpu **4-9**
 cpu class **4-20, 4-24**
docommand, job definition keyword (Composer CLI) **8-40**
domain
 Definition window **4-26**
domain definition (command line) **8-37**
domain manager, defined **A-2**
domains **4-24 to 4-26**
 backup domain manager, defined **A-1**
 defined **A-1**
 defining **4-24 to 4-26**
 listing in Composer **4-24**
 showdomain command (Conman CLI) **9-76**
 Stop actions **6-30**
dumpsec command **2-36**

E

- edit command (Composer CLI) **8-19**
- end keyword (Composer CLI) **8-57**
- error messages
 - access method **C-11**
 - IP validation **A-11**
- every keyword (Composer CLI) **8-58**
- Every option, Job Dependencies window **5-34**
- Everyday option, On/Except panel **5-10**
- evtsize command **10-15**
- EXCEPT Calendars dialog, ON/Except panel **5-12**
- except keyword (Composer CLI) **8-59**
- exit command (Composer CLI) **8-20**
- exit command (Conman CLI) **9-50**
- exiting
 - Composer **4-2**
 - Conman **6-2**
 - Maestro **1-17**
- extended agent host, defined **A-2**
- extended agents **A-7 to A-8**
 - defined **A-2**
 - jobs **A-7 to A-8**
 - UNIX, Local **A-7**
 - UNIX, Remote **A-8**
- EXTERNAL schedule **6-55**
- EXTERNAL schedule, internetwork dependencies **D-6**
- extract programs **7-23 to 7-38**

F

- fault-tolerant agent, defined **A-2**
- fence command (Conman CLI) **9-51**
- Fence for CPU dialog, SHOWCPUS **6-27**
- file browser dialog **4-5**
- files
 - access capabilities **2-27**
 - default for Conman object filters **6-13**
 - dependency states **6-62**
 - display file command (Conman CLI) **9-47**
 - for Opens dependencies in jobs **5-41**
 - for Opens dependencies in schedules **5-21**
 - objects dependent on **6-63**
 - script files
 - job definition option **4-33**
 - submitting during production **6-19**

- security attributes **2-23**
- showfiles command (Conman CLI) **9-77 to 9-79**
- SHOWFILES window **6-62 to 6-63**
- standard list file **6-51**
- submit file command (Conman CLI) **9-104 to 9-106**
- submitting during production **6-19**
- version.info **10-34**

Filters

- Composer
 - calendars **4-41**
 - cpus **4-8**
 - jobs **4-31**
 - schedules **5-4**

- Conman
 - cpus **6-25**
 - jobs **6-45**
 - resources **6-57**
 - schedules **6-34**

final schedule **1-15, 2-4, 3-28**

follows keyword (Composer CLI) **8-61**

Follows Sched/Job panel

- Add Job Dependencies window **6-81**

- Add Schedule Dependencies window **6-68**

- Job Dependencies window **5-36**

- Schedule Definition window **5-17**

fonts, setting Conman display of **6-9**

fta, ft-agent - see *fault-tolerant agent* **A-2**

Full Status, cpu definition option **4-11**

G

gconman command **4-2, 6-1**

gethostbyname() **A-10**

Global Options **2-1 to 2-7**

Global Prompt Definitions window **4-48**

global prompts, *see prompts*

globalopts file

- changes take effect **2-1**

- location **2-7**

- mounting **1-13 to 1-15**

gmaestro command **1-16, 6-1**

groupname schedules **10-2**

GUI vs. CLI interfaces **1-6**

GUI, introduction **1-16 to 1-20**

H

Help Browser Window **1-19**
help command (Composer CLI) **8-21**
help command (Conman CLI) **9-52**
help, on-line **1-19**
history for jobs **2-3**
holidays calendar, defining **4-44**
HOME variable **3-3**
Host CPU, cpu definition option **4-12**
host cpu, defined **A-2**

I

ignore calendars **2-3**
Ignore, cpu definition option **4-11**
in order keyword (Composer CLI) **8-63**
installation and set up **1-9 to 1-16**
interactive jobs
 stdlist files **3-9**
interactive keyword **4-33, 8-40**
interactive option **6-19, 6-20**
interactive programs, Window NT **8-40**
interfaces, Maestro **1-6**
internetwork dependencies **6-55, D-1 to D-4**
 EXTERNAL schedule **D-6**
 introduction **D-2**
 method options file **D-4**
 MPE master **D-2**
 specifying in Composer **D-5**
IP address validation **A-10 to A-12**

J

jbxtract program **7-23**
Jnextday job **3-28**
Job Dependencies window **5-30 to 5-46**
 command buttons **5-31**
 Comment box **5-31**
 Follows Sched/Job panel **5-36**
 Needs Resources panel **5-44**
 offset, At and Until times **5-33**
 Opens Files panel **5-41**
 Options panel **5-33**
 Prompts panel **5-39, 6-83**
Job Details Listing **7-3, 7-12**
Job File, MPE job definition option **4-38**

Job Filter

Composer **4-31**

Conman **6-45**

Job Histogram **7-5, 7-18**

Job History Listing **7-4, 7-17**

job history runs **2-3**

job limit

for cpus, Conman **6-26**

for schedules, Composer **5-15**

for schedules, Conman **6-38**

job statement (Composer CLI) **8-65**

job termination **3-7**

jobinfo command **10-16**

Jobman

Local Options **2-9**

operation **3-2, A-5 to A-6**

shutdown command (Conman CLI) **9-95**

start command (Conman CLI) **9-96**

stop command (Conman CLI) **9-99**

variables **3-3**

Jobman management process **1-8**

jobmanrc script **3-4**

jobs

access capabilities **2-27**

addep job command (Conman CLI) **9-26**

adding dependencies to in Conman **6-77**

altpri command (Conman CLI) **9-31**

at and batch utility commands **10-2 to 10-5**

cancel job command (Conman CLI) **9-33**

cancelling during production **6-49**

command, submitting during production **6-20**

comments **5-31**

confirming **6-50**

defining **4-28 to 4-38**

defining dependencies for **5-30 to 5-46**

defining with Composer CLI **8-39 to 8-42**

deldep job command (Conman CLI) **9-43**

dependencies, adding in Conman **6-52**

dependencies, deleting in Conman **6-52**

dependencies, maximum **5-1**

dependencies, order resolved **5-32**

display job command (Conman CLI) **9-48**

execution of **3-1 to 3-9**

extended agent **A-7 to A-8**

Filters

Composer **4-31**
Conman **6-45**
history runs **2-3**
Job Definition window **4-32**
Job Dependencies window **5-30**
job details and statistics **6-53**
Job Details Listing **7-12**
Job Histogram **7-18**
Job History Listing **7-17**
job statement in schedules (Composer CLI) **8-65**
job steps (Conman CLI) **9-68 to 9-70**
jobinfo command **10-16**
kill command (Conman CLI) **9-53**
killing **6-49**
List of window **4-28**
logging statistics **3-23 to 3-25**
maximum dependencies **5-1**
Maximum Run Time **6-54**
Minimum Run Time **6-54**
MPE Job Definition window **4-38**
naming rules **4-28**
objects dependent on **6-52**
order in schedules **5-29**
parameter use in defining **4-33, 8-41**
priority, Conman **6-48**
Recovery Job option, Composer **4-34**
release job command (Conman CLI) **9-63 to 9-64**
releasing dependencies **6-49**
rep1 command **7-3**
rep11 command **7-7**
rep7 command **7-4**
rep8 command **7-5**
rerun command (Conman CLI) **9-68 to 9-70**
rerunning during production **6-49**
security attributes **2-24**
showexec utility command **10-30 to 10-31**
showjobs command (Conman CLI) **9-80 to 9-86**
SHOWJOBS window **6-41 to 6-55**
standard list file **6-51**
states **6-42, 9-16, 9-81**
submit docommand command (Conman CLI) **9-101 to 9-103**
submit file command (Conman CLI) **9-104 to 9-106**
submit job command (Conman CLI) **9-107 to 9-110**
submitting during production **6-18**
termination **3-7**

Jobs panel, Schedule Definition window **5-28**
JOBS schedule **6-15, 9-102, 9-105, 9-109**
jobstdl command **10-18 to 10-19**

K

keyboard, using to navigate GUI **1-18**
kill command (Conman CLI) **9-53**
killing jobs, SHOWJOBS **6-49**

L

limit cpu command (Conman CLI) **9-54**
Limit for CPU dialog, SHOWCPUS **6-26**
Limit for Schedule dialog, SHOWSCHEDULES **6-38**
limit keyword (Composer CLI) **8-67**
Limit option, for schedule definition **5-15**
limit sched command (Conman CLI) **9-55**
Link action, cpus **6-28**
Link action, domains **6-30**
link command (Conman CLI) **9-56**
links
 defining **4-6 to 4-14**
 explained **A-3**
 link command (Conman CLI) **9-56**
 showcpus command (Conman CLI) **9-73 to 9-75**
 unlink command (Conman CLI) **9-116**
list command (Composer CLI)Composer CLI
 print **8-15 to 8-18**
List of domains, Composer **4-24**
List of windows, Composer
 Calendars **4-39**
 CPU Classes **4-18**
 CPUs **4-6**
 Jobs **4-28**
 Schedules **5-1**
list title bar, SHOW windows **6-23**
lists, available **4-4**
listsym command (Conman CLI) **9-58**
local configuration script **3-5**
 example **3-9**
Local Options **2-8 to 2-15**
local prompt definition, in schedules **5-20**
Local UNIX access method **A-7**
LOCAL_RC_OK variable **3-4**
localopts file

- changes take effect **2-8**
- location **2-14**
- mounting **1-13 to 1-15**
- sample listing **2-15**
- log files names **3-19**
- logman command **3-23 to 3-25**
- LOGNAME variable **3-3**
- Logon, job definition option **4-32**

M

Maestro

- case sensitivity **1-18**
- Console window **6-4**
- help **1-19**
- Main Window **1-16**
- user interfaces **1-6**

maestro command **10-20**

Maestro Composer main window **4-2**

Maestro Conman main window **6-1**

Maestro console

- console command (Conman CLI) **9-40**
- telop command (Conman CLI) **9-115**

Maestro Main Window **1-16**

Maestro Resource Listing **7-3, 7-16**

MAESTROCOLUMNS variable **8-2**

MAESTROLINES variable **8-2**

MAESTROLPCOLUMNS variable **7-2, 8-2, 9-3**

MAESTROLPLINES variable **7-2, 8-2, 9-3**

MAIL_ON_ABEND variable **3-4**

Mailman

- management process, defined **1-8**
- operation **A-5 to A-6**
- Options **2-10**
- shutdown command (Conman CLI) **9-95**
- start command (Conman CLI) **9-96**
- stop command (Conman CLI) **9-99**

main windows

- Maestro **1-16**
- Maestro Composer **4-2**
- Maestro Conman **6-1**

makecal utility command **10-21 to 10-22**

makesec command **2-37**

master cpu

- changing **A-15**
- defined **A-2**
- naming **2-4, 2-10**
- master domain, defined **A-2**
- master files
 - build command (Composer CLI) **8-8**
- masksed file
 - and schedulr command **3-12**
 - defined **3-11**
- merge stdlists Option **2-10**
- messages
 - Composer **4-3**
 - console **6-4**
- method options file
 - Network agents **D-4**
- mixed networks (UNIX/MPE) **B-1 to B-13**
- mm read **2-10**
- mm response **2-10**
- mm retry link **2-10**
- mm sound off **2-10**
- mm unlink **2-10**
- modify command (Composer CLI) **8-22**
- morestdl command **10-23**
- mounting network files **1-13 to 1-15**
- mouse, pop-up menu selection **1-18**
- mozart directory **1-13 to 1-15, A-14 to A-17**
- mozart directory option **2-12**
- MPE Job Definition window **4-38**
- MPE slaves, Global Options **2-5**
- MPE/UNIX mixed networks **B-1 to B-13**

N

- navigation, using keyboard and mouse **1-18**
- needs keyword (Composer CLI) **8-68**
- Needs Resources panel
 - Add Job Dependencies window **6-87**
 - Add Schedule Dependencies window **6-75**
 - Job Dependencies window **5-44**
 - Schedule Definition window **5-25**
- NetConf file **A-9**
- Netman
 - management process, defined **1-8**
 - NetConf file **A-9**
 - operation **A-5 to A-6**

- Options **2-10**
 - production phase **3-26**
 - shutdown command (Conman CLI) **9-95**
 - start command (Conman CLI) **9-96**
 - StartUp command and automating network processes **3-26**
 - StartUp utility command **10-32**
- Network agents **D-1 to D-4**
 - method options file **D-4**
- networks
 - changing the master **A-15**
 - initialization problems **A-13**
 - IP validation **A-10 to A-12**
 - link problems **A-14**
 - loss of master **A-15 to A-17**
 - mixed (UNIX/MPE) **B-1 to B-13**
 - mounting, NFS **1-13**
 - operation **A-1 to A-6**
 - recovery **A-13 to A-17**
 - standby master **A-14**
- new command (Composer CLI) **8-24**
- NFS mounting **1-14**
- nm ipvalidate **2-10**
- nm mortal **2-11**
- nm port **2-11**
- nm read **2-11**
- nm retry **2-11**
- Node, cpu definition option **4-9**

O

- object filters **6-12 to 6-14**
 - default files **6-13**
 - defaults **6-13**
 - numbers in title bar **6-12**
 - opening **6-12**
 - pre-configuring **6-13**
 - preconfiguring **6-7**
 - Save & Close button **6-13**
- Objects menu, Composer main window **4-2**
- offset, At and Until times
 - Job Dependencies window **5-33**
 - Schedule Definition window **5-13**
- ON Calendars dialog, On/Except panel **5-11**
- on keyword (Composer CLI) **8-70**
- On/Except panel, Schedule Definition window **5-10**

on-line help **1-19**
Open file browser dialog **4-5**
Open File Qualifier dialog
 Job Dependencies window **5-43**
 Schedule Definition window **5-23**
Opens file dependencies
 Conman, SHOWFILES **6-62** to **6-63**
 objects dependent on **6-63**
 states **6-62**
Opens Files panel
 Add Job Dependencies window **6-85**
 Add Schedule Dependencies window **6-72**
 Job Dependencies window **5-41**
 Schedule Definition window **5-21**
opens keyword (Composer CLI) **8-72**
Operating System, cpu definition option **4-10**
Options panel
 Add Job Dependencies window **6-79**
 Add Schedule Dependencies window **6-66**
 Job Dependencies window **5-33**
 Schedule Definition window **5-13**
options, recovery **4-34**
order
 jobs in schedules **5-29**

P

parameters
 defining **4-45** to **4-47**
 defining with Composer CLI **8-46** to **8-47**
 Definitions window **4-45**
 in a Composer CLI job or schedule definition **8-46**
 in a job or schedule definition **4-47**
 in a job script **4-47**
 parms command **10-25**
 security attributes **2-25**
 use in job definitions **4-33**, **8-41**
 User Parameters Listing **7-15**
parameters directory option **2-12**
parms command **4-47**, **10-25**
Password, User scheduling object option **4-55**
passwords
 altpass command (Conman CLI) **9-30**
PATH variable **3-3**
pextract program **7-27**

Planned Production report
 Detail report sample **7-19**
 reptr command **7-8**

Planned Production Schedule **7-7, 7-21**

pobox directory **A-14**

post-production **3-23 to 3-25**
 logman command **3-23 to 3-25**
 reports **3-16**
 reptr command **3-16**

Preferences dialog, Conman **6-9**

pre-production **3-11 to 3-22**
 compiler command **3-14**
 reports **3-16**
 reptr command **3-16**
 schedul command **3-12**
 stageman command **3-18 to 3-22**

print command (Composer CLI) **8-15 to 8-18**

Print dialog, Maestro **1-20**

printing
 unsaved data **4-5**

priority
 altpri command (Conman CLI) **9-31**

Priority for Job dialog, SHOWJOBS **6-48**

Priority for Schedule dialog, SHOWSCHEDULES **6-38**

priority keyword (Composer CLI) **8-74**

Priority option, Options panel
 Job Dependencies window **5-34**
 Schedule Definition window **5-14**

processes, Maestro's production **1-8**

prodsked file
 compiler command **3-14**
 defined **3-11**
 schedul command **3-12**
 validate command (Composer CLI) **8-28**

Production Control file, see *Symphony file* **6-1**

production cycle **3-10**
 automating **3-28**
 post-production **3-23 to 3-25**
 pre-production **3-11 to 3-22**
 production phase **3-26**

production processes **1-8**

production processing **6-1**

prompt keyword (Composer CLI) **8-75**

Prompt Messages Listing **7-3, 7-13**

prompt, Composer command **8-3**

prompt, Conman command **9-4**

prompts

- as job dependencies **5-39**

- as schedule dependencies **5-19**

- Carry Forward **3-21**

- defining global **4-48 to 4-50**

- defining with Composer CLI **8-48**

- Global Prompt Definitions window **4-48**

- local, job dependency **5-40**

- local, schedule definition **5-20**

- objects dependent on **6-61**

- Prompt Messages Listing **7-13**

- recall command (Conman CLI) **9-59**

- Recovery Prompt option, composer **4-35**

- rep2 command **7-3**

- reply command (Conman CLI) **9-67**

- replying to **6-60**

- security attributes **2-25**

- showprompts command (Conman CLI) **9-87 to 9-89**

- SHOWPROMPTS window **6-59 to 6-61**

- states **6-59**

Prompts panel

- Add Job Dependencies window **6-83**

- Add Schedule Dependencies window **6-70**

- Job Dependencies window **5-39**

- Schedule Definition window **5-19**

prxtract program **7-25**

Q

quitting, *see exiting*

R

r11xtr program **7-29**

recall command (Conman CLI) **9-59**

recovery options for jobs **4-34**

redo command (Composer CLI) **8-25**

redo command (Conman CLI) **9-61**

Refresh Window action, Conman SHOW windows **6-22**

release command **10-27 to 10-28**

release job command (Conman CLI) **9-63 to 9-64**

release sched command (Conman CLI) **9-65 to 9-66**

releasing dependencies **6-39**

- job **6-49**

- schedule **6-39**

remote Console Manager **6-10**
remote jobs, see *internetwork dependencies*
Remote UNIX access method **A-8**
rep11 command **7-7**
rep1-rep4 commands **7-3**
rep7 command **7-4**
rep8 command **7-5**
replace command (Composer CLI) **8-27**
replicated systems **4-18**
reply command (Conman CLI) **9-67**
replying to prompts, SHOWPROMPTS **6-60**
reports **7-1 to 7-38**

- extract programs **7-23 to 7-38**
- post-production **3-16**
- pre-production **3-16**
- rep11 command **7-7**
- rep1-rep4 commands **7-3**
- rep7 command **7-4**
- rep8 command **7-5**
- reptr command **3-16, 7-8**
- samples **7-12 to 7-22**
 - Actual Production Detail **7-20**
 - Cross Reference Report **7-22**
 - Job Details Listing **7-12**
 - Job Histogram **7-18**
 - Job History Listing **7-17**
 - Maestro Resource Listing **7-16**
 - Planned Production Detail **7-19**
 - Planned Production Schedule **7-21**
 - Prompt Messages Listing **7-13**
 - User Calendar Listing **7-14**
 - User Parameters Listing **7-15**
- xref command **7-10**

reptr command **3-16, 7-8**
Request option, On/Except panel **5-10**
Requires Confirmation option, Job Dependencies window **5-35**
rerun command (Conman CLI) **9-68 to 9-70**
rerun job, SHOWJOBS **6-49**
Rerun, job recovery option **4-34**
reserved words **4-3**
Resolve Dependencies, cpu definition option **4-11**
resource command (Conman CLI) **9-71**
Resource Definitions window **4-51**
Resource Filter, Conman **6-57**
resources

- changing total units, Conman **6-57**
- defining **4-51 to 4-53**
- defining with Composer CLI **8-49**
- Filter, Conman **6-57**
- Maestro Resource Listing **7-16**
- objects dependent on **6-58**
- release command **10-27 to 10-28**
- resource command (Conman CLI) **9-71**
- Resource Definitions window **4-51**
- security attributes **2-25**
- showresources command (Conman CLI) **9-90 to 9-91**
- SHOWRESOURCES window **6-56 to 6-58**
- retain rerun job name **2-4**
- reextract program **7-28**
- rights for job logons **4-55**
- rmstdlist command **10-29**
- rules mode Option **2-5**

S

- sa, s-agent - see *standard agent* **A-2**
- Save & Close button, Conman object filters **6-13**
- shedlog directory **3-19**
- Schedule Definition window **5-6 to 5-29**
 - Comment box **5-7**
 - data panels **5-6**
 - Define menu **5-8**
 - dependency tabs **5-6**
 - Follows Sched/Job panel **5-17**
 - Jobs panel **5-28**
 - Needs Resources panel **5-25**
 - offset, At and Until times **5-13**
 - On/Except panel **5-10**
 - Opens Files panel **5-21**
 - Options panel **5-13**
 - Prompts panel **5-19**
- Schedule Filter
 - Composer **5-4**
 - Conman **6-34**
- schedule keyword (Composer CLI) **8-77**
- schedules
 - adddep sched command (Conman CLI) **9-28**
 - adding dependencies to in Conman **6-64**
 - at schedule **10-2**
 - batch schedule **10-2**

cancel sched command (Conman CLI) **9-35, 9-38**
cancelling during production **6-39**
Carry Forward **3-20 to 3-21**
cf schedules **3-20 to 3-21**
comments **5-7**
defining **5-6 to 5-29**
Definition window **5-6**
deldep sched command (Conman CLI) **9-45**
dependencies for, defined **5-1**
dependencies, adding in Conman **6-39**
dependencies, deleting in Conman **6-39**
dependencies, maximum **5-1**
dependencies, order resolved **5-9, 8-52**
display sched command (Conman CLI) **9-49**
Filters
 Composer **5-4**
 Conman **6-34**
final schedule **3-28**
groupname schedules **10-2**
JOBS schedule **6-15, 9-102, 9-105, 9-109**
launch time examples **5-15 to 5-16**
limit sched command (Conman CLI) **9-55**
List of window **5-1**
maximum dependencies **5-1**
priority, Composer **5-14**
priority, Conman **6-38**
release sched command (Conman CLI) **9-65 to 9-66**
scheduling language (Composer CLI) **8-50 to 8-78**
security attributes **2-26**
showschedules command (Conman CLI) **9-92 to 9-94**
SHOWSCHEDULES window **6-32 to 6-40**
states **6-32, 9-24, 9-92**
submit sched command (Conman CLI) **9-111 to 9-113**
submitting during production **6-17**
scheduling language
 viewing from SHOWSCHEDULES window **6-40**
scheduling language (Composer CLI) **8-50 to 8-78**
 at **8-53**
 carryforward **8-54**
 case sensitivity **8-52**
 comments **8-55**
 confirmed **8-56**
 dependencies **8-52**
 end **8-57**
 every **8-58**

- except **8-59**
- follows **8-61**
- in order **8-63**
- job statement **8-65**
- limit **8-67**
- list of keywords **8-51**
- needs **8-68**
- on **8-70**
- opens **8-72**
- priority **8-74**
- prompt **8-75**
- schedule **8-77**
- summary **8-50**
- until **8-78**

schedul command **3-12**

Script File, job definition option **4-33**

security **2-16 to 2-37**

- access capabilities **2-27 to 2-31**
- Composer **4-3**
- dumpsec command **2-36**
- makesec command **2-37**
- object attributes **2-22 to 2-26**
- sample listing **2-32 to 2-35**
- user attributes **2-20**
- user definitions **2-17 to 2-19**

Security file

- changes take effect **2-16, 2-37**
- mounting **1-13 to 1-15**
- sample listing **2-32 to 2-35**

Server, cpu definition option **4-11**

services of Netman **A-9**

Set Dependencies option, Jobs panel **5-29**

set mpe job pri to zero Option **2-5**

setsym command (Conman CLI) **9-72**

SHELL_TYPE variable **3-4**

SHOW windows **6-21 to 6-63**

- Actions menu **6-22**
- Allow Command Edit **6-23**
- basics **6-21 to 6-23**
- closing **6-21**
- display area **6-21**
- list title bar **6-23**
- object filters **6-12 to 6-14**
- opening **6-21**
- refreshing **6-22**

- resizing columns **6-22**
- SHOWCPUS **6-24 to 6-28**
- SHOWDOMAINS **6-29 to 6-30**
- SHOWFILES **6-62 to 6-63**
- SHOWJOBS **6-41 to 6-55**
- SHOWPROMPTS **6-59 to 6-61**
- SHOWRESOURCES **6-56 to 6-58**
- SHOWSCHEDULES **6-32 to 6-40**
- Stop Auto Refresh **6-22**
- window title bar **6-21**
- SHOWCPUS
 - Filter dialog **6-25**
 - job fence **6-27**
 - job limit **6-26**
 - Link/Unlink actions **6-28**
 - Start action **6-27, 6-30**
 - Stop action **6-27**
- showcpus command (Conman CLI) **9-73 to 9-75**
- showdomain command (Conman CLI) **9-76**
- SHOWDOMAINS
 - Link/Unlink actions **6-30**
 - Stop action **6-30**
- showexec utility command **10-30 to 10-31**
- SHOWFILES
 - objects dependent on files **6-63**
- showfiles command (Conman CLI) **9-77 to 9-79**
- SHOWJOBS
 - confirming jobs **6-50**
 - dependencies, adding **6-52**
 - dependencies, deleting **6-52**
 - Filter dialog **6-45**
 - job details **6-53**
 - objects dependent on jobs **6-52**
 - priority **6-48**
 - releasing dependencies **6-49**
 - standard list file viewing **6-51**
- showjobs command (Conman CLI) **9-80 to 9-86**
- SHOWJOBS window
 - internetwork dependencies **6-55**
- SHOWPROMPTS
 - objects dependent on prompts **6-61**
- showprompts command (Conman CLI) **9-87 to 9-89**
- SHOWRESOURCES
 - changing total units **6-57**
 - Filter dialog **6-57**

- objects dependent on resources **6-58**
- showresources command (Conman CLI) **9-90 to 9-91**
- SHOWSCHEDULES
 - dependencies, adding **6-39**
 - dependencies, deleting **6-39**
 - Filter dialog **6-34**
 - job limit **6-38**
 - priority **6-38**
 - releasing dependencies **6-39**
 - viewing scheduling language **6-40**
- showschedules command (Conman CLI) **9-92 to 9-94**
- shutdown command (Conman CLI) **3-26, 9-95**
- special characters **8-4**
- stageman command **3-18 to 3-22**
- standard agent, defined **A-2**
- standard configuration script **3-4**
- standard list file
 - viewing in SHOWJOBS **6-51**
- standard list files
 - delete command **10-14**
 - format **3-7**
 - interactive jobs **3-9**
 - jobstdl command **10-18 to 10-19**
 - merge stdlists option **2-10**
 - morestdl command **10-23**
 - naming **3-7 to 3-8**
 - printing **3-9**
 - rmstdlist command **10-29**
 - stdlist showjobs option **9-80**
 - stdlistwidth Option **2-11**
 - viewing remote **3-8**
- Start action, SHOWCPUS **6-27, 6-30**
- start command (Conman CLI) **9-96**
- Start command (Conman) and production **3-26**
- start time, defining for Maestro day **2-4**
- StartUp command **1-15**
- StartUp command and automating network processes **3-26**
- StartUp utility command **10-32**
- states
 - file dependencies **6-62**
 - jobs **6-42, 9-16, 9-81**
 - prompts **6-59**
 - schedule **6-32**
 - schedules **9-24, 9-92**
- statistics

elapsed vs. cpu time **3-24 to 3-25**
logman **3-23 to 3-25**
status bar, Composer windows **4-3**
status command (Conman CLI) **9-98**
status of Maestro **6-4**
stdlist file, see *standard list files* **6-51**
stdlist width **2-11**
Stop action, SHOWCPUS **6-27**
Stop action, SHOWDOMAINS **6-30**
Stop Auto Refresh **6-22**
stop command (Conman CLI) **9-99**
Stop command (Conman) and production **3-26**
Stop, job recovery option **4-34**
streamlogon
 job definition keyword (Composer CLI) **8-40**
Streamlogon, MPE job definition option **4-38**
Submit Command dialog **6-20**
Submit dialogs
 common attributes **6-15 to 6-16**
 dependencies **6-16**
 opening **6-15**
submit docommand command (Conman CLI) **9-101 to 9-103**
submit file command (Conman CLI) **9-104 to 9-106**
Submit File dialog **6-19**
submit job command (Conman CLI) **9-107 to 9-110**
Submit Job dialog **6-18**
submit sched command (Conman CLI) **9-111 to 9-113**
Submit Schedule dialog **6-17**
submitted jobs, at and batch **3-9**
Submitting ad hoc production **6-15 to 6-20**
Symnew file, compiler command **3-14**
Symphony file
 and managing production with Conman **6-1**
 changing **6-6**
 defined **3-11**
 listsym command (Conman CLI) **9-58**
 log files names **3-19**
 setsym command (Conman CLI) **9-72**
 size of internal table **A-9**
 stageman command **3-22**
syntax
 calendars (Composer CLI) **8-45**
 cpu classes (Composer CLI) **8-36**
 cpus (Composer CLI) **8-32 to 8-35**
 jobs (Composer CLI) **8-39 to 8-42**

- parameters (Composer CLI) **8-46 to 8-47**
- prompts (Composer CLI) **8-48**
- resources (Composer CLI) **8-49**
- User objects (Composer CLI) **8-43**
- syslog local **2-11**
- sysloglocal Option **2-13**
- system commands
 - Composer CLI **8-9**
- system commands, executing in Conman CLI **9-37**
- system logging **2-13**

T

- Tabs, dependency, Schedule Definition window **5-6**
- TCP Address
 - cpu definition option **4-9**
 - Maestro default **4-9**
- tellop command (Conman CLI) **9-115**
- terminating jobs, Conman Kill command **6-49**
- test options, Opens Files panel
 - Job Dependencies window **5-43**
 - Schedule Definition window **5-23**
- thiscpu **2-11**
- Total Units option, resource definition **4-51**
- Total Units, resource definition option **4-52**
- trusted domain users **4-55, 8-43**
- types of **A-1 to A-2**

U

- unison network directory option **2-12**
- UNISON_CPU variable **3-3**
- UNISON_EXIT variable **3-4**
- UNISON_HOST variable **3-3**
- UNISON_JCL variable **3-4**
- UNISON_JOB variable **3-3**
- UNISON_JOBNUM variable **3-3**
- UNISON_MASTER variable **3-3**
- UNISON_RUN variable **3-3**
- UNISON_SCHED variable **3-3**
- UNISON_SHELL variable **3-3**
- UNISON_STDLIST variable **3-4**
- UNIX extended agent
 - Local **A-7**
 - Remote **A-8**
- UNIX/MPE mixed networks **B-1 to B-13**

Unlink action, cpus **6-28**
Unlink action, domains **6-30**
unlink command (Conman CLI) **9-116**
until keyword (Composer CLI) **8-78**
Until option, Options panel
 Schedule Definition window **5-14**
Until time expiration event **5-14**
User Calendar Listing **7-3, 7-14**
user interfaces, GUI vs. CLI **1-6**
User Job, MPE job definition option **4-38**
User object
 altpass command (Conman CLI) **9-30**
 Definition window **4-54**
User objects
 defining **4-54 to 4-56**
 defining with Composer CLI **8-43**
User Parameters Listing **7-3, 7-15**
user rights for job logons **4-55**
userjobs schedule **2-7**
utility commands **10-1 to 10-34**
 at and batch **10-2 to 10-5**
 cpuinfo **10-6 to 10-7**
 datecalc **10-8 to 10-12**
 dbexpand **10-13**
 delete **10-14**
 evtsize **10-15**
 jobinfo **10-16**
 jobstdl **10-18 to 10-19**
 list of commands **10-1**
 maestro **10-20**
 makecal **10-21 to 10-22**
 morestdl **10-23**
 parms
 10-25
 release **10-27 to 10-28**
 rmstdlist **10-29**
 showexec **10-30 to 10-31**
 StartUp **10-32**
 version **10-33 to 10-34**

V

validate command (Composer CLI) **8-28**
Value, parameter definition option **4-45**
variables

Jobman **3-3**
jobmanrc **3-4**
version **4-2**
version command (Composer CLI) **8-30**
version command (Conman CLI) **9-118**
version command (utility) **10-33** to **10-34**
version.info file **10-34**

W

Weekdays option, On/Except panel **5-10**
wildcards
 defined **1-18**
 specifying for a Follows job **5-18**
words, reserved **4-3**
Workdays option
 and holidays calendar **4-44**
 On/Except panel **5-10**
wr read **2-11**
wr unlink **2-11**
Writer process **1-8**
Writer, operation **A-5** to **A-6**

X

xa, x-agent - see *extended agent* **A-2**
x-host - see *extended agent host* **A-2**
xref command **7-10**
xrxtct program **7-31** to **7-38**