

Unicenter[®] AutoSys[®] Job Management for UNIX

User Guide

4.5



Computer Associates[®]

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

© 2003 Computer Associates International, Inc.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

Chapter 1: Introduction

Jobs	1–2
Defining Jobs	1–2
Graphical User Interface	1–2
Job Information Language.....	1–2
System Components	1–3
Event Server	1–3
High-Availability Option: Dual-Event Servers	1–4
Event Processor	1–4
High-Availability Option: Shadow Event Processor.....	1–5
Remote Agent	1–5
Example Scenario on UNIX	1–6
Explanation.....	1–7
Interface Components	1–8
Machines	1–8
Instance	1–8
Events	1–9
Alarms	1–10
Utilities	1–10
Basic Functionality	1–11
Explanation	1–12
Extending Functionality	1–13
Contacting Technical Support.....	1–14

Chapter 2: Security

Overview	2-1
Native Security	2-2
Security on Events Sent by Users	2-2
Security on Events Sent by the Event Processor	2-4
System-Level Security	2-7
Database Field Verification	2-7
Job Definition Encryption	2-7
Remote Agent Authentication	2-8
User Authentication	2-8
Event Processor Authentication	2-9
User and Database Administrator Passwords	2-9
Job-Level Security	2-10
Job Ownership	2-10
User Types	2-11
Permission Types	2-11
Granting Permissions	2-12
Job Permissions and Windows	2-13
Security Control	2-13
Superuser Privileges	2-14
Edit Superuser	2-14
Exec Superuser	2-15
Restricting Access to Jobs	2-16
Remote Agent Security	2-17
eTrust Access Control	2-17
Policy Manager	2-19
Security Access	2-20
Disable Security	2-20
Controlling the Event Processor	2-21
Asset-Level Security	2-21
eTrust Resource Classes	2-22
eTrust Access Modes	2-23
Security Enabled Applications	2-33
Security Call Logic	2-35

Chapter 3: Jobs

Job Types and Structure	3–2
Basic Job Information	3–3
Command Jobs	3–3
Box Jobs	3–4
Starting Conditions for Box Jobs	3–4
File Watcher Jobs	3–5
Basic Job Attributes	3–6
Command Job Attributes	3–6
File Watcher Job Attributes	3–6
Box Job Attributes	3–7
Job States and Status	3–8
Example State Diagram: Simple Jobs	3–10
Example State Diagram: Box Jobs	3–12
Starting Parameters	3–14
Starting Parameters and Boxes	3–14
Date/Time Dependencies	3–15
TZ Environment Variable	3–15
Custom Calendars	3–16
Job Dependencies Related to Job Status	3–16
Cross-Instance Job Dependencies	3–18
Event Processors	3–19
Event Servers	3–20
Example Job Dependencies	3–20
Managing Job Status	3–22
Job Dependencies Based on Exit Codes	3–23
Using Exit Codes and Batch Files with Jobs Running on Windows	3–24
Job Dependencies Based on Global Variables	3–25
Job Run Numbers and Names	3–26
Defining Jobs	3–27
Graphical User Interface Components	3–27

Chapter 4: Job Attributes

Job Attributes and Job Definitions.....	4-1
Using JIL to Create a Job Definition.....	4-2
Using the GUI to Create a Job Definition	4-2
Chapter Organization	4-2
Essential Job Attributes	4-4
Attributes Common to All Job Types	4-4
Job Name	4-4
Job Type	4-4
Job Owner	4-4
Command Jobs Attributes	4-5
Command	4-5
Machine to Run On	4-7
File Watcher Job Attributes	4-8
Machine to Run On	4-8
File to Watch For	4-8
Box Job Attributes	4-8
Optional Job Attributes	4-9
Common Job Starting Attributes	4-9
Start Date / Time Dependence.....	4-9
Days of the Week	4-9
Days to Run Through a Custom Calendar	4-10
Days to NOT Run Through a Custom Calendar	4-10
Specific Times of Day to Run	4-10
Time of Day Not to Run	4-11
Specific Times Every Hour to Run	4-11
Job Dependencies (Starting Conditions)	4-12
Common General Attributes.....	4-12
Description.....	4-12
Box Name	4-13
Minimum Runtime Alarm	4-13
Maximum Runtime Alarm.....	4-14
Terminate Due to Runtime.....	4-14
Send Alarm if the Job Fails	4-15
Terminate the Box if the Job Fails	4-15

Terminate the Job if the Box Fails	4-15
Number of Times to Restart a Job	4-16
Time Zone for Job	4-16
Delete Job After Completion.....	4-17
Autohold	4-17
Permissions.....	4-18
Command Job Attributes	4-19
Profile.....	4-19
Redirection of the Standard Input File	4-20
Redirection of the Standard Output File	4-20
Redirection of the Standard Error File	4-21
Job Load.....	4-21
Queue Priority	4-22
Job Overrides	4-22
Maximum Exit Code for Success	4-23
Average Runtimes.....	4-24
Heartbeat-Interval	4-24
Resource Check: File Space	4-25
File Watcher Job Attributes	4-26
Watch File Minimum Size	4-26
Watch Interval	4-26
Resource Check: File Space	4-27
Box Job Attributes	4-27
Box Successful Completion	4-27
Box Failure	4-28
Date and Time Attributes and Time Changes	4-29
The Time Change	4-29
Behavior During Time Change	4-30
Spring Time Change	4-31
Fall Time Change	4-32

Chapter 5: Box Job Logic

Basic Box Concepts	5–1
Default Box Job Behavior	5–1
When You Should Not Use a Box	5–2
What Happens When a Box Runs	5–3
Simple Box Job	5–4
Box Job Attributes and Terminators	5–5
Attributes in a Box Job Definition	5–5
Example of a Non-Default Success Condition	5–5
Attributes in a Job Definition	5–5
Time Conditions in a Box	5–6
Force Starting Jobs in a Box	5–7
How Job Status Changes Affect Box Status	5–8
Examples	5–9
Advanced Conditions in Box Jobs	5–9
Default Box Success and Box Failure	5–11
Explicit Box Success and Box Failure	5–12
Using the Box Terminator Attribute	5–13
Using the Job Terminator Attribute	5–14
Advanced Job Streams	5–15
Scenario on the First of the Month	5–15
Scenario on the Second of Month	5–16
Scenario I on First of the Month	5–17
Scenario II on First of the Month	5–18
Scenario III on First of the Month	5–19

Chapter 6: Defining Jobs Using the GUI

Starting the GUI	6–1
GUI Control Panel	6–2
Job Definition Dialogs	6–3
The Job Definition Dialog	6–3
Date/Time Options Dialog	6–5
Job Definition Advanced Features Dialog	6–6
Creating a Simple Command Job	6–7

Creating a File Watcher Job	6–9
Creating a Dependent Command Job	6–13
Creating a Box Job	6–15
Changing a Job	6–17
Setting Time Dependencies	6–19
Additional Time Setting Features	6–20
Deleting a Job	6–21
Deleting a Box Job	6–22
Specifying One-Time Job Overrides	6–22
Setting Job Overrides	6–24
Customizing the Job Definition GUI	6–25
Database Connection Time-Out Interval	6–25
Job Definition Title Bar Text and Icon Text	6–26

Chapter 7: Defining Jobs Using JIL

Job Information Language (JIL)	7–1
JIL Syntax Rules	7–1
Rule 1	7–1
Rule 2	7–2
Rule 3	7–2
Rule 4	7–2
Rule 5	7–2
Rule 6	7–2
Rule 7	7–3
JIL Subcommands	7–3
Submitting Job Definitions	7–4
Running JIL	7–4
Creating a Simple Command Job	7–5
Creating a File Watcher Job	7–6
Creating a Dependent Command Job	7–7
Creating a Box	7–8
Adding Machines	7–9
Changing a Job	7–10
Setting Time Dependencies	7–11

Additional Time Setting Features	7-11
Deleting a Job	7-13
Deleting a Box Job	7-13
Specifying One-Time Job Overrides	7-14
Setting Job Overrides	7-15
Example JIL Script	7-16

Chapter 8: The Graphical Calendar Facility

Scheduling Jobs with Calendars	8-2
Starting the Calendar Facility	8-3
Calendar Facility Screens	8-3
Calendar Definition Window	8-4
Menu Bar	8-5
File Menu	8-5
Edit Menu	8-6
Tools Menu	8-7
Options Menu	8-8
Calendar Display	8-9
Date States	8-9
Navigation Controls	8-10
Shift Months Area	8-10
Skip Button Area	8-11
Number of Conflicts Area	8-11
Color Key Area	8-11
Creating a Simple Calendar	8-12
Dates Prior to Today's Date	8-13
Calendar Selection Dialog	8-13
Term Calendar Rule Dialog	8-14
Rule Specification	8-15
Action Area	8-16
Date Range Area	8-16
Date Selection Rule Area	8-16
Examples of Date Selection Rules	8-18
Setting Dates	8-18

Blocking Dates	8–18
Rescheduling Rule	8–20
Move Direction	8–20
To Day	8–20
Example	8–21
Control	8–21
Term Calendar Viewer	8–22
Combining Calendars	8–23
Printing Calendars	8–24
Import/Export File Name Dialog	8–25
Importing Calendar Text Files	8–25
Exporting Calendars	8–27
Customizing the Calendar Facility	8–28
Font Selection Resources	8–29
Object Color	8–29
Date Range	8–30
Window Size	8–30
Print Command	8–30
Calendar Title Bar Text and Icon Text	8–30

Chapter 9: Load Balancing and Queuing Jobs

Real Machines	9–1
Virtual Machines	9–2
Defining Machines	9–2
Specifying Machine Load (max_load)	9–3
Job Attributes and Load Balancing and Queuing	9–3
Specifying Relative Processing Power (factor)	9–4
Using max_load and factor	9–4
Machine Definitions	9–5
Defining a Real Machine	9–6
Deleting Real Machines	9–6
Defining a Virtual Machine	9–7
Deleting Virtual Machines	9–8
Load Balancing	9–9

Force Starting Jobs	9–11
Load Balancing Using ServerVision	9–12
Examining the ServerVision Profile File	9–13
Testing the Algorithms	9–14
ServerVision Monitoring and Reporting	9–15
Queuing Jobs	9–15
Queuing and Simple Load Limiting	9–15
Queueing with Priority	9–17
Subsets—Individual Queues	9–19
Load Units and Virtual Machines	9–20
Multiple Machine Queues	9–20
User-Defined Load Balancing	9–21

Chapter 10: The Operator Console

Alarm Manager	10–1
Operator Console Screens	10–2
Starting the Operator Console	10–2
Job Activity Console	10–3
Menu Bar	10–4
Job List	10–4
Currently Selected Job	10–6
Starting Conditions	10–7
Reports	10–8
Control Area	10–9
Action Buttons	10–9
Send Event Dialog	10–10
Canceling a Sent Event	10–12
Control Buttons	10–13
Job Path (History) Dialog	10–15
Alarm Button	10–15
Exit Button	10–16
Resizing Regions of the Job Activity Console	10–16
Job Selection Dialog	10–17
Specifying a Job by Name	10–18

Specifying Jobs by Status	10–19
Specifying Jobs by Machine	10–19
Selecting Machines	10–20
Sorting the Specified Jobs	10–21
Setting the Job Selection Criteria	10–21
Changing the Console Clock Perspective	10–22
Alarm Manager Dialog	10–23
Alarm Manager Menu Bar	10–24
Alarm List	10–25
Currently Selected Alarm	10–26
Control	10–26
Freeze Frame Button	10–26
Select Job Button	10–27
New Alarm Button	10–27
Registering Responses and Changing Alarm States	10–28
Alarm Selection Dialog	10–29
Select by Type	10–30
Select by State	10–30
Select by Time	10–31
Customizing the Operator Console	10–32
Refresh Time Interval	10–33
Alarm Poll Time Interval	10–33
Changing Fonts	10–33
Freeze Frame at Start Up	10–34
Font Selection Resources	10–34
Label Font Resources	10–34
List Font Resources	10–34
Object Color	10–35
Currently Selected Job Name Field	10–35
Background Color of Variable Fields	10–35
Border Colors	10–35
Primary Interface Color	10–36
Toggle Button Color	10–36
Job List Column Widths	10–36
Atomic Condition Fields	10–36

Operator Console Size	10-37
Default Report Type	10-37
Alarm List Column Width	10-37
Operator Console Title Bar Text and Icon Text	10-38
User-Configurable Action Buttons	10-38
Console Clock Perspective	10-39

Chapter 11: Monitoring and Reporting Jobs

About Monitors and Browsers	11-1
Monitors	11-2
Reports	11-3
Defining Monitors and Reports	11-3
Using the GUI	11-4
Using JIL	11-4
Chapter Organization	11-4
Essential Monitor/Report Attributes	11-5
Common Essential Attributes—General	11-5
Monitor/Report Name	11-5
Mode	11-5
Common Essential Attributes—Events	11-5
All Events	11-6
Alarms	11-6
All Job Status Events	11-7
Individual Job Status Events	11-7
Job Filter	11-7
Essential Report Attributes	11-8
Current Run Only	11-8
Events After a Certain Date/Time	11-8
Optional Monitor/Report Attributes	11-9
Optional Monitor Attributes	11-9
Sound	11-9
Verification Required for Alarms	11-10
Optional Report Attributes	11-10
Defining Monitors and Reports Using the GUI	11-11

The Monitor/Browser Dialog	11-11
Defining an Example Monitor and Report	11-13
Defining a Monitor	11-13
Defining a Report	11-15
Defining Monitors and Reports using JIL.....	11-17
Defining Monitors	11-18
Defining a Report	11-19
Running a Monitor.....	11-20
Customizing the Monitor/Browser	11-20
Database Connection Time-Out Interval.....	11-21
Monitor/Browser Title Bar Text and Icon Text	11-21

Chapter 12: Maintaining

Maintaining the Event Processor	12-1
Starting the Event Processor.....	12-1
eventor Command	12-2
Starting in Global Auto Hold Mode	12-3
Monitoring the Event Processor	12-3
Event Processor Log File Size	12-4
Stopping the Event Processor	12-4
Shadow Event Processor Rollover	12-6
Restoring the Primary Event Processor	12-7
Running in Test Mode.....	12-8
\$AUTOTESTMODE = 1.....	12-9
\$AUTOTESTMODE = 2.....	12-9
Maintenance Commands	12-10
chase	12-10
clean_files	12-11
Backing Up Definitions	12-12
Restoring Definitions.....	12-14
Database Overview	12-15
Event Server Overview	12-15
Using Dual Event Server Mode	12-16
Database Storage Requirements	12-16

Database Architecture	12-17
General Database Maintenance	12-18
Daily Database Maintenance.....	12-18
DBMaint Script	12-19
Modifying the DBMaint Script	12-20
Data Locking.....	12-21
Event Server Rollover Recovery	12-22
Event Server Crash	12-22
Synchronizing the Event Servers	12-23
Improving Database Performance	12-24
Improving Sybase Database Performance	12-24
Improving Oracle Database Performance	12-25
Maintaining Bundled Sybase SQL Servers	12-26
Sybase Architecture.....	12-26
Sybase Environment	12-27
Default Sybase Users.....	12-27
Database Users	12-28
Changing the System Administrator Password	12-28
Starting Sybase	12-29
Stopping Sybase	12-30
Accessing Sybase	12-31
Identifying Processes Connected to the Database	12-32
Displaying the Database Date and Time	12-33
Bundled Sybase Backup and Recovery	12-33
Defining a Dump Device	12-34
Sybase Backup Server	12-35
Dumping the Database	12-37
Loading the Database	12-38
Recovering a Bundled Sybase Database	12-39

Chapter 13: Configuring

Configuration File	13–1
Sample Configuration File	13–2
Configuration File Parameters	13–4
Database Time-Out Period	13–5
DBLibWaitTime (Sybase Only)	13–5
Cross-Instance Database Connections	13–6
XInstanceDBDropTime	13–6
SNMP Cpnnections	13–6
SnmpManagerHosts	13–7
SnmpCommunity	13–7
Database Connections	13–7
DBEventReconnect	13–8
Event Processor Cascading Errors	13–8
EDNumErrors, EDErrTimeInt	13–9
Machines to Check for Running Event Processors	13–9
EDMachines	13–9
Third Machine for Event Processor Contention	13–10
ThirdMachine	13–10
Event Processor Log Disk Space	13–11
Internal Database Maintenance	13–12
DBMaintTime, DBMaintCmd	13–12
Event Transfer	13–12
EvtTransferWaitTime	13–13
Sendevent Retries	13–13
SendeventMaxRetries	13–13
SendeventRetryInterval	13–13
Heartbeats	13–13
Check_Heartbeat	13–14
Shadow Event Processor Pings	13–14
ShadowPingDelay	13–14
Remote Agent Log Files Directory	13–15
AutoRemoteDir	13–15
File Maintenance	13–15
CleanTmpFiles	13–15

RemoteProFiles	13–16
Number of Restart Attempts	13–17
MaxRestartTries	13–17
Calculating the Wait Time Between Restarts	13–18
RestartConstant, RestartFactor, MaxRestartWait	13–18
Method of Load Balancing	13–18
MachineMethod	13–18
KILLJOB Signals	13–19
KillSignals	13–19
Port Number for Remote Agent	13–20
AutoRemPort	13–20
Cross-Platform Scheduling	13–21
AutoSysAgentSupport	13–21
Instance Wide Append Parameter	13–22
AutoInstWideAppendx	13–22
Job Starting Interval	13–23
InetdSleepTime	13–23
Unicenter Event Management Integration	13–24
UnicenterEvents	13–24
The auto.profile File	13–25
Sample Default auto.profile	13–25
Remote Agent Database Connection Settings	13–26
Sybase	13–26
Oracle	13–26
Modifying Remote Agent Settings	13–27
Remote Agent Socket Connection	13–27
Running Two Versions of Remote Agents	13–28
Configuring Remote Authentication	13–29
Configuring Event Processor Authentication	13–29
The /etc/ autostuff File	13–30
Client-Side Security	13–31
User-Defined Alarm Callbacks	13–32
Notification Example	13–33

Chapter 14: Troubleshooting

Event Server Troubleshooting	14-2
Event Server Is Down (Sybase)	14-2
Sybase Deadlock	14-4
Not Enough User Connections (Bundled Sybase)	14-5
archive_events Fails (Bundled Sybase)	14-7
Event Server Will Not Start (Bundled Sybase)	14-9
Event Processor Troubleshooting	14-10
Event Processor Is Down	14-10
Event Processor Will Not Start	14-11
Remote Agent Troubleshooting	14-12
Remote Agent Verification	14-12
autoping	14-12
Database Verification	14-12
Remote Agent Will Not Start	14-13
Remote Agent Will Start - Command Will Not Run	14-16
Remote Agent Starts, Command Runs—No RUNNING Event Is Sent	14-19
xql Will Not Start (Sybase Only)	14-23
Remote Agent Not Found	14-23
Jobs Run Twice	14-25
Job Failure Troubleshooting	14-26
Jobs Run Only From the Command Line	14-26

Appendix A: Integrating with the Mainframe and Unicenter AutoSys Agents for AS/400 and OpenVMS

Definition of Terms	A-2
Related Documentation	A-3
Job Scheduling for the Enterprise	A-4
Prerequisites	A-4
Configuring for Enterprise Job Scheduling	A-5
Stop the Event Processor	A-5
Configure the Machine	A-5
Set the AutoSysAgentSupport Parameter	A-5
Set the AutoSysAgentSupportReceiveSubmit Parameter	A-6

Create the config.EXTERNAL File	A-6
Example of config.EXTERNAL	A-7
Ensure Consistent Integration Settings	A-8
Configure the Communication Components	A-8
Restart the Event Processor	A-8
About asbIII	A-9
Environment Variable for asbIII	A-10
PRIMARYCCISYSID	A-10
Bi-Directional Scheduling	A-11
Running Jobs on AutoSys on Behalf of a Workload Manager	A-11
Unicenter AutoSys JM and Unicenter AutoSys JM Connect Cross-Platform Dependencies	A-12
Job Scheduler Interdependencies	A-13
Notation for Cross-Platform Job Dependencies	A-14
Cross-Platform Dependency Example	A-14
Naming Conventions for Unicenter AutoSys JM Connect Cross-Platform Jobs	A-15
Running Unicenter AutoSys JM Jobs on Unicenter AutoSys Agents	A-16
Agent Job Names and User IDs	A-16
Running Jobs on Agent Managed Machines	A-17
Defining Agent Machines	A-18
Job Definition Examples	A-19
Unicenter AutoSys Agent Machine in an Job Definition	A-20
Log and Trace Information	A-20
Unicenter AutoSys JM Connect and Unicenter AutoSys Agent Job Statuses	A-21
Unsupported Attributes for Unicenter AutoSys JM Connect or Unicenter AutoSys JM Agent Jobs	A-22
Cross-Platform Limitations	A-23

Appendix B: Troubleshooting CCI

Troubleshooting Tools for Remote CCI Connections	B-1
netstat	B-1
ping	B-2
nslookup	B-2
traceroute	B-2
ccinet	B-3
CCI Command Line Controls	B-4

cci show	B-4
cci semashow and cci semaclear X	B-5
cci shutdown	B-5
cci debugon and cci debugoff	B-6
Reinstalling CCI	B-8

Appendix C: General Debugging

Appendix D: Unicenter Integration

This guide is for users responsible for defining jobs to Unicenter® AutoSys® Job Management (Unicenter AutoSys JM) and monitoring and managing these jobs. It assumes familiarity with the operating system on which jobs will be run, and it assumes that you have already installed and are running Unicenter AutoSys JM using the procedures described in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

Unicenter AutoSys JM is an automated job control system for scheduling, monitoring, and reporting. These jobs can reside on any Unicenter AutoSys JM - configured machine that is attached to a network.

A job is any single command, executable, script, or Windows batch file. Each job definition contains a variety of qualifying attributes, including the conditions specifying when and where a job should be run.

As with most control systems, there are many ways to correctly define and implement jobs. It is likely that the way you utilize Unicenter AutoSys JM to address your distributed computing needs will evolve over time. As you become more familiar with both the features of Unicenter AutoSys JM and the characteristics of your own jobs, you will also refine your use of Unicenter AutoSys JM.

However, before you install and use Unicenter AutoSys JM, it is important to understand the basic system, its components, and how these components work together.

This chapter provides a brief overview of Unicenter AutoSys JM, its system architecture, and features.

Jobs

In the Unicenter AutoSys JM environment, a job is a single action that can be performed on a valid client machine. On UNIX, this action can be any single command or shell script, and on Windows, this action can be any single command, executable, or batch file. In addition, job definitions include a set of qualifying attributes.

For information on defining, running, managing, monitoring, and reporting on jobs, see the corresponding chapters in this guide.

Defining Jobs

Using utilities, you can define a job by assigning it a name and specifying the attributes that describe its associated behavior. These specifications make up the job definition. Two methods you can use to create job definitions are as follows:

- Using the Graphical User Interface (GUI).
- Using the Job Information Language (JIL) through a command-line interface.

Graphical User Interface

The GUI lets you interactively set the attributes that describe when, where, and how a job should run. You create job definitions using the GUI Control Panel and the dialogs you can launch from it. The fields in the GUIs correspond to the AutoSys JIL sub-commands and attributes. In addition, from the GUI Control Panel, you can open applications that lets you define calendars, monitors, and reports, and let you monitor and manage jobs.

Job Information Language

JIL is a specification language, with its own syntax, that is used to describe when, where, and how a job should run. When you enter the `jil` command, you get the `jil` command prompt, at which you can enter the job definitions one line at a time using this special language. When you exit the `jil` command-line interface, the job definition is loaded into the database. Alternatively, you can enter the definition as a text file and redirect the file to the `jil` command. In this case, the `jil` command activates the language processor, interprets the information in the text file, and loads this information in the database.

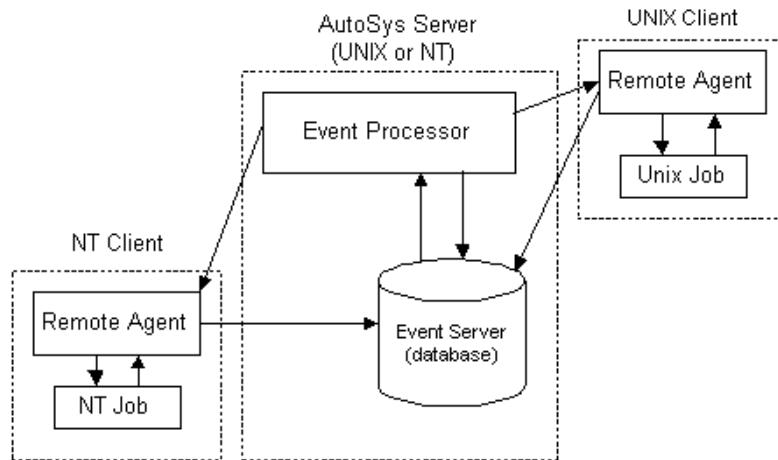
System Components

The main system components are as follows:

- Event server (AutoSys database)
- Event processor
- Remote agent

In addition, Unicenter AutoSys JM provides utilities to help you define, run, and maintain instances and jobs. The included utilities are platform-specific; however, all platforms include the GUI components and JIL. Both the GUI and JIL let you define, manage, monitor, and report on jobs.

The following figure illustrates the Unicenter AutoSys JM system components in a basic configuration. In addition, this figure illustrates the communication paths between the components.



Event Server

The event server or database (the RDBMS) is the data repository for all system information and events as well as all jobs, monitor, and report definitions. Event server refers to the database where all the information, events, and job definitions are stored.

Occasionally, the database is called a data server, which actually describes a server instance. That is, it is either a UNIX or Windows process, and it is associated data space (or raw disk storage), that can include multiple databases or tablespaces.

Note: The database refers to the specific server instance and the “autosys” database for that instance. Some utilities, such as isql (Sybase), let you specify a particular server and database.

High-Availability Option: Dual-Event Servers

Unicenter AutoSys JM can be configured to run using two databases, or dual-event servers. This feature provides complete redundancy. Therefore, if you lose one event server due to hardware, software, or network problems, operations can continue on the second event server without loss of information or functionality.

For various reasons, database users often run multiple instances of servers that are unaware of the other servers on the network. When implementing Unicenter AutoSys JM, the database can run stand-alone for Unicenter AutoSys JM only, or it can be shared with other applications.

For more information on using dual-event servers, see Dual-Event Servers in the chapter “Introduction” in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

Event Processor

The event processor is the heart of Unicenter AutoSys JM; it interprets and processes all the events it reads from the database. Sometimes called the `event_demon`, the event processor is the program, running either as a UNIX process or as a Windows service that actually runs Unicenter AutoSys JM. It schedules and starts jobs.

After you start it, the event processor continually scans the database for events to be processed. When it finds one, it checks whether the event satisfies the starting conditions for any job in the database.

Based on this information, the event processor first determines what actions are to be taken, then instructs the appropriate remote agent process to perform the actions. These actions may be the starting or stopping of jobs, checking for resources, monitoring existing jobs, or initiating corrective procedures.

High-Availability Option: Shadow Event Processor

Unicenter AutoSys JM lets you set up a second event processor, called the shadow event processor. This second processor should run on a separate machine to avoid a single point of failure.

The shadow event processor remains in an idle mode, receiving periodic messages (pings) from the primary event processor. Basically, these messages indicate that all is well. However, if the primary event processor fails for some reason, the shadow event processor will take over the responsibility of interpreting and processing events.

For more information on running a shadow event processor, see *Shadow Event Processor* in the chapter “Introduction” in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

Remote Agent

On a UNIX machine, the remote agent is a temporary process started by the event processor to perform a specific task on a remote (client) machine. On a Windows machine, the remote agent is a Windows service running on a remote (client) machine that is directed by the event processor to perform specific tasks.

The remote agent starts the command specified for a given job, sends running and completion information about a task to the event server, then exits. If the remote agent is unable to transfer the information, it waits and tries again until it can successfully communicate with the database.

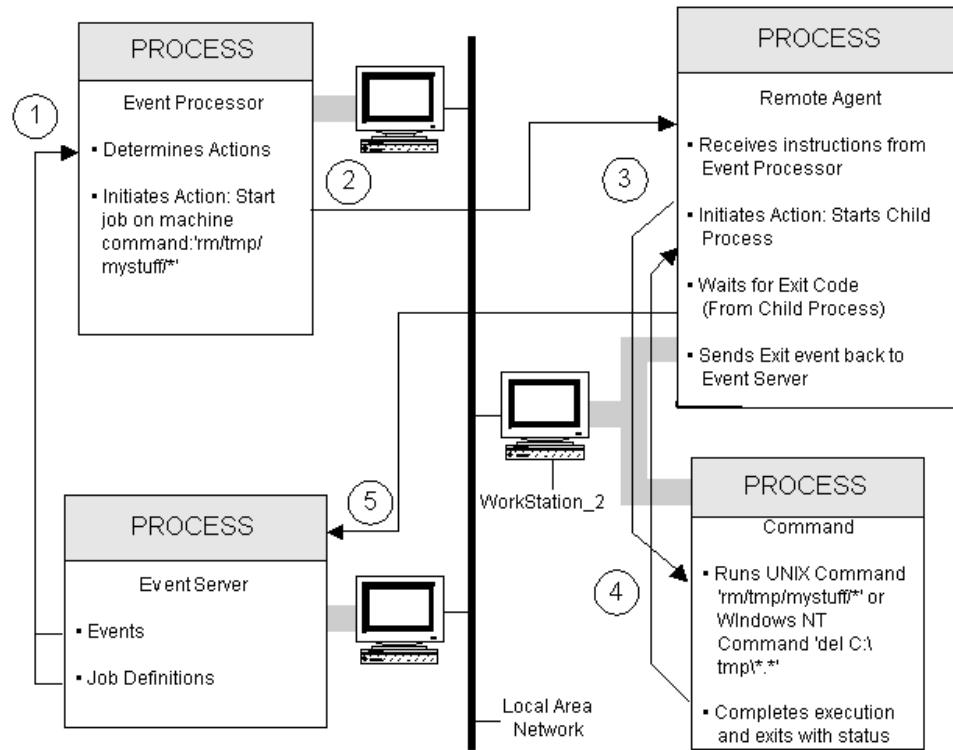
Example Scenario on UNIX

The example scenario in the following figure and the numbered explanations that follow it illustrate the interactions between the event server, the event processor, and remote agents.

In the example, the following UNIX command line is to be run on “WorkStation_2,” at the start date and time specified in the job definition:

```
rm /tmp/mystuff/*
```

Note: Understanding this example will help you answer many questions that may arise during your experiences with Unicenter AutoSys JM.



Note: In this example, the three primary components are shown running on different machines. Typically, the event processor and the event server run on the same machine.

Explanation

The following numbered steps explain the interactions in the example scenario:

1. From the event server, the event processor reads a new event, which is a STARTJOB event with a start time condition that has been met. Then the event processor reads the appropriate job definition from the database and, based on that definition, determines what action to take. In the example, it runs the `rm /tmp/mystuff/*` command on “WorkStation_2.”
2. The event processor communicates with the remote agent on “WorkStation_2.” As soon as the remote agent receives the instructions from the event processor, the connection between the two processes is dropped. After the connection is dropped, the job will run to completion, even if the event processor stops running.
3. The remote agent performs resource checks, such as ensuring that the minimum specified numbers of processes are available, then “forks” a child process that will actually run the specified command.
4. The command completes and exits, and the remote agent captures the command’s exit code.
5. The remote agent communicates the event (exit code, status, and so forth) directly to the event server. If the database is unavailable for any reason, the remote agent will go into a wait and resend cycle until it can deliver the message.

Only two processes need to be running: the event processor and the event server. When these two components are running, Unicenter AutoSys JM is fully operational. The remote agent is started on a client machine once per job. As soon as the job ends and the remote agent sends a completion event to the database, the remote agent exits.

Note: The remote agent is started on the client machine by the event processor talking to the internet demon (inetd) on the client machine. For this to happen, inetd must also be running. However, since UNIX is responsible for starting this demon, it is not considered a process.

Interface Components

To define, monitor, and report on jobs, you can use either the GUI or JIL. In addition, the Operator Console and its dialogs provide a sophisticated method of monitoring jobs in real time. This feature lets you view all jobs that are defined, whether or not they are currently active.

Machines

From a hardware perspective, the Unicenter AutoSys JM architecture is composed of the following two types of machines attached to a network:

- **Server Machine**

The server is the machine on which the event processor, the event server (database), or both, reside. In a basic configuration, both the event processor and the event server reside on the same machine.

- **Client Machine**

The client is the machine on which the remote agent software resides, and where jobs are to be run. A remote agent must be installed on the server machine, and it can also be installed on separate physical client machines.

Instance

An instance is one licensed version of Unicenter AutoSys JM software running as a server with one or more clients, on a single machine or on multiple machines. An instance is defined by the instance ID, which is a capitalized three-letter identifier defined by the \$AUTOSERV environment variable. An instance uses its own event server and event processor and operates independently of other instances.

You may want to install multiple Unicenter AutoSys JM instances. For example, you may want to have one instance for production and another for development. Multiple instances can run on the same machine, and can schedule jobs on the same machines without interfering or affecting the other instances.

Events

Unicenter AutoSys JM is completely event-driven; that is, for a job to be activated by the event processor, an event must occur on which the job depends. For example, a prerequisite job has completed running successfully or a required file has been received.

Events can come from a number of sources, including the following:

- Jobs changing states, such as starting, finishing successfully, and so forth.
- Internal verification agents, such as detected errors.
- Events sent with the sendevent command, sent from the Send Event dialog, the command line, or user applications.

As each event is processed, the event processor scans the database for jobs that are dependent on that event in some way. If the event satisfies another job's starting condition, that job is either started immediately, or if necessary, queued for the next qualified and available machine. The completion of one job can cause another job to be started, and in this way, jobs progress in a controlled sequence.

Alarms

Alarms are special events that notify operations personnel of situations requiring their attention. Alarms are integral to the automated use of Unicenter AutoSys JM. That is, jobs can be scheduled to run based on a number of conditions, but some facility is necessary for addressing incidents that require manual intervention. For example, a set of jobs could be dependent on the arrival of a file, and the file is long overdue. It is important that someone investigates the situation, make a decision, and resolve the problem.

These are some important aspects of alarms which are listed following:

- Alarms are informational only. Any action to be taken due to a problem is initiated by a separate action event.
- Alarms are system messages about a detected problem.
- Alarms are sent through the system as an event.

Alarms have special monitoring features to ensure they will be noticed. For more information about these features, see the chapters “The Operator Console” and “Monitoring and Reporting Jobs,” in this guide.

Utilities

To help you define, control, and report on jobs, Unicenter AutoSys JM has its own specification language called Job Information Language, or JIL, for defining jobs, machines, monitors, and reports. This language is processed by the `jil` command, which reads and interprets the JIL statements that you enter and then performs the appropriate actions, such as adding a new job definition to the database.

Unicenter AutoSys JM also provides a set of commands that run essential utility programs for defining, controlling, and reporting on jobs. For example, the `autorep` command lets you generate a variety of reports about job execution, and the `sendevent` command lets you manually control job processing.

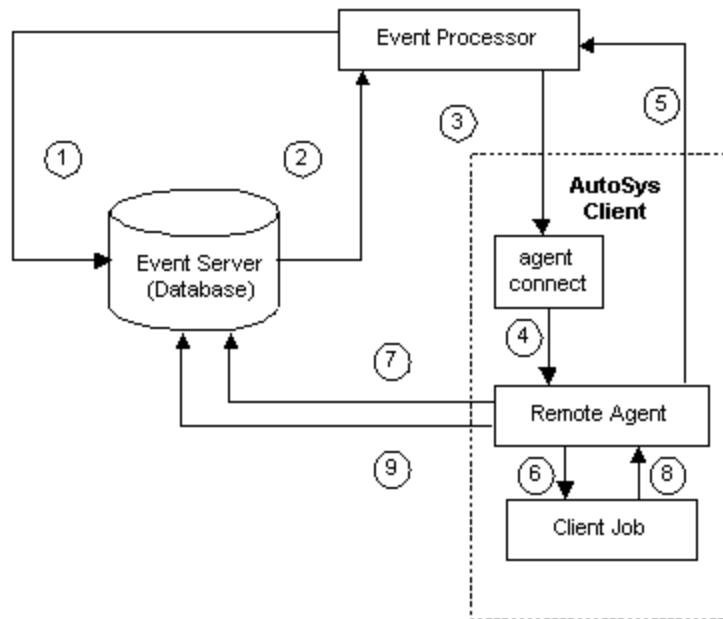
Additional utility programs are provided to assist you in troubleshooting, running monitors and browsers, and starting and stopping Unicenter AutoSys JM and its components. Unicenter AutoSys JM also provides a database maintenance utility that runs daily by default.

Basic Functionality

The diagram in the following figure and the numbered explanations that follow it illustrate how Unicenter AutoSys JM performs its most basic function—starting a job (that is, executing a command) on a client machine.

Working through this example will be very helpful for understanding how Unicenter AutoSys JM processes jobs.

Note: In the following figure, the major components are shown as separate entities. Typically, the event processor and the event server are installed on the same server machine (along with a required remote agent), and other remote agents are installed on separate client machines.



Explanation

1. The event processor scans the event server for the next event to process. If no event is ready, the event processor scans again in five seconds.
2. The event processor reads from the event server that an event is ready. If the event is a STARTJOB event, the job definition and attributes are retrieved from the Event Server, including the command and the pointer (full path name on the client machine) to the profile file to be used for the job. In addition, for jobs running on Windows machines, the event processor retrieves from the database the user IDs and passwords required to run the job on the client machine.
3. The event processor processes the event. If the event is a STARTJOB, the event processor attempts to establish a connection with the remote agent on the client machine, and passes the job attributes to the client machine.

The event processor sends a CHANGE_STATUS event marking in the event server that the job is in STARTING state.

4. On a UNIX machine, the inetd invokes the remote agent. On a Windows machine, the remote agent logs onto the machine as the user, defined as the job's owner, using the user IDs and passwords passed to it from the event processor.
5. The remote agent sends an acknowledgment back to the event processor indicating that it has received the job parameters. The socket connection is terminated. At this point, the event processor resumes scanning the event server database, looking for events to process.
6. The remote agent starts a process and executes the command in the job definition.
7. The remote agent issues a CHANGE_STATUS event marking in the event server that the job is in RUNNING state.
8. The client job process runs to completion, then returns an exit code to the remote agent and quits.

9. The remote agent sends the event server a CHANGE_STATUS event corresponding to the completion status of the job and passes back an exit code, using the communications facilities of the database.

If the return status is SUCCESS, the remote agent deletes the log file in its temporary file directory (usually tmp) on the client machine (if so specified in the AutoSys configuration file on UNIX or with the AutoSys Administrator on Windows).

The remote agent quits.

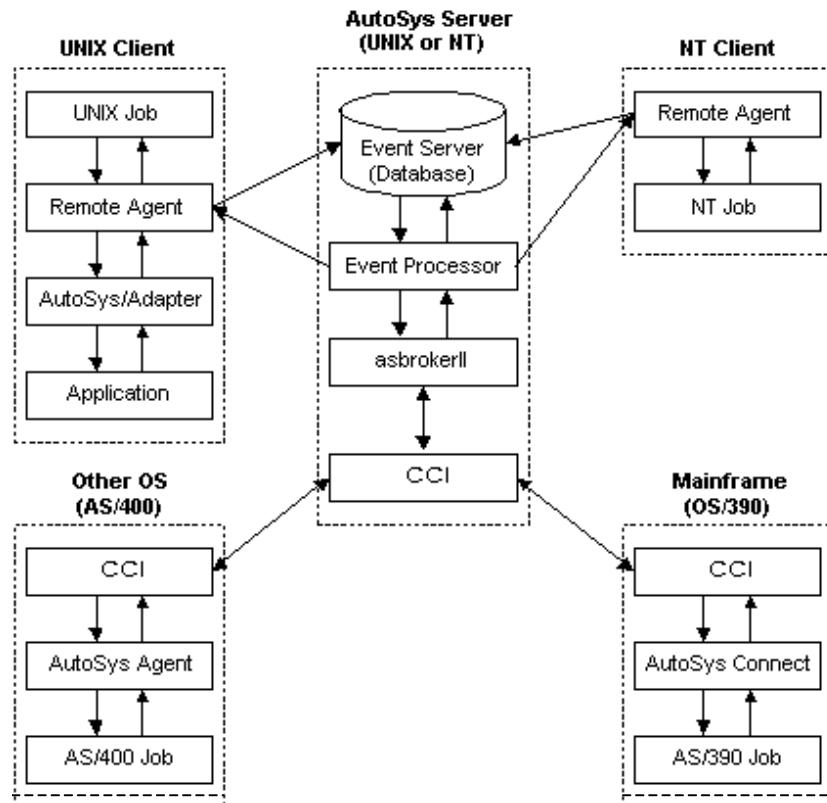
The event processor, which is scanning the event server, sees the process completion status, determines if there are dependent jobs, and evaluates the rest of the dependent jobs' starting conditions. For each job found whose remaining conditions are satisfied, the event processor sends a STARTJOB command to the event server, which it will then process in the next cycle.

Extending Functionality

You can extend Unicenter AutoSys JM jobs with the Unicenter AutoSys JM Connect and Unicenter AutoSys Agent integration components. Using cross-platform job dependency notation, you can define Unicenter AutoSys JM jobs to conditionally start based on the status of a Unicenter AutoSys JM Connect job running on a mainframe, and you can define Unicenter AutoSys JM Connect jobs to conditionally start based on the status of a Unicenter AutoSys JM job. You can also define Unicenter AutoSys JM jobs that will run on a Unicenter AutoSys Agent machine, if the Unicenter AutoSys Agent machine is defined to Unicenter AutoSys JM.

In addition, you can install various Unicenter AutoSys JM/Adapters. The application-specific Adapters lets you initiate, control, and report on the status of jobs related to an application using the sophisticated job scheduling capabilities of Unicenter AutoSys JM. Contact your Computer Associates International, Inc. (CA) sales representative for information on supported Unicenter AutoSys JM/Adapters.

The following figure illustrates an extended configuration that includes the Unicenter AutoSys JM connect and Unicenter AutoSys Agent integration components and a Unicenter AutoSys JM/ Adapter installation on a UNIX client.



Contacting Technical Support

You can contact us with any questions or problems you have. You will be directed to an experienced software engineer familiar with Unicenter AutoSys JM. You can contact CA Technical Support at esupport.ca.com.

To set up Unicenter AutoSys JM correctly, you should understand the security features that control where and by who certain secured activities can take place. If you are installing on both UNIX and Windows, you must understand how security is implemented on both systems.

For information about security on Windows, see the chapter Security in the *Unicenter AutoSys Job Management for Windows User Guide*.

Overview

Unicenter AutoSys JM is able to run in either eTrust™ secured mode or native security mode. External security (eTrust) can be enabled during the installation of the product, or later on by an authorized EXEC superuser. Once external security is enabled, the eTrust security package will be called to authorize the user, and to determine if they can turn off security in the product.

Note: While external security is enabled, native security is not enforced.

For more information on enabling eTrust security, see Security Control, in this chapter.

For more information on controlling the security setting with the Unicenter AutoSys Secure utility, see `autosys_secure` in the *Unicenter AutoSys Job Management for UNIX and Windows Reference Guide*.

Native Security

Unicenter AutoSys JM native security includes the following:

- System-level security
- Job-level security
- Superuser privileges
- UNIX and Windows file permissions (See Restricting Access to Jobs in this chapter.)

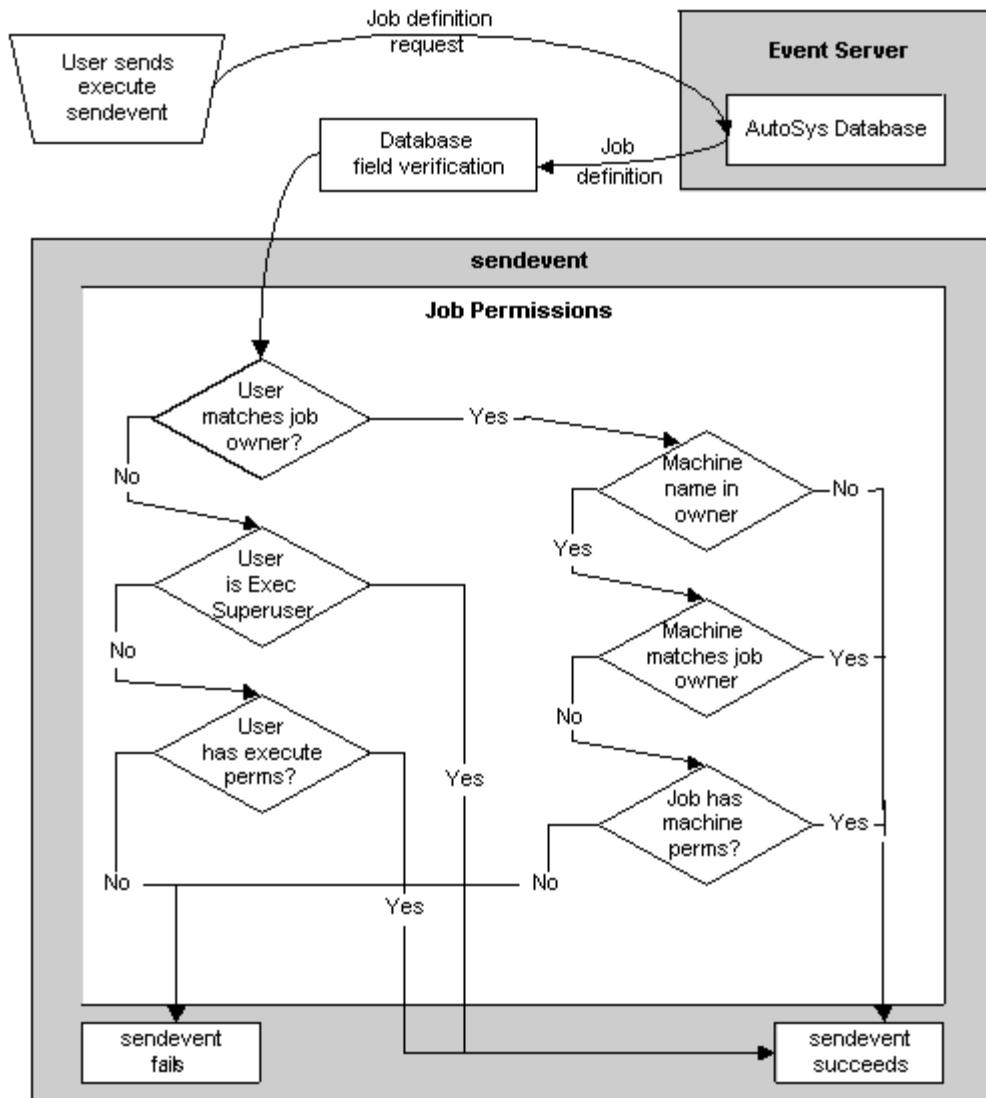
Security is initiated when either a user sends events that affect the running of a job or the event processor sends events that affect a job.

Security on Events Sent by Users

By using the sendevent command or the Send Event dialog, you can send execute events that affect the running of a job. The execute events that you can send, if you have the appropriate permissions are as follows:

Security Events	
CHANGE_PRIORITY	JOB_ON_HOLD
CHANGE_STATUS	JOB_ON_ICE
DELETEJOB	KILLJOB
FORCE_STARTJOB	SEND_SIGNAL
JOB_OFF_HOLD	STARTJOB
JOB_OFF_ICE	

If you start a job by sending an event, the job permissions are checked as shown in the following figure.



The previous figure shows how Unicenter AutoSys JM checks for the following when a user starts a job by sending an event:

1. Checks the database to determine if the job definition was tampered with. If so, the job definition is invalid, and the job is not run.
2. Does the user match the owner as indicated in the job definition?
3. Is the user the EXEC superuser as defined with `autosys_secure`?

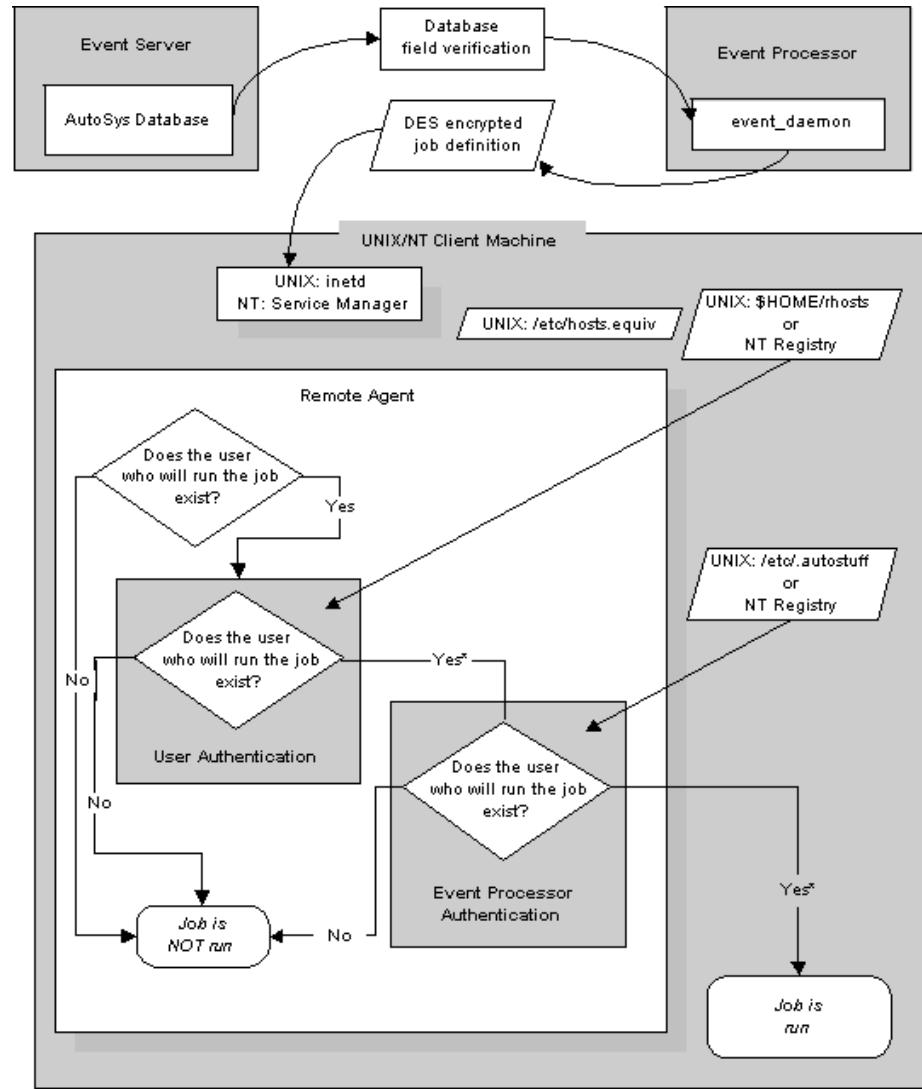
4. Does the user have job execute permissions as indicated in the job definition?
5. Is there a machine name in the owner value of the job definition? The EDIT superuser can remove this portion of the owner.
6. Does the machine portion of the user logon match the job owner machine portion?
7. Does the job have machine permission as indicated by the job definition?

Security on Events Sent by the Event Processor

In addition to sending execute events on jobs; you can schedule jobs to start at certain times or under certain conditions. When a job is scheduled to start automatically, permissions are checked on the remote agent machine on which the job is to run.

The event processor scans the event server for any jobs with starting conditions that have been met. When the starting conditions for a job are met, the event processor sends a STARTJOB event to the designated remote agent machine.

The following figure shows the permissions and security checks that occur on a UNIX machine before a job is allowed to start on the machine.



Note: In the figure, an asterisk (*) indicates checks that are made only if the specific method of remote authentication is enabled (see Remote Agent Authentication in this chapter).

The previous figure shows how Unicenter AutoSys JM checks for the following when the event processor sends a STARTJOB event to a remote agent machine:

1. Checks the database to determine if the job definition was tampered with. If so, the job definition is invalid, and the job is not run.
2. Checks the DES encrypted job definition to determine if the event processor can connect to the remote agent machine.
3. Does the user who is defined as the job owner (*user@machine*) have a logon account on the remote agent machine?
4. If user authentication is enabled, is the user a trusted user (as defined in the /etc/hosts.equiv and \$HOME/.rhosts files)?
5. If event processor authentication is enabled, does the requesting event processor have permission to run jobs on this remote agent machine?

Note: The EDIT superuser can enable remote authentication by using the autosys_secure utility.

System-Level Security

The security scheme prevents unauthorized access to facilities, which in turn prevents unauthorized access to jobs. The following features handle system security:

- Database field verification
- Job definition encryption
- Remote agent authentication
- User and database administrator passwords

Note: On UNIX, the database field and control string encryption features provide a level of security comparable to the security provided in the native UNIX environment.

Database Field Verification

To secure the database, Unicenter AutoSys JM not only encrypts some fields specified in a job definition, but also generates a checksum from fields in the job definition, and stores the checksum in the database. Whenever a job is accessed, its checksum is regenerated and compared to the one in the database. If the checksums are different, this indicates that someone tampered with the job definition in the database, probably by using an SQL command. In this case, the job is disabled and cannot be executed.

To re-enable a disabled job, the owner or the edit superuser must access the definition and re-save it, by using either the JIL update_job sub-command or the Job Definition dialog.

Job Definition Encryption

To secure the remote agent from unauthorized access, the event processor encrypts the information in a job definition sent over the socket to the remote agent. The remote agent then decrypts the job information and continues to process the job. If the remote agent receives any job information from the event processor that it does not recognize, it issues an error message and will not process the job.

Remote Agent Authentication

The two remote agent authentication methods Unicenter AutoSys JM provides are:

- User authentication
- Event processor authentication

By default, both user authentication and event processor authentication are disabled. The edit superuser must enable them by using the `autosys_secure` command.

User Authentication

This remote authentication method uses UNIX `ruserok()` authentication to verify that a user has permission to start a job on a client machine. It accomplishes this by telling the client's remote agent to make the `ruserok()` UNIX system call to check the client machine's `/etc/hosts.equiv` and the user's `.rhosts` file to validate that the requesting user is registered in that environment. This function call performs a local verification, and it is not related in any way to `rshd` or `rlogind`. To activate this type of remote authentication, use the `autosys_secure` command.

The `hosts.equiv` or `.rhosts` file entries must match the job owner and machine name field exactly. For example, if the owner is `tarzan@jungle`, the `hosts.equiv` or `.rhosts` file must contain `jungle`. Similarly, if the owner is `tarzan@jungle.vine.com`, the `hosts.equiv` or `.rhosts` file must contain `jungle.vine.com`. If they do not match, jobs will fail to run on that machine when `ruserok()` remote authentication is in use.

For information on enabling this type of remote authentication, see `autosys_secure` in the chapter *Commands* in the *Unicenter AutoSys Job Management Reference Guide*.

Event Processor Authentication

When event processor authentication is enabled, the remote agent verifies that it has permission to process requests from the requesting event processor before starting each job. It does this by reading the /etc/.autostuff file on the machine on which the remote agent is running. For information on enabling event processor authentication, see `autosys_secure` in the chapter Commands in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

Note: Before enabling event processor authentication, you must set up and properly configure the /etc/.autostuff file on every client machine that will participate in this authentication method, as described in Configuring Remote Authentication in the chapter Configuring, in this guide.

User and Database Administrator Passwords

When you install Unicenter AutoSys JM and configure your database, an `autosys` user is added to the database with a password set to `autosys`. The `autosys` user is the owner of the database and can make changes to specific information in the database. To enhance system security, we recommend that you change the `autosys` user password with the `autosys_secure` command.

When you install with bundled Sybase, the database system administrator ID is `sa`, and the password is `sysadmin`. To enhance security, we recommend that you change the system administrator password by using the `xql` utility.

You must supply the `autosys` and `sa` user IDs and passwords when you use several utilities. For example, when using the `xql` utility to query the database, you must know both the `autosys` user password and the `sa` system administrator password.

For information on changing the `autosys` user password see `autosys_secure`, and for information on changing the `sa` password and querying the database, see `xql` in the chapter Commands in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

Job-Level Security

The security scheme provides individuals and groups of users with edit and execute permissions on a job-by-job basis. For jobs running on UNIX, Unicenter AutoSys JM supports owner, group, and world edit and execute permissions. For jobs running on Windows, Unicenter AutoSys JM supports owner and world edit and execute permissions.

By default, only the user logged on as the owner of a job can edit or execute a job. The owner can extend permissions to other users and other machines, as described in the following sections.

Job Ownership

By default, the owner of a job is the user who defines that job on a particular machine. When a user defines a job on UNIX, the user ID is retrieved from the UNIX environment and attached to the job in the form of user@machine. The owner is defined by the owner job attribute. By default, only the owner can edit and execute the job.

The user@machine combination must have execute permission for any command specified in a job on the machine where the job command is to run. The job owner must also have permission to access any device, resource, and so forth that the command needs to access. For this process to work, the job owner must have the appropriate system permissions.

The owner's umask write permission is used as the default edit permission of the job, and the umask execute permission is used as the default execute permission of the job.

If a job is run on a Windows client machine, the edit superuser must have entered the valid Windows user ID and password for the owner into the database. For more information about the edit superuser, see Edit Superuser in this chapter.

User Types

Like UNIX, Unicenter AutoSys JM uses the notion of three types of users for any jobs as follows:

Owner

The user who created the job.

Group

Any user who is in the same primary group as the owner.

World

Every user.

Unicenter AutoSys JM uses the UNIX user ID (uid) and group ID (gid) of a job's owner to control the following:

- Who can edit, override, or delete a job definition.
- Who can execute the UNIX command specified in a job.

The owner of a job can let other users edit and execute the job by setting the permissions in the job definition (discussed in the following section).

Permission Types

By default, only the owner has edit and execute permissions on a job, and all edit and execute permissions are valid only on the machine on which the job was defined. However, the owner can grant different types of permissions when defining a job.

Similar to UNIX, Unicenter AutoSys JM associates different types of permissions with each job. Every job has the following permission types:

Edit

Users can edit, override, or delete a job definition.

Execute

Users can send an execute event that affects the running of a job by using the sendevent command or the Send Event dialog. For a list of the execute events that users can send, see Security on Events Sent by Users in this chapter.

Machine

Users logged onto a machine other than the one on which a job was created can edit or execute the job.

Note: In order for a job to run on a machine other than the one on which the job was defined, the owner of that job must have an account on that machine.

Granting Permissions

The owner of a job cannot override his or her ownership designation; only the edit superuser has the authority to change the owner job attribute. However, the owner can grant other users edit and execute permissions for a job by using the GUI or JIL to set the permission job attribute in the job definition.

The following table shows the permissions that you can set by using JIL or the Permission toggle buttons on the Job Definitions Advanced Features dialog.

GUI	JIL	Meaning
Group Execute	gx	Users assigned to the job owner's primary group can execute the job if logged onto the machine where the job was created (the machine specified in the owner attribute, that is, user@machine).
Group Edit	ge	Users assigned to the job owner's primary group can edit the job if logged onto the machine where the job was created (the machine specified in the owner attribute, that is, user@machine).
All Hosts Execute	mx	Users, regardless of the machine logged onto, can execute the job (otherwise, the user must be logged onto the machine specified in the owner attribute, that is, user@machine).
All Hosts Edit	me	Users, regardless of the machine logged onto, can edit the job (otherwise, the user must be logged onto the machine specified in the owner attribute, that is, user@machine).
World Execute	wx	Users can execute the job if logged onto the machine where the job was created (the machine specified in the owner attribute, that is, user@machine).

GUI	JIL	Meaning
World Edit	we	Users can edit the job if logged onto the machine where the job was created (the machine specified in the owner attribute, that is, user@machine).

Note: A job and the command it executes will always run as the user specified in the owner attribute of the job definition. Execute permissions determine who can execute events against the job, but not who the job runs as. Even if World Execute permissions are granted, the job will still run as the user.

Job Permissions and Windows

If you are defining jobs and running them on different operating systems, keep the following in mind:

- When defining a job to run on a Windows machine, you can set group permissions, but they will be ignored. Group permissions will be used if a job is edited or executed on a UNIX machine.
- When editing a job from a Windows machine, the group edit permission is ignored. In this case, the user editing the job must be the owner of the job, or World Edit permissions must be specified for the job.
- When executing a job from a Windows machine, the group execute permission is ignored. In this case, the user executing the job must be the owner of the job, or World Execute permissions must be specified for the job.

Security Control

External security is controlled by a setting in the Unicenter AutoSys JM database. You can turn external security on or off by using the autosys_secure binary. For more information on autosys_secure see the *Unicenter AutoSys Job Management Reference Guide*.

Superuser Privileges

Unicenter AutoSys JM provides you the ability to create more than one EDIT or EXEC Superuser. You can define these superusers by using the `autosys_secure` command. For information about defining the edit and exec superusers, see the chapters Server Installation for Sybase or Server Installation for Oracle in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

Edit Superuser

Only the edit superuser has permission to do the following:

- Edit or delete any job regardless of its owner or its permissions.
- Change the owner attribute of a job.
- Change the database password, change the remote authentication method, and add and change Windows user IDs and passwords by using the `autosys_secure` command.

The edit superuser can override user authentication (if enabled) on a job-by-job basis by changing the owner of the job from the form `user@machine` to the form `user`. User authentication of the job at execution time is not performed on the client machine. For more information about changing the job owner, see owner attribute in the chapter JIL/GUI Job Definitions in the *Unicenter AutoSys Job Management Reference Guide*.

Note: The purpose of the `user@machine` form is to prevent users from running jobs on machines where they do not have the appropriate permission. For example, `root@machine` prevents root on any machine from running root jobs on all machines.

The edit superuser must enter valid Windows user IDs and passwords into the database. These user IDs and passwords are required to log onto and run jobs on Windows client machines. When a remote agent runs a job on a machine, it logs on as the user defined in the owner attribute for the job. To do this, the event processor retrieves encrypted versions of the IDs and passwords for the user@host_or_domain and the user@machine from the event server and passes them to the remote agent. For information about entering and changing Windows user IDs and passwords, see `autosys_secure` in the chapter Commands in the *Unicenter AutoSys Job Management Reference Guide*.

Note: Any user who knows an existing user ID and password can change that password or delete that user and password.

Exec Superuser

Only the exec superuser has permission to do the following:

- Issue commands that affect the running or the state of any job, either using the `sendevent` command or the Send Event dialog.
- Enable eTrust Access Control
- Set the Subscriber Authentication Security Word
- Stop the event processor by issuing the following command:

```
sendevent -E STOP_DEMON
```

Note: Exec superuser privileges are usually granted to the night operator.

Restricting Access to Jobs

Using the UNIX chmod command, you can change permissions on many files to control which users can view jobs, execute jobs, edit jobs, and change calendars.

First, you must ensure that only authorized users can change permissions on the files and directories in the directory structure.

Then, you should determine what level of security you want, for example:

- Only authorized users can use Unicenter AutoSys JM.
- Any user can view jobs and reports about jobs, such as using autorep to see the status of a job, but only authorized users can create jobs and calendars or make changes to them.

If you want only authorized users to access Unicenter AutoSys JM, ensure that only those users have execute permissions on the files in the bin directory.

If you want all users to view reports about jobs, but only authorized users to create and edit jobs and calendars, ensure that the following files in the %AUTOSYS%/bin directory are executable only by the authorized users. This will also prevent unauthorized users from making changes to the configuration.

Secure the following files:

File Names

archive_events	DBMaint	sendevent
autocal	dbspace	timescape*
autocal_asc	dbstatistics	xql
autocons*	hostscape*	zaplus
autotimezone	jil	zql
clean_files	jobscape*	

Note: An asterisk (*) indicates files that can be executable by all users as long as sendevent and jil are not executable. This lets users use the GUI to view job states, but does not let them add new jobs or calendars or start jobs (even if the job has world execute permissions).

You should also protect the files in the \$AUTOUSER directory from modification by ensuring that only users authorized to change the configuration have write permission on the files. Read permission is necessary to source the environment files.

Remote Agent Security

In the auto.profile file for the remote agent machine, you can specify a list of users whose jobs are prohibited from running on that machine. For information on this, see Client-side Security in the chapter Configuring, in this guide.

eTrust Access Control

Unicenter AutoSys JM provides you with Asset Level Security, if selected during installation. This security is accomplished through integration with eTrust™ Access Control (eTrust AC). All GUI applications and all Command Line Interfaces will have call outs to security. User-defined classes within eTrust AC will be used to govern what types of resources can be controlled by which users.

Since the event processor and remote agent will not enforce security, policy changes will not affect resources which were entered into the database. For example; if the security administrator withdraws a user's permission to create jobs, Unicenter AutoSys JM will continue to run jobs created by the user before the change.

Note: Policy changes can only be made by users who have been assigned eTrust administrative rights and only from host machines that have the proper eTrust access rights. The primary eTrust administrator and administrator host is determined during the eTrust Server installation portion of the Unicenter AutoSys JM installation. Additional eTrust administrators and administrative hosts can be defined by the primary administrator using the autosys_secure binary menu item [7] followed by items [3] and [4] respectively. The same tasks can alternatively be accomplished through eTrust AC using the eTrust Policy Manager on Windows or the selang command line utility. For more information on selang, see the *eTrust Access Control for UNIX Reference Guide*.

If you turn on *eTrust AC* security, the job-level security and superuser security supported in native mode will no longer be adhered to.

Note: Wherever Unicenter AutoSys JM binaries are installed, a local *eTrust* database will be created called *seosdb*. This database will subscribe to the machine where the *eTrust Local Policy Model Database* (*pmdb*) was created to ensure that security policies are pushed out to each machine. Any security calls made by these binaries will go against the local *seosdb*, rather than a remote security database, to avoid unnecessary network traffic. The *seosdb* can be subscribed to the parent *pmdb* either during the *eTrust* client installation portion of the Unicenter AutoSys JM installation or by an *eTrust* administrator from an administrative host through the *autosys_secure* binary menu item [7] sub-item [5]. The same task can alternatively be accomplished through *eTrust AC* using the *eTrust Policy Manager* on Windows or the *selang* command line utility. For more information on *selang*, see the *eTrust Access Control for UNIX Reference Guide*.

If *eTrust* security is enabled, you must establish a subscriber authentication security word before any secured executables will work properly. Before establishing your security word is a good opportunity to define your enterprise security policy since Unicenter AutoSys JM is effectively locked down until you establish the security word.

When you are ready to establish your security word, run *autosys_secure* as a user and from a host that are authorized to administer the *eTrust pmdb*. Choose menu item 7 followed by item 2. You will be prompted for your security word. The only time you will ever be prompted for this word again is if you decide to change your security word.

Note: To provide cross-platform compatibility, the security word is stored in the *eTrust* database and the Unicenter AutoSys JM database in upper case. This renders the security word case-insensitive. For example, if you create the security word 'my_word' and then decide to change it, when you are prompted for the existing security word, you could successfully enter 'My_Word' or 'my_WORD'; and Unicenter AutoSys JM will see them as the same.

If you have reason to believe that your security word has been compromised, you should change it using *autosys_secure*. With that information it would be possible for a malicious user to setup a local *eTrust* policy and circumvent Unicenter AutoSys JM security.

The security word you provide is stored in the Unicenter AutoSys JM database and the *eTrust* database. Before checking security, all secured Unicenter AutoSys JM executables will read the security word from the Unicenter AutoSys JM database and compare it to the security word in the *eTrust* database. It will only be present in *eTrust* if the local installation is a valid subscriber to the enterprise security policy. If there are problems verifying the security word, access to secured assets will be denied.

Note: The *eTrust AC* audit log may have failed to check the security word resource each time you run a secured binary. This is expected behavior. If you would rather not see these failures, you can do one of two things.

- You can create a filter that will not include these entries. For more information see the section Audit Filters in the *eTrust Access Control Administrator Guide*.
- You can change your user resources so that these failures will not be logged. By default, all users are configured to cause log entries to be generated on access failures (regardless of which resource the failure occurred with). However, the security word resource has been created to not cause log entries to be generated on failure (since that is expected behavior in this case). You can change (or create) your users to not create an audit log entry when a failure occurs. This leaves it up to the individual resources to create failure entries in the audit log (the default behavior for resources). You can change the audit rules using either selang or the Policy Manager GUI. For more information on configuring audit rules see the *eTrust Access Control Administrator Guide*.

You can globally enable or disable *eTrust* using `autosys_secure`. In order to disable *eTrust*, you must be granted execute access to the `SECADM` resource.

Policy Manager

All modifications to security access of any Unicenter AutoSys resource can easily be done through the *eTrust Policy Manager* on Windows. You can also modify security access using the `selang` command line utility. For more information on `selang`, see the *eTrust Access Control for UNIX Reference Guide*. The *eTrust Policy Manager* lets you modify and set security levels for all user-defined classes provided by Unicenter AutoSys JM.

Security Access

The Policy Manager is used to modify security.

To modify security do the following:

1. Click Start, Programs, Computer Associates, *eTrust*, Access Control, Policy Manager.

The *eTrust* Policy Manager appears.

2. Select File, Connect.
3. Click Add (Insert) in the Host Selection dialog.
4. Enter the hostname where the *autosys* PMDB was installed.
5. Check the Connect to PMDB check box and enter *autosys* as the PMDB name.

A new entry appears in the Host Selection dialog

6. Select the new entry, and click OK.
7. Once connected, click Resources in the left program bar Access Control.
8. To view *Unicenter AutoSys JM* security classes, expand the User-Defined classes' folder.

Selecting a class will list the available policies governing access to a specific resource.

Disable Security

To access the Disable Security option in the *autosys_secure* command.

For more information about *autosys_secure*, see the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

Controlling the Event Processor

If eTrust security has been enabled through autosys_secure, you, by default, are prevented from shutting down the event processor.

Asset-Level Security

If selected during installation, Unicenter AutoSys JM provides you asset-level security through integration with eTrust AC version 5.2. All Unicenter AutoSys JM GUI applications and Command Line Interfaces will call out to the security engine bundled with the installation program if eTrust AC is currently enabled. User-defined classes within eTrust will be used to govern what types of resources can be controlled by which users.

For more information on eTrust AC see the *eTrust Access Control for UNIX User Guide*.

Since the event processor and remote agent will not enforce security, policy changes will not affect resources which were entered into the database under the previous policy. For example, if the security administrator withdraws a user's permission to create jobs, Unicenter AutoSys JM will continue to run jobs the user created before the change.

During the installation of eTrust AC, a (PMDB) was created called autosys, on what will be considered the master security server. On the master security server, eTrust AC will subscribe a client database to the autosys PMDB. The install will ask for the users that will be defined as administrators to the eTrust database, but will not import existing OS users into the eTrust AC database.

Unicenter AutoSys JM will be able to run in either eTrust secured mode or regular mode. External security can be enabled during the installation of the product, or later on by an authorized EXEC super user. Once security is enabled, the external security package will be called to authorize the user to determine if they can turn off security in the product.

Important! When working in a mixed environment (UNIX, Windows) and using eTrust AC security only, you must be vigilant as to how resources are added to an eTrust AC database. Since UNIX is case-sensitive and Windows is case-preserving, it is easy to enter a name with the wrong case which will then not be correctly recognized.

For example, you may want to create a user 'Administrator' that you will allow to administer the 'autosys' PMDB from a Windows machine. If you create the user as 'administrator' (lowercase 'a') and then try to run the policy manager from a Windows box where you are logged in as 'Administrator' you will be denied access. This can be confusing because Windows will let you log in to the 'Administrator' account as 'administrator.' The key is that the user in the PMDB must follow the case as it is preserved on the Windows machine.

For more information on enabling security see Security control, in this chapter. For more information on case-sensitivity, see the *eTrust Access Control for UNIX Reference Guide*.

eTrust Resource Classes

To secure the product, a set of classes will be defined that pertain to Unicenter AutoSys JM. These classes are used to control access to jobs, calendars, cycles, machines, global variables, and the owner field of a job. There are also classes to prevent unauthorized users from starting or shutting down Unicenter AutoSys JM, disabling security, and to prevent unauthorized users from accessing Unicenter AutoSys JM Interface for Java GUI.

Unicenter AutoSys JM will use the following eTrust User-Defined Classes. These classes will be created in the eTrust database and the PMDB autosys. The classes are eTrust enabled and will make security callouts prior to performing an action on a specified object.

as-job	as-calendar	as-machine
as-gvar	as-owner	as-control
as-view	as-list	

The name of each eTrust resource will be the name of the corresponding Unicenter AutoSys JM object, a period, and the name of the instance.

For example, Unicenter AutoSys JM will query eTrust about
as-job *payroll*.ACE
when a user tries to update job *payroll* in instance ACE.

Note: The security administrator must use the object. instance convention when creating policies. You can use wildcards to create policies which apply to multiple objects among different instances.

For more information on Resource Classes, see the *eTrust Access Control for UNIX Reference Guide*.

eTrust Access Modes

Unicenter AutoSys JM will utilize the following access modes on each of the various resource classes. The use of these access modes is explained in more detail with the description of each class.

- READ
- CREATE
- DELETE
- EXECUTE
- WRITE

as-job Class

The as-job class will control access to job objects. All of the eTrust access modes will be applied to this object.

READ

Prevents users from being able to view jobs or their contents.

Binary	Security Checkpoints
autocons (Job Activity Console)	Job list, contents of dependent jobs, contents of starting conditions if as-list\AUTOCONS denied. Regardless of as-list, as-job is checked before displaying job details.
autocal	If as-list\AUTOCAL denied
autocons (Alarm Manager)	If as-list\JOBDEF denied
autorep	If as-list\AUTOREP denied, otherwise: -J job, -q
autosc (Job Definition)	When searching for jobs, if as-list\JOBDEF denied before allowing a job to be opened
autostatad	-J job if as-list\AUTOSTAT denied
autostatus	-J job
hostscape	If as-list\XPERT denied
job_depends	-J job, if as-list\JOBDEP denied
jobscape	If as-list\XPERT denied
monbro	If as-list\MONBRO denied
timescape	If as-list\XPERT denied

CREATE

Prevents users from creating a job object.

Binary	Security Checkpoints
autosc (Job Definition)	On save (if new job)
jil	update_job

DELETE

Prevents users from deleting jobs, including deleting the job using the sendevent command.

Binary	Security Checkpoints
autosc (Job Definition)	On delete
jil	delete_job
sendevent	-e DELETEJOB

EXECUTE

Controls whether a user is allowed to issue a sendevent against the job object.

Binary	Security Checkpoints
sendevent	-e STARTJOB -e KILLJOB -e FORCE_STARTJOB -e JOB_ON_ICE -e JOB_OFF_ICE -e JOB_ON_HOLD -e JOB_OFF_HOLD -e COMMENT (not global)

WRITE

Prevents unauthorized users from updating an existing job object.

Binary	Security Checkpoints
autosc (Job Definition)	On save (if existing job)
jil	insert_job
sendevent	-e CHANGE_PRIORITY

as-calendar Class

The as-calendar class will control access to calendar objects.

READ

Prevents users from being able to view calendars or their contents.

Binary	Security Checkpoints
autocal	Ifile – open, if as-list\AUTOCAL denied
autosc (Job Definition)	If as-list\JOBDEF denied
autocal_asc	PRINT

CREATE

Prevents users from creating a calendar object.

Binary	Security Checkpoints
autocal	File – save, File – new, File -- rename
autocal_asc	ADD (new)

DELETE

Prevents users from deleting a calendar.

Binary	Security Checkpoints
autocal	File – delete, File -- rename
autocal_asc	DELETE

EXECUTE

Controls whether a user is allowed to specify the given calendar to run or to exclude within a job object.

Binary	Security Checkpoints
autosc (Job Definition)	On save (for run calendar and exclude calendar)
jil	run_calendar, exclude_calendar

WRITE

Prevents unauthorized users from updating existing calendar objects.

Binary	Security Checkpoints
autocal	File – save as
autocal_asc	ADD (existing)

as-machine Class

The as-machine class will control access to machine objects. This will control who can do what to the machine object including whether or not it can be used by a user in a job definition. All of the eTrust access modes will be applied to this object.

READ

Prevents users from being able to view machines or their contents.

Binary	Security Checkpoints
autocons (Job Activity Console)	Machine list in 'Job Selection,' if as-list\AUTOCNS denied
autorep	-m machine, if as-list\AUTOREP denied
hostscape	If as-list\XPERT denied
jobscape	If as-list\XPERT denied

CREATE

Prevents users from creating machine objects.

Binary	Security Checkpoints
jil	insert_machine

DELETE

Prevents users from deleting machine objects.

Binary	Security Checkpoints
jil	delete_machine

EXECUTE

Unless authorized, EXEC controls whether a user is allowed to specify that machine inside a job object.

Binary	Security Checkpoints
autosc (Job Definition)	On save (machine field)
jil	machine
sendevent	-e STARTJOB -e KILLJOB -e FORCE_STARTJOB -e JOB_ON_ICE -e JOB_OFF_ICE -e JOB_ON_HOLD -e JOB_OFF_HOLD -e COMMENT (not global)

as-gvar Class

The as-gvar class will control access to global variable objects. Since this object is only controlled through the sendevent binary, the access modes will be checked during sendevent execution. All of the eTrust access modes will be applied to this object.

READ

Prevents users from being able to view specific global variable objects.

Binary	Security Checkpoints
autorep	-g variable
autostatus	-g variable

CREATE

Prevents users from creating specific global variable objects.

Binary	Security Checkpoints
sendevent	-g (new variable)

DELETE

Prevents users from deleting specific global variable objects.

Binary	Security Checkpoints
sendevent	-g variable=DELETE

EXECUTE

Prevents users from using sendevent all together against global variables.

Binary	Security Checkpoints
sendevent	-e SET_GLOBAL, all-g options

WRITE

Prevents unauthorized users from updating an existing specific global variable objects.

Binary	Security Checkpoints
sendevent	-g (existing variable)

as-owner Class

The as-owner class will control access to the job owner field in the job object. This will be used to control what owner can be specified in a job definition. For example, when a new job is created, by default the user ID of the job creator is automatically used. However, when a different user is to be used, security will be called to determine if the owner specified is allowed to be used by the current user.

EXECUTE

Prevents users from using unauthorized user-id's.

Binary	Security Checkpoints
autosc (Job Definition)	On save (owner field)
jil	owner

as-control Class

The as-control class will control access to critical services within Unicenter AutoSys JM.

EXECUTE

Control critical resources through the following:

Binary	Security Checkpoints
sendevent	-e STOP_DEMON

STOP_DEMON

Controls who can stop the event processor. Applies to both the sendevent command, and the service control manager on Windows.

Note: If eTrust security has been enabled then by default, the user will be prevented from stopping the event processor from the Service Control Manager and can only use sendevent.

SECADM

For Internal Use only.

WEBLOG

For Internal Use only.

WEBADM

For Internal Use only.

as-view Class

The as-view class will control access to the various views defined in the Unicenter AutoSys JM Web Interface GUIs, including preventing graphical representations of certain jobs.

Note: For performance reasons, it is not feasible to call security for each individual object that is to be displayed on the web browser.

READ

Prevents users from being able to bring up a particular view, preventing access to jobs they are not authorized to see.

CREATE

Prevents users from creating views defined and maintained by Unicenter AutoSys JM Web Interface.

DELETE

Prevents users from deleting views defined and maintained by Unicenter AutoSys JM Web Interface.

WRITE

Prevents unauthorized users from updating views defined and maintained by Unicenter AutoSys JM Web Interface.

as-list Class

The as-list class will control telling programs to bypass security for read-only operation, as in autocons or autorep, where the information displayed does not constitute a security violation.

Notes:

By using the default of this class Unicenter AutoSys JM will not incur the tremendous overhead of issuing a security call for each individual line item displayed.

This class is provided for those users that do not believe that status or report type functions that do not display the detail of the asset warrant a security call on each object.

In an environment where there are thousands of jobs, issuing a security call against each individual job, just to see status type or summary, may cause unnecessary security overhead.

READ

Control security bypass through the following:

AUTOREP

Controls read access for the autorep program. This value will be ignored for any autorep report that specifies the *-q* option.

Binary	Security Checkpoints
autorep	-m ALL
	-J ALL
	-J box
	-g ALL

AUTOCONS

Controls read access to the console type programs including the Unicenter AutoSys JM Web Interface GUI.

Binary	Security Checkpoints
autocons (Job Activity Console)	At startup

AUTOCAL

Controls read access within the Calendar Definition GUI.

Binary	Security Checkpoints
autocal	File – open
	Job Definition Reference List
	Calendar Selection list in Term Calendar Viewer

AUTOSTAT

Controls read access within the autostatad binary.

Binary	Security Checkpoints
autostatad	-J %

MONBRO

Controls read access to the monitor/browser GUI.

Binary	Security Checkpoints
monbro	Event related to secured jobs

JOBDEP

Controls read access to the job_depends GUI.

Binary	Security Checkpoints
jobdep	-c -J ALL -c -J % [-t, -d] % [-t, -d] ALL [-t, -d] box

XPERT

Controls read access to the XPERT GUI.

Binary	Security Checkpoints
hostscape	Lookup Matches dialog, Machine list in Job Selection dialog, Machines in main view, Jobs listed in each machine in the main view
jobscape	Lookup Matches dialog, Machine list in Job Selection Dialog, Jobs listed in main view
timescape	Lookup Matches dialog, Machine list in Job Selection Dialog, Jobs listed in main view

Security Enabled Applications**Example 1**

If read access to the autocons resource belonging to the as-list class has been granted to the Unicenter AutoSys JM *instance*, then the individual read access checks for each job is ignored and the entire list of jobs is displayed.

To disable list access for autocons, open the eTrust Policy Manager to the as-list class, and select AUTOCONS*.

Note: The owner of the AUTOCONS* resource has been set to nobody. This is necessary as all security checks will automatically pass if called by the owner.

Before re-launching autocons, read access to the job 'goodtest' will be disabled. From the Policy Manager a resource has now been created for any job beginning with good. All access has been disabled for these jobs.

Now when autocons starts up, a warning message will be displayed indicating that as-list access failed, and a read access check will be performed for each job before displaying it.

Note: The resource that Unicenter AutoSys JM used to check for read access is AUTOCONS.INSTANCE.

Example 2

A similar behavior is exhibited when running autorep from the command prompt.

First list access will be checked for the AUTOREP resource in the as-list class. If access is granted the requested jobs will be reported without performing read access checks.

However, if list access has been denied, then read access checks will be performed for each job before displaying job information.

Notes:

If a job in a box fails read access, but the box passes read access, the failed job will not be displayed.

If a box job fails read access none of the jobs within the box, and the box will not be displayed.

Security Call Logic

This section walks through the logical flow of creating, updating, and deleting an object.

Creating an Object

The following represents a logical flow for the creation of any object:

1. Call security to validate user has authority to assign the object in the specified security group by calling security with execute permission on the security group.
2. Call security to validate user can create the object by passing in the security group name and specifying create authority.
3. For Job objects only — call security again and validate the owner field using an asset of as-owner and a permission of execute.
4. For Job only — call security passing in the security group of the machine with an execute permission if that machine can be used.

Updating an Object

The following represents a logical flow for updating any object:

1. Call security to validate user has authority to update objects in the security group using the original security group of the object.
2. If the security group is being modified, call security to ensure that the user has update authority to objects in the security group.
3. For Jobs only — Call security on the owner field and machine field as if on a create object.

Deleting an Object

The following represents a logical flow for deleting any object.

Call security to validate user authority to delete objects from the specified security group.

All activity controlled by Unicenter AutoSys JM is based on jobs. Other objects, such as Monitors, Browsers (Reports), and the Operator Console, serve to track job progress. A job is the basic building block upon which the entire operations cycle is built.

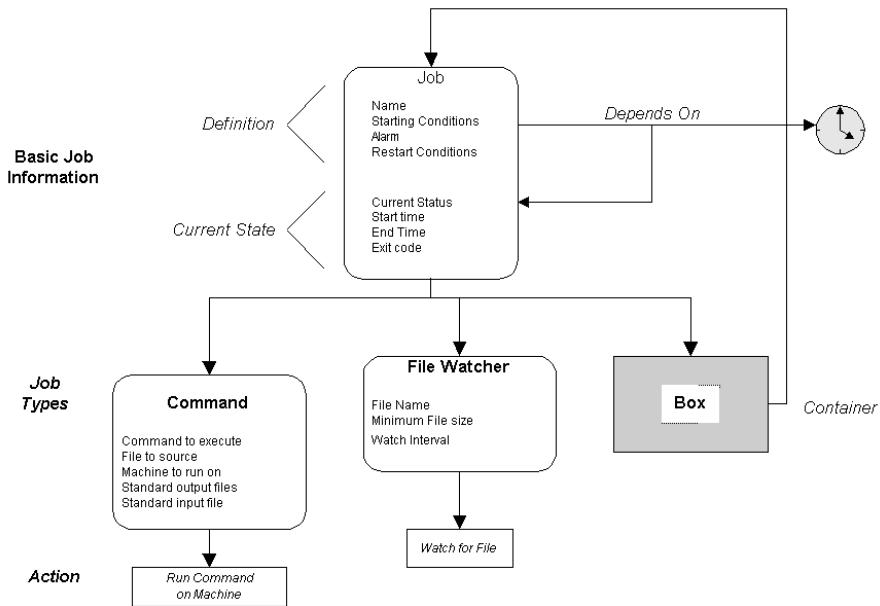
A job is any single command or executable, UNIX shell script, or Windows batch file. Each job definition contains a variety of qualifying attributes, including the conditions specifying when and where a job should be run.

As with most control systems, there are many ways to correctly define and implement jobs. It is likely that the way you utilize Unicenter AutoSys JM to address your distributed computing needs will evolve over time. As you become more familiar with both the features and the characteristics of your own jobs, you will also refine your use of Unicenter AutoSys JM.

Note: Before continuing with this chapter, read the chapter Maintaining for details on starting the event processor, which must be running before you start any processes.

Job Types and Structure

The following figure illustrates the structure of a job. There are three types of jobs: command, file watcher, and box. These job types have a majority of job attributes in common, and Unicenter AutoSys JM treats them all similarly. The primary differences between them are the actions that are taken when the job is run.



As their names imply, command jobs execute commands, box jobs are containers that hold other jobs (including other boxes), and file watcher jobs watch for the arrival of a specified file.

Basic Job Information

In the previous figure, the attributes listed inside the job region comprise what is called the basic job information, and are common to all jobs regardless of type. These attributes include the identifier name, the starting conditions, any specified alarms, the restart conditions, and a variety of other settings not shown (such as the box, if any, the job is in).

Notice, in the figure, that a job's starting conditions can be contingent on the date, time, and the status of any other job.

Command Jobs

The command job is commonly thought of (and referred to) as a job. The command can be a shell script, an executable program, a file transfer, and so forth. When this type of job is run, the result is the execution of a specified command on a client machine. When all the starting conditions are met, Unicenter AutoSys JM runs this command and captures its exit code upon completion. The exit event (either SUCCESS or FAILURE) and the exit code value are stored in the database.

In addition to the primary functionality described previously, a command job has the following supporting features:

Command Job Options	Action
Resource criteria	Unicenter AutoSys JM will check that a certain amount of free file space is available before starting a process. If it is not available, an alarm is sent and the job is rescheduled to start after a suitable delay.
Profile script	For each job, you can specify a script to be sourced before the execution of the command that defines the environment in which the command is to be run. All commands are run under the Bourne shell (/bin/sh). Therefore, all statements in the profile must use /bin/sh syntax.

Command Job Options	Action
I/O standard files	For each job, you can specify the standard input, standard output, and standard error files. To do this, use the JIL std_* commands, or use the Job Definition Advanced Features dialog.

Box Jobs

In the environment, the box job (or box) is a container of other jobs. A box job can be used to organize and control process flow. The box itself performs no actions, although it can trigger other jobs to run. An important feature of this type of job is that boxes can be put inside of other boxes. When this is done, jobs related by like starting conditions (not by similar application types) can be grouped and operated on in a logical way.

Note: Box jobs are very powerful tools for organizing, managing, and administering large numbers of jobs that have similar starting conditions or have complex logic flows. Knowing how and when to use boxes is often the result of some experimentation.

For more information on box jobs, see the chapter [Box Job Logic](#), in this guide.

Starting Conditions for Box Jobs

If no other starting conditions are specified at the job level, a job within a box will run as soon as the starting conditions for the box are satisfied. If several jobs in a box do not have job-level starting conditions, they will all run in parallel. Each time any job in a box changes state; the other jobs are checked to see if they are eligible to start running.

If jobs in a box have a priority attribute setting, they will be processed in order of priority, highest to lowest.

Jobs inside of boxes will be run only once per box execution. If you do specify multiple start times for a job during one box processing cycle, only the first start time will be used. This prevents jobs in boxes from inadvertently running multiple times.

Unicenter AutoSys JM starts a job if the current time matches, or is later than, the start time. In addition to explicit starting conditions, jobs inside of boxes have the following implicit condition: the box job itself is running. This means that jobs inside of a box will start only if the box job itself is running. However, if a job inside a box starts and the box job is stopped, the started job runs to completion.

Note: Some caution must be exercised when placing a job with more than one time-related starting condition in a box. For example, a job that runs at 15 and 45 minutes past the hour is placed in a box that runs every hour. The first time the box starts; the job runs at 15 minutes past the hour. A future start is then issued for 45 minutes past the hour, by which time the box has completed. As a result, the job will not run until the box is running again at the top of the next hour. At that time, the job runs as soon as the box starts because it is past the start time. The job runs, another future start job is issued for 15 minutes past the hour, the box completes, and the cycle repeats itself.

File Watcher Jobs

A file watcher job is similar to a command job. However, instead of starting a user-specified command on a client machine, it starts a process that monitors for the existence and size of a specific operating system file. When that file reaches a certain minimum size, and is no longer growing in size, the file watcher job completes successfully, indicating that the file has arrived.

Using file watcher jobs provides a means of integrating events that are external to Unicenter AutoSys JM into the processing conditions of jobs. For example, a file needs to be downloaded from a mainframe, and it is expected to arrive after 2:00 a.m. After it arrives, a batch job is to be run to process it, possibly even starting a whole sequence of jobs.

You could set up a file watcher job to start at 2:00 a.m., wait for the arrival of the specified file, and then exit. You could also set up the batch job so that the completion of the file watcher job is its only starting condition.

Basic Job Attributes

For each of the three job types, some job attributes are required. There are additional optional attributes that you can use for more advanced job definitions.

Note: The owner attribute is required for all job types, but is automatically assigned by Unicenter AutoSys JM.

Command Job Attributes

The basic command job definition has the following required attributes:

Job Name

The unique job identifier by which a job is referenced.

Command

The UNIX shell script, command, or application program to be executed.

Machine Name

The name of the machine on which the command is to be run.

Starting Conditions

The date/time or job dependency conditions necessary for the job to be run.
(This is not required, such as in cases where a job will always be started manually.)

File Watcher Job Attributes

The basic file watcher job definition has the following required attributes:

Job Name

The unique job identifier by which a job is referenced.

File Name to Watch For

The name of the file for which to watch.

Machine Name

The name of the machine on which the command is to be run.

Starting Conditions

The date/time or job status conditions necessary for the job to be run. (This is not required, such as in cases where a job will always be started manually.)

Box Job Attributes

The basic box job definition has the following required attributes:

Box Name

The unique job identifier by which the box is referenced. This name is used by other jobs as the name of their parent box.

Starting Conditions

The date/time or job status conditions necessary for the job to be run. (This is not required, such as in cases where a job will always be started manually.)

Job States and Status

Unicenter AutoSys JM keeps track of the current state, or status, of every job. The value of a job's status is used to determine when to start other jobs that are dependent on the job. The job status is displayed in the job report generated by the autorep command, and in the job report you can view in the Job Activity Console.

A job can have one of the following statuses:

Job State	Status
INACTIVE	The job has not yet been processed. Either the job has never been run, or its status was intentionally altered to turn off its previous completion status.
ACTIVATED	The top-level box that this job is in is now in the RUNNING state, but the job itself has not started yet.
STARTING	The event processor has initiated the start job procedure with the remote agent.
RUNNING	The job is running. If the job is a box job, this value simply means that the jobs within the box can be started (other conditions permitting). If it is a command or file watcher job, the value means that the process is actually running on the remote machine.
SUCCESS	The job exited with an exit code equal to or less than the maximum exit code for success. By default, only the exit code 0 is interpreted as success. However, a range of values up to the maximum exit code for success can be reserved for each job to be interpreted as success. If the job is a box job, this value means that all the jobs within the box have finished with the status SUCCESS (the default), or the Exit Condition for Box Success evaluated to true. (These exit conditions are discussed further in later sections.)

Job State	Status
FAILURE	The job exited with an exit code greater than the maximum exit code for success. By default, any number greater than zero is interpreted as failure. If the job is a box job, a FAILURE status means either that at least one job within the box exited with the status FAILURE (the default), or that the Exit Condition for Box Failure evaluated to true. Unicenter AutoSys JM issues an alarm if a job fails.
TERMINATED	The job terminated while in the RUNNING state. A job can be terminated if a user sends a KILLJOB event or if it was defined to terminate if the box it is in failed. If the job itself fails, it has a FAILURE status, not a TERMINATED status. A job may also be terminated if it has exceeded the maximum runtime (term_run_time attribute, if one was specified for the job), or if it was killed from the command line through a UNIX kill command. Unicenter AutoSys JM issues an alarm if a job is terminated.
RESTART	The job was unable to start due to hardware or application problems, and has been scheduled to restart.
QUE_WAIT	The job can logically run (that is, all the starting conditions have been met), but there are not enough machine resources available.
ON_HOLD	This job is on hold and will not be run until it receives the JOB_OFF_HOLD event.
ON_ICE	This job is removed from all conditions and logic, but is still defined. Operationally, this condition is like deactivating the job. It will remain on ice until it receives the JOB_OFF_ICE event.
<p>The difference between on hold and on ice is that when an on hold job is taken off hold, if its starting conditions are already satisfied, it will be scheduled to run, and it will run. On the other hand, if an on ice job is taken off ice, it will not start, even if its starting conditions are already satisfied. This job will not run until its starting conditions reoccur.</p>	

The other major distinction is that jobs downstream from the job that is on ice will run as though the job succeeded. Whereas, all dependent jobs do not run when a job is in on hold—nothing downstream from this job will run.

For details on how on ice affects boxes, see the chapter Box Job Logic, in this guide.

Example State Diagram: Simple Jobs

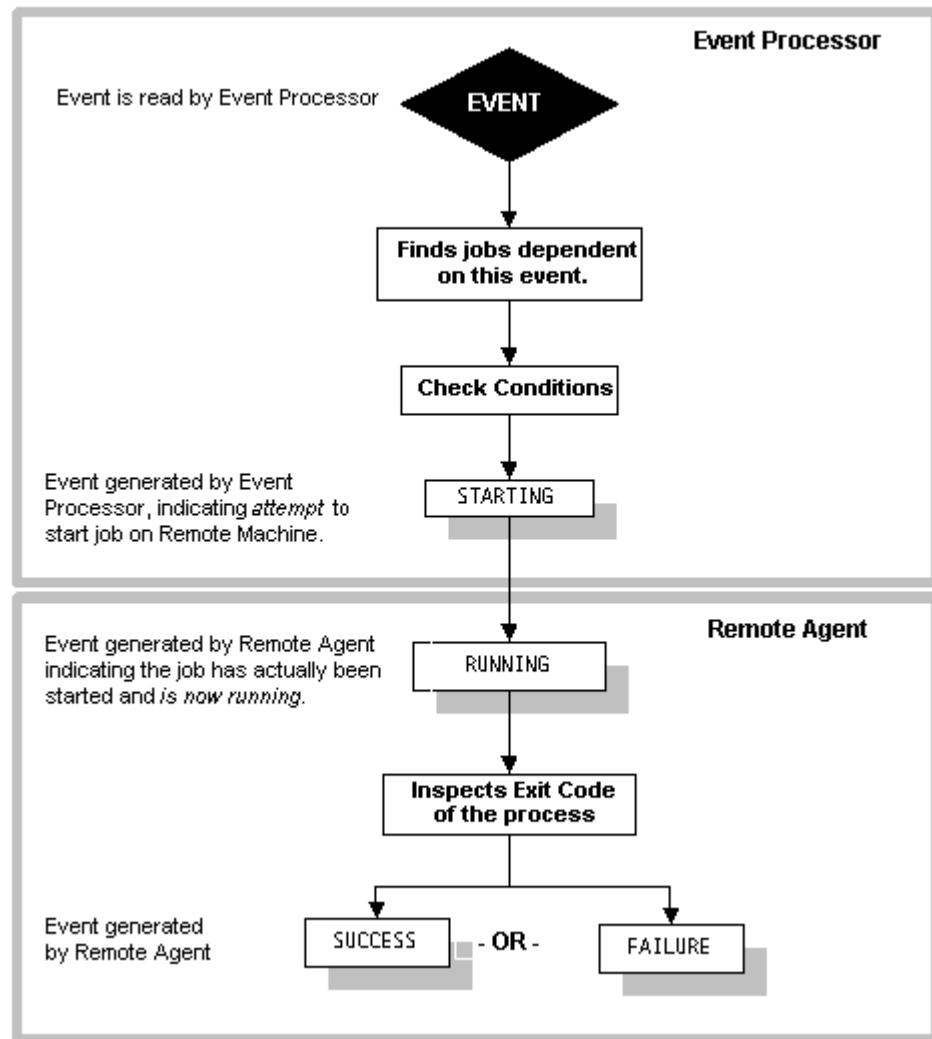
When a change in job status occurs, it is reported as a CHANGE_STATUS event, and the event processor records it in its log when this status is processed. For example, when the job test_job changes from the STARTING state to the RUNNING state, the event processor log will contain the following entry:

```
EVENT: CHANGE_STATUS STATUS: RUNNING JOB: test_job
```

Note: In the following diagrams, a state is depicted using the following box drawing:

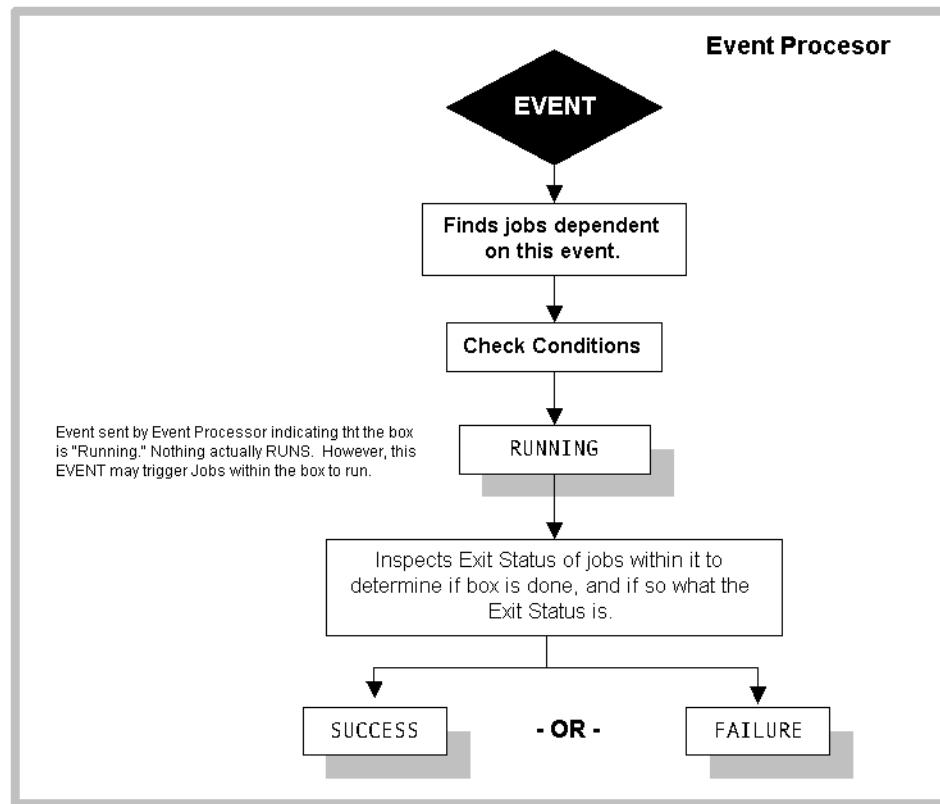


The following diagram depicts the simplest state transition for a job, in which an event satisfies the starting conditions for the job. The job starts, processes, and completes with either a failure or success exit code.



Example State Diagram: Box Jobs

For a job in a box, the job first goes into the ACTIVATED state when the top-level box it is in goes into the RUNNING state, as shown in the following figure. After the job starts, the remainder of the scenario is the same as for simple jobs.



In the case of a box, the box always goes into the RUNNING state as soon as all its starting conditions are met. This RUNNING event usually triggers jobs within the box to also start.

If the job has a priority associated with it, all its starting conditions have been met, and there are not enough machine resources available, it goes into the QUE_WAIT state. Once the resources become available, it goes into the STARTING state, then runs.

The value of status reflects the event processing. Therefore, a job may have actually completed on a machine and if the event processor has not processed that event yet, Unicenter AutoSys JM will still show the job's status as RUNNING. By displaying the detail of the job (either in the Job Activity Console, or in the output of the autorep command), you can see all the events for a job, including those that have not processed yet.

In addition, the status always reflects the most recent event that was processed. Therefore, after a job has completed, the status will remain as it is on completion. If it ended successfully, the status will remain as SUCCESS until the job is run again.

Note: When a box job starts, all jobs within the box change state to ACTIVATED before they run. Jobs will then run immediately, unless other conditions apply. If a box completes before a job is run, the job is set to INACTIVE at the time of box completion. As a result, jobs do not retain their statuses from previous box processing cycles once a new box cycle has begun.

Starting Parameters

Unicenter AutoSys JM determines whether to start or not to start a job based on the evaluation of the starting conditions (or starting parameters) defined for the job. These conditions can be one or more of the following:

- Date and time scheduling parameters are met (it is or has passed the specified date and time).
- Starting Conditions specified in the job definition evaluate to true.
- For jobs in a box, the box must be in the RUNNING state.
- The current status of the job is not ON_HOLD or ON_ICE.

Every time an event changes any of the above conditions, Unicenter AutoSys JM finds all the jobs that may be affected by this change, and determines whether or not to start them.

Note: It is very important to keep in mind the above four conditions. In order for a job to start, all defined starting conditions must be true.

Starting Parameters and Boxes

Be aware that for a job in a box to start, the box job must be running. Placing a job in a box means that the job inherits all the starting conditions of the box job. It also means that if there are no additional conditions on the job, it will be started as soon as the box is started. Also, a job runs only once per box execution.

By default, there is no concept of sequential job processing in a box. For example, if there are three jobs inside a box, and none of them has any additional conditions, then when the box is started, all three jobs will be started.

To implement a sequence within a box, you must specify additional starting conditions for each job. For example, Job1 may have no starting conditions, while Job2 is dependent on the completion of Job1, and Job3 is dependent on the completion of Job2.

Be aware that if this scenario was implemented and Job2 were placed ON_ICE, then Job1 and Job3 would start simultaneously as soon as the box they are in started running. (Jobs that are dependent on a job that is ON_ICE run as if that starting condition has been satisfied.)

Date/Time Dependencies

Jobs can be automatically scheduled to start at a certain date and time, based on the information you supply using JIL statements or the GUI. You define these dependencies by specifying the days or dates and times for time-based job starts. Unicenter AutoSys JM then calculates a matrix of these values and starts jobs at those times. A time range cannot span more than 24 hours. You can also specify a time zone to apply to your starting times.

For example, if you define a job to be started on Monday, Wednesday, and Friday at 8:00 a.m. and 5:00 p.m., it will be started 6 times a week: Monday at 8:00 a.m. and 5:00 p.m., Wednesday at 8:00 a.m. and 5:00 p.m., and Friday at 8:00 a.m. and 5:00 p.m.

You can specify days of the week or actual dates. However, you cannot specify both. You can specify days of the week using JIL or the GUI, but you can only specify actual dates through the use of custom calendars, which you can define using the Graphical Calendar Facility.

You can specify times as certain times of the day, or hourly, denoted in minutes past the hour. Again, the two formats are mutually exclusive. You can specify either form using JIL or the GUI (you do not have to create custom calendars).

TZ Environment Variable

By default, jobs with time-based starting conditions that do not specify a time zone are scheduled to start based on the time zone of the TZ environment variable (the same time zone under which the event processor runs).

Before you start the event processor, ensure that the TZ environment variable is set. The 3.4.4 event processor must be started once after you upgrade your database to insert the value of the TZ environment variable into the database. Do this before executing jil, autosc, autocons, or autorep.

Custom Calendars

Using the Graphical Calendar Facility or the autocal_asc utility, you can define any number of custom calendars, each with a unique name and containing any number of dates or date/time combinations. You can use these calendars in one of two ways: as days on which to run the jobs with which they are associated, or as days on which to not run the jobs with which they are associated. Calendars exist independently of any jobs that may be associated with them; they are referenced by jobs through job definitions.

Job Dependencies Related to Job Status

You can start jobs based on the current status of one or more jobs. These jobs must exist in the database. These starting conditions enable you to program simple or complex prerequisites that must be met in order to initiate a job.

Starting conditions can be as simple as specifying JobB to start when JobA achieves a SUCCESS status, and JobC to start when JobB achieves a SUCCESS status. In this way, you can implement a single-threaded, batch queue-like logic.

You can configure more complex conditions by combining a series of conditions with the AND or the OR logical operators. You may use the pipe symbol (|) instead of the word OR, and the ampersand symbol (&) instead of the word AND. Spaces between conditions and delimiters are optional. You can specify even more complex conditions by grouping the expressions in parentheses. The parentheses force precedence, and the equation is evaluated from left to right.

Given the sample script of:

```
(success (JobA) and success (JobB)) or (done (JobD) AND done (Job E))
```

would be evaluated, and the results would be A and B or D and E, reading from left to right.

Note: If a condition is specified for an undefined job, the condition will be evaluated as FALSE, and any jobs dependent on this condition will not run. To check for this type of invalid condition statement, you can use chk_cond, the stored procedure.

The syntax for defining job dependencies is the same whether the job is being defined using JIL or the GUI. The only difference is the JIL statement will begin with the JIL condition keyword, while the GUI field will only contain the language for the dependency itself. The dependency specification can take one of the following three forms:

- Based on the current status of other jobs
- Based on the UNIX exit codes of other jobs
- Based on global variables

The syntax for conditions based on job status is as follows:

status(job_name)

where:

<i>status</i>	Indicates the status of one of the following:
<i>success</i>	Indicates that the status condition for job_name is SUCCESS.
<i>failure</i>	Indicates that the status condition for job_name is FAILURE.
<i>done</i>	Indicates that the status condition for job_name is SUCCESS, FAILURE or TERMINATED.
<i>terminated</i>	Indicates that the status condition for job_name is TERMINATED.
<i>notrunning</i>	Indicates that the status condition for job_name is anything except RUNNING.
<i>job_name</i>	Indicates the job on which the new job is dependent.

You can abbreviate the status condition identifiers with the first letter, using s, f, d, t, and n. You can also abbreviate the dependency specification exit code with the letter e and VALUE (of a global variable) with the letter v. These abbreviations can be uppercase or lowercase.

You can control the value of the SUCCESS status by using the Maximum Exit Code for Success attribute, which can be set for a job. If you specify this attribute, any job that exits with an exit code less than or equal to the specified value will be treated as a success. A FAILURE status means the job exited with an exit code higher than this value. The convention (and the default) for normal job completion is 0. A TERMINATED status means the job was killed.

Note: Either uppercase or lowercase can be used to specify a status. However, the case cannot be mixed in either of the forms described.

Cross-Instance Job Dependencies

Cross-instance job dependencies can be implemented among different instances.

An instance is one licensed version of Unicenter AutoSys JM software running as a server, and as a server/client, on a single machine or on multiple machines. It uses its own event server and event processor and operates independently of other instances.

Multiple instances are not inherently connected, but they can communicate with each other. You can define jobs to have cross-instance dependencies, and multiple instances can send events to each other.

For example, multiple instances can send events to each other by way of a sendevent command line like the following:

```
sendevent -E STARTJOB -J job_name -S autoserv
```

The *job_name* argument is a job defined for the instance indicated by the *autoserv* argument, which is the instance's unique, capitalized three-character identifier.

In addition, jobs can be associated with more than one instance of AutoSys. For example, a job defined to run on one instance could have as a starting condition the successful completion of a job running on a different instance.

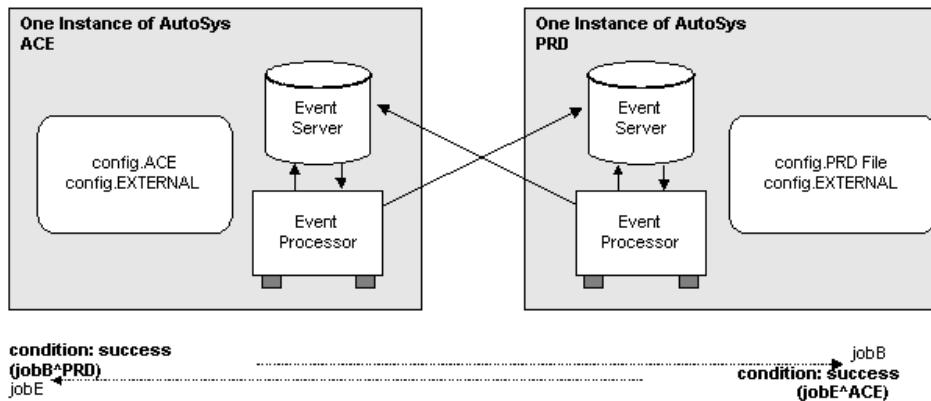
The specification for such a job dependency may look like the following:

```
condition: success(jobA) AND success(jobB^PRD)
```

The success (jobB[^]PRD) condition specifies the successful completion of a job named jobB running on a different instance specified with the three-letter ID of PRD. If the dependency specification does not include a caret (^) and a different instance ID, the current instance will be used, by default.

Each time a cross-instance dependency is encountered, an EXTERNAL_DEPENDENCY event is sent from the requesting instance. If the target instance cannot be reached, an INSTANCE_UNAVAILABLE alarm is issued.

The following figure shows two instances, each with a single-event server, exchanging cross-instance job dependencies.



Different instances can run from the same executables and can have the same values for \$AUTOSYS and \$AUTOUSER, both on the event processor machine and on machines running remote agents. However, they must have a different value for \$AUTOSERV.

For information on configuring for cross-instance job dependencies, see Running Cross-Instance Job Dependencies in the chapter Introduction in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

Event Processors

When cross-instance dependencies are implemented, different event processors can do the following:

- Run on different server machines or on the same server machine.
- Access the same client machines to start jobs.
- Send events to other instances.

Note: If the event server of a target instance is down, the event processor will try to resend an event (or events) every five minutes until the other instance's event server can be reached.

Event Servers

Event servers keep track of the cross-instance job dependencies. Each time a job definition with a cross-instance job dependency is submitted to the database, the following entries are made:

- An entry to the ext_job table of the issuing instance. The entries in this table specify the status of jobs in other instances in which this instance has an interest.
- An entry to the req_job table of the receiving instance. The entries in this table specify the jobs that have been specified as a job dependency in a job definition on the source AutoSys instance.

Jobs are entered using the job name, a caret symbol (^), and the instance name, as shown following.

jobB^PRD

The use of multiple databases is completely independent of instances using cross-instance dependencies. You can have multiple instances, each using dual-event servers.

Note: When communicating with event servers, event processors can only connect to those instances with like event servers. That is, instances with Sybase data servers can only connect with other instances having Sybase data servers. The same holds true for instances with Oracle databases.

Example Job Dependencies

For a job that runs only if the job named DB_BACKUP succeeds, the job dependency specification would be written as follows:

```
success (DB_BACKUP)  
or  
s (DB_BACKUP)
```

You can configure more complex conditions by combining a series of conditions with the AND or the OR logical operators. You can enter these operators in uppercase or lowercase, but not in mixed case. In addition, you can use the pipe symbol (!) instead of the word OR, and the ampersand (&) instead of the word AND. Spaces between conditions and delimiters are optional.

You can specify conditions that are more complex by grouping the expressions in parentheses. The parentheses do not imply any sort of precedence; they are simply used for grouping. For example, if JobC should only be started when both JobA and JobB complete successfully or when both JobD and JobE complete, regardless of whether they failed, succeeded, or terminated, you would specify the following dependency in the job definition for JobC:

```
(success (JobA) AND success (JobB)) OR (done (JobD) AND done (JobE))
```

or

```
(s (JobA) &s (JobB)) | (d (JobD) &d (JobE))
```

As indicated in this example, you can use any job status as part of the specification for a specific job's starting conditions. With this latitude, you can program branching paths that must be taken and that will provide alternate actions for error conditions.

For example, if JobB fails after processing only partially, you may want to call a routine titled Backout that backs out of the changes that were made. You would specify the following job dependency in the job definition for Backout:

```
failure (JobB)
```

or

```
f (JobB)
```

You use the notrunning operator to keep multiple jobs from running simultaneously (that is, running one job is exclusive of any others). For example, it may be best not to run a database dump (DB_DUMP) and a file backup (BACKUP) at the same time. This would cause the hard disk to be accessed very frequently. However, you may have a smaller job that can run as long as both of these resource-intensive jobs are not running. You would specify the smaller job's dependency like the following:

```
notrunning (DB_DUMP) AND notrunning (BACKUP)
```

Note: If you have jobs that you want to run exclusively, use the virtual machine and job queuing feature that is described in the chapter Load Balancing and Queuing Jobs, in this guide.

Managing Job Status

Starting conditions that are based on job status use the current (or most recent) completion status of the job. The current completion status is defined by the last execution of the job, regardless of when it last ran.

However, if you wish to enforce the concept of time-based processing cycles, where the completion status of a job for some previous time period should not affect the processing of this time cycle, there are several options you can use to control statuses.

When a box job is started, all the jobs within the box have their status changed to ACTIVATED. Therefore, downstream jobs in the box that depend on the completion of jobs upstream in the same box will use only the completion statuses from this run of the box. Placing the jobs in one processing cycle inside a top-level box and setting the box to start at the beginning of the processing cycle will prevent time-critical jobs from being affected by invalid information.

When a job is first entered into the database and prior to its being run for the first time, its status is set to INACTIVE. By changing to INACTIVE the status of jobs that have completed, but whose completion status should no longer be used in dependent job conditions, the completion status from the last run will no longer be the current status, and it will not be used.

To change a job status to INACTIVE, use the GUI (Send Event dialog), or use the sendevent command. Of course, you can create an AutoSys job to accomplish this as well. If you change the status of a top-level box to INACTIVE, all the jobs in the box are recursively set to INACTIVE.

Deleting and reinserting the job using JIL will accomplish the same thing. However, the past reporting history on the job will no longer be available. (Updating a job using JIL does not change the status of the job.)

Job Dependencies Based on Exit Codes

In addition to job status, you can base job dependencies on exit codes that indicate completed tasks. In this way, you can implement specific branching logic for recovering from job failures. For example, if a broken communication line results in JobA failing with an exit code of 4 and when this code is encountered, you want the system to execute a shell script (JobB) that redials the line, the syntax you would use to specify this type of job dependency is the following:

```
exitcode (job_name) operator value
```

where:

job_name

Indicates the name of the job upon which the new job is dependent.

operator

Indicates one of the following exitcode comparison operators: =, != (not equal), <, >, <=, or >=

value

Indicates any numeric value.

You can abbreviate the dependency specification exitcode with the letter e (uppercase or lowercase).

For this example, you would enter the following for the job dependency specification for the JobB redial job:

```
e (JobA) = 4
```

You can use any job status or exit codes as part of the specification for starting conditions. With this latitude, you can program branching paths that will provide alternative actions for all types of error conditions.

Using Exit Codes and Batch Files with Jobs Running on Windows

When you are defining jobs that will run batch files on Windows, you should be aware of, and account for, the Windows specific behavior.

Windows programs return an exit value that is programmed within the executable code. This exit value is the last thing returned to Windows when the program terminates.

Generally, a zero exit code indicates success; while a nonzero exit code indicates an error. The expected error values should be documented with each individual program, but some programs can return unexpected exit codes. You should modify these programs so that they return expected values. Use these values when specifying exit code dependencies.

Jobs are created using standard Windows process creation techniques. After the job has been created, the Remote Agent waits for the job to complete. When the job completes, Unicenter AutoSys JM gets the program exit code from Windows and stores it in the database for later use.

When launching programs directly from Unicenter AutoSys JM, the exit codes are returned and put in the database. However, there are some exit code behaviors that you must take into consideration when using Unicenter AutoSys JM to start *.BAT batch files.

The exit code returned from a batch file is the return code from the last operation executed from within that particular batch file. Consider the following example:

```
REM test batch file
test
if errorlevel 1 goto bad
goto good
:bad
del test.tmp
:good
exit
```

This example batch file will return a zero exit code as long as test.tmp exists. If test.tmp does not exist, the return code is from the del line and not from the line that executes test. Therefore, this batch file will return a zero (successful) exit code, even if the test failed to execute as intended.

To help handle situations like this, Unicenter AutoSys JM supplies a program called FALSE.EXE. This program is located for Windows %AUTOSYS%\bin directory and takes only one parameter, which is the exit code you want false to return on completion. You can use false in the previous example batch file, like the following:

```
REM test batch file
test
if errorlevel 1 goto bad
exit
:bad
del test.tmp
false 1
```

When test fails with errorlevel 1, this batch file will return an exit code of 1 from false, whether the test.tmp file exists or not.

Job Dependencies Based on Global Variables

Job dependencies can also be based on global variables you set using the Send Event dialog or the sendevent command. When using global variables in this way, the value of the expression must evaluate to TRUE for the job dependency to be satisfied. For example, you have a set of jobs in a box that are only supposed to run on special occasions, such as only on your manager's approval. In this case, you would set the global variable named manager-ok to OK, and make the top-level box job dependent on this global variable.

Global variables are referenced using the following expression:

VALUE (global_name) operator value

where:

VALUE

Can be uppercase or lowercase.

global_name

Indicates the name of the global variable upon which the job is dependent.

operator

Indicates one of the following: =, != (not equal), <, >, <=, or >=

value

Indicates any numeric value or text string (no quotes or spaces).

Note: global_name and the value can each be a maximum of 30 characters.

When using the Job Definition dialog to define a job, enter the expression shown here, in the Starting Condition field. When using JIL, enter the previous expression in the appropriate JIL script using the condition attribute.

You can abbreviate the dependency specification VALUE with the letter v (uppercase or lowercase).

In the example cited previously, you would enter the following syntax for the job's condition statement:

VALUE(manager-ok) = OK

or

v (manager-ok) = OK

Job Run Numbers and Names

Unicenter AutoSys JM employs the notion of run numbers for jobs. The run number is a unique integer associated with every run of a job. Consecutive run numbers are assigned every time a top-level job starts.

A top-level job is a job that is not contained in a box, and these run numbers are inherited by every job that is in a box. This means that all jobs within a top-level box have the same run number as the number used for the run of the box. This design permits runs of nested jobs to be associated together within the same run.

If there are restarts of a job, the run number remains the same, and the ntrys field is incremented. In the standard reports (see the autorep command in the chapter Commands in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*), these two values are displayed in the Run column as run_num/ntry.

The value of run_num/ntry is defined in the runtime environment for the job, and it is accessible to those shell scripts or executables executed as the job's command. This value is contained in the variable \$AUTORUN.

Unicenter AutoSys JM also maintains a value for each job's name, which is defined in the runtime environment for the job. As with \$AUTORUN, this value is accessible to those shell scripts or executables executed as the job's UNIX command. The value is contained in the variable \$AUTO_JOB_NAME.

Defining Jobs

You can define jobs using one of two methods: JIL statements or the GUI (in the Job Definition dialog). When you use JIL statements, you can input them interactively to the `jil` command, or you can store them in text files, which you can redirect into the `jil` command.

Alternatively, you can open the GUI by using the `autosc` command, and you can enter the job definition by filling in the appropriate fields of the Job Definition dialog and its associated dialogs.

You should back up your job definitions periodically, so you can have a file to restore from in case of system failure. This process is explained in Backing Up Definitions in the chapter Maintaining, in this guide.

Graphical User Interface Components

You can use the GUI to interactively define, run, manage, monitor, and report on jobs. Unicenter AutoSys JM provides the following top-level windows and dialogs, which you can launch from the GUI Control Panel:

- **Job Definition Dialog**

Used to define jobs. The Job Definition dialog and its related dialogs lets you create, view, edit, and delete job definitions for command jobs, box jobs, and file watcher jobs.

- **Graphical Calendar Facility**

Used to define calendar definitions. The Graphical Calendar Facility and its related dialogs let you create calendars in order to simplify job scheduling. It lets you create custom rules, block certain dates, set up conflict resolution, build calendars based on combinations of other calendars, and preview calendar definitions before assigning them to jobs. Then, you can assign them to a certain job, using the Job Definition dialog.

- **Operator Console**

Used to monitor and manage jobs. The Operator Console consists of the following components: Job Activity console, Job Selection dialog, Alarm Manager Dialog, and Alarm Selection dialog. The Job Activity console lets you monitor AutoSys jobs, and filter the jobs that it displays, you can use the Job Selection dialog. The Alarm Manager lets you browse and handle alarms. To filter the alarms that the Alarm Manager displays, you can use the Alarm Selection dialog.

- **Monitor/Browser**

Used to define monitors and reports. The Monitor/Browser lets you define filters by which you can screen system information. Monitors provide real-time views of the system. Browsers (reports) provide historical views of system information.

This chapter describes the essential and optional job attributes used to define jobs in Unicenter AutoSys JM. These attributes determine what a job does, as well as when and where it will run.

Job Attributes and Job Definitions

You define a new job by assigning it a name and specifying any number of attributes that describe its intended behavior. This specification of a job's behavior is called a job definition. There are two methods of creating a job definition: using JIL and using the GUI. Regardless of method, the specified set of attributes is the same, and the job definition is always stored in the database.

Before modifying or deleting an existing job, make sure the job is not running.

You should back up your job definitions periodically so you can have a file to restore from in case of system failure. This process is explained in Backing Up Definitions in the chapter "Maintaining," in this guide.

Using JIL to Create a Job Definition

When using JIL to create a job definition, you enter the `jil` command to display the JIL prompt. At this prompt, you define a job using the `insert_job` subcommand followed by any desired *attribute:value* statements, which specify an action to be performed. You can also use JIL subcommands to modify or delete an existing job definition.

Your JIL commands will look like:

```
insert_job: job_name  
attribute:value  
...
```

where:

- | | |
|--------------------------|---|
| <i>job_name</i> | Specifies a unique job name. |
| <i>attribute_keyword</i> | Identifies one of the legal JIL attributes. |
| <i>value</i> | Indicates the setting to be applied to the attribute. |

Using the GUI to Create a Job Definition

When using the GUI to create a job definition, issue the `autosc` command, select the Job Definition button, and set the various attributes and their values using the text fields and push-buttons in the Job Definition dialog.

Chapter Organization

In this chapter, job attributes are organized into two categories: essential and optional. Essential attributes are those that must be specified in order for the job definition to be accepted. As the name implies, optional attributes are not necessarily required for a job definition to be accepted.

For each attribute described in this chapter, we indicate its name, its JIL attribute keyword, its corresponding GUI object, or GUI field name, and a description of its use.

In the previous example, the “Job Type” attribute, which specifies whether a job is a command, file watcher, or box, is specified with the JIL keyword `job_type` and is identified in the GUI as the field with the name “Job Type.”

Because the chapter “JIL/GUI Job Definitions” in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide* is organized alphabetically by JIL keywords, the keywords in this chapter can act as “pointers” to more detailed descriptions about a particular attribute in the reference chapter.

Essential Job Attributes

Attributes Common to All Job Types

The following attributes are common to all job types: command, file watcher, and box. Although defaults may be available, the attributes in this section are still essential due to the fact that every job definition must include them, whether by default or by explicit specification.

Job Name

JIL Keyword	insert_job
GUI Field Name	Job Name
Description	The job name is used to identify the job, and must be unique. It can be from 1 to 30 alphanumeric characters, and is terminated with white space. Embedded blanks and tabs are illegal. Command, file watcher, and box jobs cannot use the same name.

Job Type

JIL Keyword	job_type
GUI Field Name	Job Type
Description	The job type specifies the type of job: command (c), file watcher (f), or box (b).

Job Owner

JIL Keyword	owner
GUI Field Name	Owner
Description	The job owner specifies whose user ID the command will be run under on the client machine. This attribute is automatically set to the user who invoked jil or the GUI to define the job, and cannot be changed except by the edit superuser.

Command Jobs Attributes

For command jobs, the following attributes must be specified in addition to those listed in Attributes Common to All Job Types (previous section).

Command

JIL Keyword	command
GUI Field Name	Command to Execute
Description	<p>The command attribute can be the name of any command, executable, UNIX shell script or batch file, and its arguments. When issuing commands that are to be run on a different operating system, you must use the syntax appropriate to the operating system of the client machine. The job's owner must have execute permission for this command on the client machine. Input and output redirection cannot be part of the command. Redirection is specified by other job attributes. Global variables can be used as part of the command name itself, or as part of the command's runtime arguments. To set a global variable, use the sendevent command, or use the Send Event dialog in the GUI.</p>

This command will be executed in the environment defined in the profile script—either the default /etc/auto.profile, or the one specified in the job definition (which you can define, and which preempts the default file). Therefore, if \$PATH is assigned in that script, that path will be searched to find the executable. For more information on specifying a profile, see Profile in the topic Command Job Attributes in this chapter.

The full path name can be specified, in which case, variables exported from the profile script can be used in the path name specification. If variable substitution is used, enclose the variable in curly braces, as in \${DIR}.

These are additional points to keep in mind with regard to the command attribute:

- Since Unicenter AutoSys JM performs an exec to run the command, multiple commands separated by a semicolon are not allowed.
- Piping or redirection of standard input, output, and error files is not allowed in the command attribute. Shell scripts can be invoked to execute piped commands and attributes, such as std_in_file used for standard input, to provide the necessary functionality.

- You cannot use the ampersand character (&) in the command attribute. A shell script can be called to provide that functionality.
- All commands are run under the Bourne shell (/bin/sh). Therefore, all statements in the profile must use /bin/sh syntax. For example:

```
Variable=value; export Variable
```

Do not use the following:

```
export Variable=value or setenv Variable Value
```

- If you are running a C-Shell (csh) script, the system will attempt to source a .cshrc file when it begins interpreting the file. Although this may be desired, the system will also overwrite any variables defined in the profile script (the default profile is /etc/auto.profile.) If you do not wish to have the .cshrc file sourced, you must invoke the csh script with the -f option. For example, the first line of the script should look like the following:

```
#!/bin/csh -f
```

- Only one file is sourced—either the default /etc/auto.profile or the profile file specified in the job definition. Therefore, the entire environment needed for the command must be defined in the profile file that will be sourced.
- Command line arguments can be passed using global variables.

Note: If a command is working properly when issued at a shell prompt, but it fails to run or run properly when specified as a command attribute, the shell and environments are probably different. If this is the case, ensure that all required command variables are specified in the profile script, either the default one or the one you have specified.

Machine to Run On

JIL Keyword	machine
GUI Field Name	Execute on Machine
Description	This attribute specifies the client machine on which the command should be run. The job's owner must have permission to access this machine and to execute the specified command at this machine. The machine can be a specific real machine (as listed in the /etc/hosts file of the server machine), a set of real machines, or a virtual machine.

Note: If you have implemented the shadow event processor feature, you should never set the machine attribute to localhost. The localhost value implies: “run on the machine on which the event processor is currently running.” The job may run normally on the primary event processor machine, and yet fail on the shadow event processor machine.

You can also specify the svload program or your own, custom load-balancing program in place of a machine name. In this case, the event processor will run the program at runtime to select the best-suited machine to run the job.

For more information about virtual machines, and choosing a machine to run on when you specify multiple machines or a load-balancing program, see the chapter “Load Balancing and Queuing Jobs,” in this guide.

File Watcher Job Attributes

For file watcher jobs, the following attributes must be specified in addition to those listed in Attributes Common to All Job Types in this chapter.

Machine to Run On

JIL Keyword	machine
GUI Field Name	Execute on Machine
Description	This attribute specifies the client machine on which the File Watcher should run. For a File Watcher, this attribute must specify a single real machine, defined in the /etc/hosts file on the server machine.

File to Watch For

JIL Keyword	watch_file
GUI Field Name	File To Watch For
Description	The name of the file to watch for must be a legal file name and must include the full path to the file. All directories in the path must exist, but the file itself does not have to exist at the time the job is defined. Environment variables defined and exported in the profile file (the specified or default), as well as global variables, can be used in the path. Wildcards cannot be used in the file name. For more information on how to use wildcards, see the <i>Unicenter AutoSys Job Management for UNIX and Windows Reference Guide</i> in the watch_file section.

When using the GUI, this field only appears when the File Watcher type has been selected. This attribute is used in combination with the Watch File Minimum File Size and Watch Interval attributes, to determine when a file is considered to have arrived.

Box Job Attributes

There are no essential attributes for box jobs other than those listed in Attributes Common to All Job Types in this chapter.

Optional Job Attributes

Common Job Starting Attributes

The following optional attributes are common to all job types: command, file watcher, and box. These attributes are used to specify when a job should start, and typically default to “inactive” or NULL if not specified.

For information on how date and time attributes are affected by the spring and fall time adjustments, see Date and Time Attributes and Time Changes in this chapter.

Start Date /Time Dependence

JIL Keyword	date_conditions
GUI Field Name	Is the Start Date/Time Dependent?
Description	The start date/time dependencies attribute is a toggle, which specifies whether or not there is date, time, or both, conditions required for starting the job. If the attribute is set to “no,” the remainder of the related date/time attributes, described following, will be ignored.

Days of the Week

JIL Keyword	days_of_week
GUI Field Name	Days of the Week
Description	The days of the week attribute specifies the days on which the job should be run. You can specify one or more days, or “all” for every day.

Days to Run Through a Custom Calendar

JIL Keyword	run_calendar
GUI Field Name	Run on Days in Calendar
Description	The days on which a job should be run can be specified by way of a custom calendar, rather than through a list of days of the week. Custom calendars, specified through the Graphical Calendar Facility, or the autocal_asc command, can include any number of dates on which the job should be run. Each calendar is stored in the database as a separate object with a unique name, and a calendar can be associated with one or more jobs, using this attribute or the exclude_calendar attribute.

Days to NOT Run Through a Custom Calendar

JIL Keyword	exclude_calendar
GUI Field Name	Do NOT Run on Days in Calendar
Description	The days on which a job should not be run can be specified by way of a custom calendar. Custom calendars, specified through the Graphical Calendar Facility, or the autocal_asc command, can include any number of dates on which the job should not be run. Each calendar is stored in the database as a separate object with a unique name, and a calendar can be associated with one or more jobs, using this attribute or the run_calendar attribute.

Specific Times of Day to Run

JIL Keyword	start_times
GUI Field Name	Times of Day
Description	This attribute specifies one or more specific times of day when the job should be started. The job will be started at each specified time of day, on every day specified in the associated date attributes. This attribute and the “Specific Times Every Hour to Run” (start_mins) attribute are mutually exclusive.

Time of Day Not to Run

JIL Keyword	run_window
GUI Field Name	Run Window
Description	<p>This attribute specifies a time range (or time window) during which a job can be started. When the starting conditions for a job have been met, Unicenter AutoSys JM checks if the current time is within the specified run window. The job will not start outside of the specified window.</p> <p>This attribute controls only when the job will start, not when it will stop running. This attribute is particularly useful when, for example, it is not known when a watched-for file will arrive, and there are certain times when jobs dependent on that file should not run. This setting can prevent a late-arriving file from causing a job to run at an inopportune time. The run window range cannot span more than 24 hours. Jobs that are not in a box must have starting conditions in addition to the run_window attribute in order for the job to be automatically started.</p>
	<p>Note: You can also block out times of day when you do not want a job to start by putting the job on hold, then taking it off hold later. The sendevent command can be used to accomplish this, executed either from the command line, through the Send Event dialog, or from within a shell script or batch file in another job.</p>

Specific Times Every Hour to Run

JIL Keyword	start_mins
GUI Field Name	Every Hour at
Description	<p>One or more specific times per hour when the job should be started can be specified. Each time is specified in minutes past the hour. The job will be started at each specified time every hour of the day, on every day specified in the associated date attributes. This attribute and the “Specific Times of Day to Run” (start_times) attribute are mutually exclusive.</p>

Job Dependencies (Starting Conditions)

JIL Keyword	condition
GUI Field Name	Starting Condition
Description	<p>Any number of job dependencies can be specified; however, every dependency must evaluate to true before the dependent job will be run. Examples of job dependencies include successful completion of a job, failure of a job, a job's exit code, and the value of a global variable. Various combinations of conditions may also be specified. Job dependencies can reference jobs residing on different instances.</p>

If a condition is specified for an undefined job, the condition will be evaluated as FALSE, and any jobs dependent on this condition will not run. To check for this type of invalid condition statement, you can use the chk_cond, stored procedure (see chk_cond (SP) in the chapter "Commands" in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*).

Note: You should study the Starting Parameters in the chapter "Jobs" and review the JIL examples provided in the chapter "Defining Jobs Using JIL." This functionality opens up all sorts of possibilities for controlling jobs, and provides information that will help you when creating your own job definitions.

Common General Attributes

The following optional attributes are common to all job types: command, file watcher, and box. These attributes are used to specify a variety of features, and typically default to "inactive" or NULL if not specified.

Description

JIL Keyword	description
GUI Field Name	Description
Description	This attribute provides a comment field, used for documentation purposes only. When entering a description using JIL, you should enclose the string in double quotes to ensure JIL properly interprets it. The GUI adds quotes for you automatically.

Box Name

JIL Keyword	box_name
GUI Field Name	Name of the Box this Job is IN
Description	Boxes allow a set of jobs to be manipulated as a group. This feature is particularly useful for setting starting conditions at the box level, to “gate” the jobs inside the box, then specifying their starting conditions relative to each other individually, if necessary. This attribute specifies the name of the box in which the job is to be placed. The specified box must already exist before you can place jobs in it.

Minimum Runtime Alarm

JIL Keyword	min_run_alarm
GUI Field Name	Minimum Runtime
Description	A minimum runtime (in minutes) can be specified for a job; the job should not end in less than the specified time. This may prevent an inadvertent truncation of the file being processed before it is complete. If the job does end prior to this time, an alarm is generated to alert someone to investigate the situation and take corrective action. Alarms are informational, and they do nothing on their own. A monitor or the Operator Console must be running and tracking alarms in order for them to be seen and acted upon in realtime.

Maximum Runtime Alarm

JIL Keyword	max_run_alarm
GUI Field Name	Maximum Runtime
Description	A maximum runtime can be specified for a job. If a maximum runtime is specified, the job should not take longer than the specified time to finish. This “reasonability” test may catch an error, such as the application being stuck in a loop, or the application waiting for additional data that may never arrive. If the job runs longer than this time, an alarm is generated to alert someone to investigate the situation and take corrective action. Alarms are informational, and they do nothing on their own. A monitor, or the Operator Console, must be running and tracking alarms in order for them to be seen and acted upon in real time.
The attribute “Terminate this job Mins after starting” (<i>term_run_time</i>) can be used to automatically terminate a job that has been running for too long. If <i>term_run_time</i> is not set, the job will continue running until manually interrupted, or it completes by itself.	

Terminate Due to Runtime

JIL Keyword	term_run_time
GUI Field Name	Terminate this job Mins after starting
Description	A maximum runtime (in minutes) can be specified for a job; the job should not take longer than the specified time to finish. This feature allows the job to be automatically terminated if it runs longer than the allotted time.

Send Alarm if the Job Fails

JIL Keyword	alarm_if_fail
GUI Field Name	Send ALARM if this Job Fails?
Description	This attribute specifies whether or not an alarm should be generated when the job fails. Failure is defined as the job completing with a FAILURE or TERMINATED status. (The “Maximum Exit Code for SUCCESS” attribute determines what codes are interpreted as FAILURE for a job, and the “Box Failure Condition” attribute determines what constitutes a box failure.) Alarms are informational, and they do nothing on their own. A monitor or the Operator Console must be running and tracking alarms in order for them to be seen and acted upon in realtime.

Terminate the Box if the Job Fails

JIL Keyword	box_terminator
GUI Field Name	If this Job fails should the Box it is IN be Terminated?
Description	This attribute specifies whether or not the box containing this job should be terminated if the job fails or terminates. By using this attribute in combination with the “Terminate the Job if the Box Fails” attribute, you can control how nested jobs react when a job fails. This attribute only applies if the job is being placed in a box.

Terminate the Job if the Box Fails

JIL Keyword	job_terminator
GUI Field Name	If the Box fails should this Job be Terminated?
Description	This attribute specifies whether or not the job should be terminated if the box it is in fails or terminates. By using this attribute in combination with the “Terminate the Box if the Job Fails” attribute, you can control how nested jobs react when a job fails. This attribute only applies if the job is being placed in a box.

Number of Times to Restart a Job

JIL Keyword	n_retrys
GUI Field Name	Number of Times to Restart this Job after a FAILURE
Description	This attribute specifies how many times, if any, the job should be restarted after exiting with a FAILURE status. The default is 0, which means the job will not be automatically restarted after an application failure. This attribute applies to application failures (for example, Unicenter AutoSys JM is unable to find a file or a command, or permissions are not properly set); it does not apply to system or network failures (for example, machine unavailability, the socket connect timed out, the fork in the Remote Agent failed, or the file system space resource check failed).
<p>The number of restarts after system or network failures is specified using the MaxRestartTrys parameter in the configuration file.</p>	

Time Zone for Job

JIL Keyword	timezone
GUI Field Name	Time Zone
Description	This attribute lets you schedule a job based on a chosen time zone. When this attribute is used, the time settings in the job are based on the specified time zone. For example, if you define a start time of 01:00 for a job running on a machine in Denver, and enter San Francisco in the Time Zone field, the job will start at 1:00 a.m. Pacific time, which is 2:00 a.m. in Denver.
<p>If you specify a time zone that includes a colon, you must quote the time zone name if you are using JIL. For example:</p>	
<pre>timezone: "IST-5:30"</pre>	
<p>If you do not quote a time zone specification that contains a colon, JIL will interpret the colon as a delimiter, producing unexpected results.</p>	
<p>Jobs with time-based starting conditions that do not specify a time zone will have their start event scheduled based on the TZ environment variable, which specifies the time zone under which the event processor is running.</p>	

Delete Job After Completion

JIL Keyword	auto_delete
GUI Field Name	Delete Job after completion?
Description	<p>This attribute indicates whether or not the job definition should be automatically deleted after successful completion. A number of hours can be specified (including “0” for immediately), or the attribute can be turned “off” by specifying a negative value (for example “-1”), which is the default. If auto_delete is set to 0, Unicenter AutoSys JM will immediately delete job definitions only if the job completed successfully.</p> <p>If the job did not complete successfully, Unicenter AutoSys JM will keep the job definition for seven days before automatically deleting it. This attribute is useful for scheduling and running a one-time batch job.</p>
<hr/>	

Autohold

JIL Keyword	auto_hold
GUI Field Name	Autohold On?
Description	<p>This feature is only for jobs in a box. When a job is in a box, it inherits the starting conditions of the box. This means that when a box goes into the RUNNING state, the box job will start all the jobs within it (unless other conditions are not satisfied). This is typically the desired behavior; however, there are occasions when it is not.</p> <p>For example, you may want to place a job in a box, but not start the job until a non-job (that is, operating system level) event arrives. By specifying “yes” to Autohold On, Unicenter AutoSys JM automatically changes the job state to ON_HOLD when the box it is in begins RUNNING. At this point, the job is in exactly the same state as if it were manually placed on hold. To start the job, take the job off hold by sending the JOB_OFF_HOLD event through the Send Event dialog or the sendevent command.</p>
<hr/>	

Permissions

JIL Keyword permission

GUI Field Name Permissions

Description In UNIX, there are three levels of permission by user ID (uid), and within Unicenter AutoSys JM, two levels of job permission.

Permissions are based on the UNIX user ID scheme. This scheme contains three types: owner, group, and world (or public). The owner is the user who originally created the job. The group is the primary group of which the owner is a member, and the world is every user with a valid logon for the system.

Unicenter AutoSys JM also allows designation of an exec superuser and an edit superuser, by way of the autosys_secure command. The edit superuser can edit any job definition, change the owner of a job, and change the permissions assigned to a job; no other user has this authority. The exec superuser can shut down the event processor, and can issue sendevent commands to any job, regardless of its owner.

The permission scheme used is as follows:

- **Execute**

If execute permissions are enabled for the user's group, allows the user to issue events that affect the running (starting, stopping, and so on) of the job. Users in primary and secondary groups have this permission.

- **Edit**

If edit permissions are enabled for the user's group, allows the user to edit or delete the job definition. Only users in the primary group have this permission.

The permission scheme is based on the same permissions used in native UNIX environment. The user ID (uid) and group ID (gid) specified in the UNIX environment do the following:

- Control access to the job definitions themselves.
- Determine the execution permissions to be used when executing the actual UNIX command specified in the job.

Command Job Attributes

For command jobs, the following optional attributes can be specified, in addition to those listed in Attributes Common to All Job Types in this chapter. These attributes typically default to “inactive” if not specified.

Profile

JIL Keyword	profile
GUI Field Name	Job Environment Profile
Description	The profile attribute specifies the file to be sourced by the Bourne shell before the specified command is executed. The remote agent always spawns a process and starts the Bourne shell in that process, passing it the name of the profile to be sourced. This profile typically includes definitions and exports of environment variables, which can be referenced in the job’s command. The primary environment variable in the profile is the \$PATH. If a profile is not specified, the default profile, /etc/auto.profile, is used. If the profile attribute is specified, that profile is searched for on the machine on which the command is to run. If a command that normally executes when entered at the command line fails when run as a job, it is usually due to the incomplete specification of the required environment for the command in the profile file.
Note:	It is essential that no Korn shell and C-shell statements appear in the profile file, because the Bourne shell that runs will not be able to process them. If you include these types of statements, unexpected results will occur, often interfering with the proper redirection of the stdin, stdout, and stderr files.

Redirection of the Standard Input File

JIL Keyword	std_in_file
GUI Field Name	File to Redirect to Standard Input
Description	The standard input file can be redirected to any file to which the job owner has read permission on the client machine. The full path name must be specified, although variables exported from the default /etc/auto.profile or job's profile file, as well as global variables, can be used in the path name specification.
	The default is /dev/null.

Redirection of the Standard Output File

JIL Keyword	std_out_file
GUI Field Name	File to Redirect to Standard Output
Description	The standard output file can be redirected to any file on the client machine to which the job owner has write permission. The full path name must be specified, although variables exported from the default /etc/auto.profile or job's profile file, as well as global variables, can be used in the path name specification. The default is /dev/null.
	By default, new information is appended to the file. By placing the following notation as the first characters in the std_out_file specification, you can specify if the error file should be appended to or overwritten:

> Overwrite file
">>> Append file

This setting overrides the instance-wide setting for the AutoInstWideAppend parameter in the configuration file. It also overrides the machine-specific setting for the AutoMachWideAppend parameter in the /etc/auto.profile file.

Note: If you are running jobs across platforms, realize that the event processor of the issuing instance controls the default behavior. If the issuing instance is Windows, the default is to overwrite this file.

Redirection of the Standard Error File

JIL Keyword	std_err_file
GUI Field Name	File to Redirect to Standard Error
Description	The standard error file can be redirected to any file on the client machine to which the job owner has write permission. The full path name must be specified, although variables exported from the default /etc/auto.profile or job's profile file, as well as global variables, can be used in the path name specification. The default is /dev/null.

By default, new information is appended to the file. By placing the following notation as the first characters in the std_err_file specification, you can specify if the error file should be appended to or overwritten:

```
> Overwrite file
>> Append file
```

This setting overrides the instance-wide setting for the AutoInstWideAppend parameter in the configuration file. It also overrides the machine-specific setting for the AutoMachWideAppend parameter in the /etc/auto.profile file.

Note: If you are running jobs across platforms, realize that the Event processor of the issuing instance controls the default behavior. If the issuing instance is Windows, the default is to overwrite this file.

Job Load

JIL Keyword	job_load
GUI Field Name	Job Load
Description	Machines can be assigned “maximum job loads,” which is a measure of the CPU load that is desirable for a machine at any given time. Similarly, jobs can be assigned loads, indicating the relative amount of processing power they consume. This scheme allows for machine loading to be controlled, and prevents a machine from being overloaded. If a job is ready to run on a designated machine, but the current load on that machine is too large to accept the new job’s load, the job will be “queued” for that machine, to be run when sufficient resources are available. For load balancing to function properly, all jobs to be run on a controlled machine must have job loads specified; otherwise, their impact on a machine cannot be measured.

Note: If you force a job to start, it will run even if its load exceeds the machine's max_load. Also, if job_load is specified for a job and no priority attribute (described following) is set, Unicenter AutoSys JM uses the default priority of zero, which means ignore the job_load and run the job immediately.

For information about load balancing on machines, see the chapter "Load Balancing and Queuing Jobs," in this guide.

Queue Priority

JIL Keyword	priority
GUI Field Name	Que Priority
Description	The queue priority establishes the relative priority of all jobs queued for a given machine, with the lower number indicating higher priority. If a job is ready to run on a designated machine, but the current load on that machine is too large to accept the new job's load, the job will be "queued" for that machine.
The priority attribute only influences the starting of jobs that are queued, unless the jobs are in a box. If jobs in a box have a priority attribute setting, they will be processed in order of priority, highest to lowest.	

Job Overrides

JIL Keyword	override_job
GUI Field Name	Edit One Time Over-Rides?

Description	You can specify a one-time job override for the next run of a particular job. An override lets you change the behavior of a job the next time the job runs.
-------------	---

The following attributes can be modified in a job override:

auto_hold	profile	watch_file_min_size
min_run_alarm	term_run_time	exclude_calendar
std_in_file	date_conditions	start_mins
command	run_calendar	watch_interval
n_retrys	watch_file	machine
std_out_file	days_of_week	start_times
condition	run_window	max_run_alarm std_err_file

For a description of how to use the GUI to specify job overrides, see Specifying One-Time Job Overrides in the chapter “Defining Jobs Using the GUI,” in this guide.

Maximum Exit Code for Success

JIL Keyword	max_exit_success
GUI Field Name	Maximum Exit Code for SUCCESS
Description	The maximum exit code for success attribute indicates what exit codes will be considered as a success. It is used when a command can exit with more than just a single exit code, indicating either “degrees of success,” or other conditions that may not indicate a failure. This attribute lets you define complex branching logic based on specific exit code values. Unicenter AutoSys JM reserves exit codes greater than 120 for internal use, so do not use exit codes of 120 or greater.

Average Runtimes

JIL Keyword	avg_runtime
GUI Field Name	(JIL only)
Description	The avg_runtime attribute is used to provide an average runtime (in minutes) for a job that is newly submitted to the database; it establishes this value in the absence of the job having been run multiple times. This attribute is used solely to establish an average runtime for the new job in the avg_job_runs table.

Heartbeat-Interval

JIL Keyword	heartbeat_interval
GUI Field Name	Heartbeat Interval (mins)
Description	Heartbeats are a means of monitoring a job's progress. It automates the common practice of outputting characters, similar to displaying "progress" asterisks across the screen as a process runs. If a job does not send a heartbeat within this specified interval, a HEARTBEAT alarm is generated. The heartbeat interval is specified in minutes.

To send a heartbeat from a C program, call the routine found in the following source file:

`$AUTOSYS/code/heartbeat.c`

To send a heartbeat from a Bourne shell script, execute the code found in the following file:

`$AUTOSYS/code/heartbeat.sh`

The event processor must be configured to check for heartbeats. To do this, modify the configuration file, which has the following name:

`$AUTOUSER/config.$AUTOSERV`

For more information about the configuration file, see Sample Configuration File in the chapter "Configuring," in this guide. For information on sending heartbeats, see Sending Heartbeats in the chapter "AutoSys API" in the *Unicenter AutoSys Job Management Reference Guide*.

Resource Check: File Space

JIL Keyword	chk_files
GUI Field Name	Resource Check - File System Space
Description	<p>This attribute specifies a minimum amount of file space that must be available on the designated file systems for the job to be started. One or more file systems, specified with full path names or directory names, and their corresponding sizes, can be specified. If multiple file systems are specified, separate them with a single space. When the remote agent is preparing to start the job on the client machine, it checks whether or not the required space is available before starting the job. If the requirements are not met, an alarm is generated and Unicenter AutoSys JM automatically reschedules the job to start again after a delay. It will perform the same resource check the next time it attempts to start. This feature is intended to prevent a job that is known to require large amounts of file space from failing due to a shortage of space during processing time.</p>

File Watcher Job Attributes

For file watcher jobs, the following attributes can be specified in addition to those listed in Essential Job Attributes (File Watcher Job Attributes) in this chapter. These attributes typically default to inactive if not specified.

Watch File Minimum Size

JIL Keyword	watch_file_min_size
GUI Field Name	Minimum File Size (in Bytes)
Description	The watch file minimum size determines when enough data has been written to the file to consider it “complete.” This attribute is specified in bytes. You should specify a reasonable file size to ensure that a nearly empty file is not assumed to be complete. Use caution with this attribute. If you specify a large file size Unicenter AutoSys JM will wait for the file to reach that size, even if the file has reached a steady state and is no longer growing.

Watch Interval

JIL Keyword	watch_interval
GUI Field Name	Time Interval (secs) to Determine Steady State
Description	The watch interval specifies (in seconds) how often the File Watcher should check the current file size to ascertain whether data is still being written to the file. The default is every 60 seconds.

Resource Check: File Space

JIL Keyword	chk_files
GUI Field Name	Resource Check - File System Space
Description	This attribute specifies a minimum amount of file space that must be available on designated file systems for a command job to be started. One or more file systems, specified with full path names or directory names, and their corresponding sizes can be specified. When the remote agent is preparing to start the job on the client machine, it checks whether the required space is available before starting the job. If the requirements are not met, an alarm is generated. File watcher jobs will still be started.

Box Job Attributes

For box jobs, the following optional attributes can be specified, in addition to those listed in Attributes Common to All Job Types in this chapter. These attributes typically default to “inactive” or NULL if not specified.

Box Successful Completion

JIL Keyword	box_success
GUI Field Name	SUCCESS Conditions
Description	The default condition required for a box to be considered successful is that every job in the box must have completed with a success condition. A box can contain complex branching logic, which can take a number of different paths, all of which constitute a success. In this case, some jobs in the box may never need to run; but if the default box behavior is applied, the jobs that did not run would prevent the box from ever completing.
	This attribute can be used to specify what is considered a success, which could be as simple as the success of a single job, or as complex as necessary. This attribute is only displayed in the GUI when you select a box job type.

Box Failure

JIL Keyword	box_failure
GUI Field Name	FAILURE Condition
Description	<p>The default condition required for a box to complete with a FAILURE status is that all jobs in the box have completed and one or more jobs in the box completed with a failure condition. A box can contain complex branching logic, which may take a number of different paths, one of which may include recovery from a failed job. In this case, you may want the box to be considered successful, even though a job within it failed. This attribute can be used to specify what will be considered as a failure, which could be as simple as the failure of a single job, or as complex as necessary. This attribute is only displayed in the GUI when you select a box job type.</p>

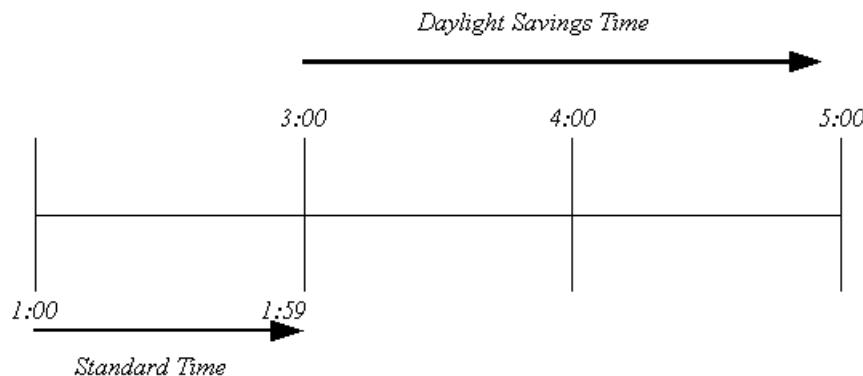
Date and Time Attributes and Time Changes

Date and Time attributes can be affected by the spring and fall time adjustments. The following sections describe the job run behavior you should expect and what plans you can make for this behavior.

The Time Change

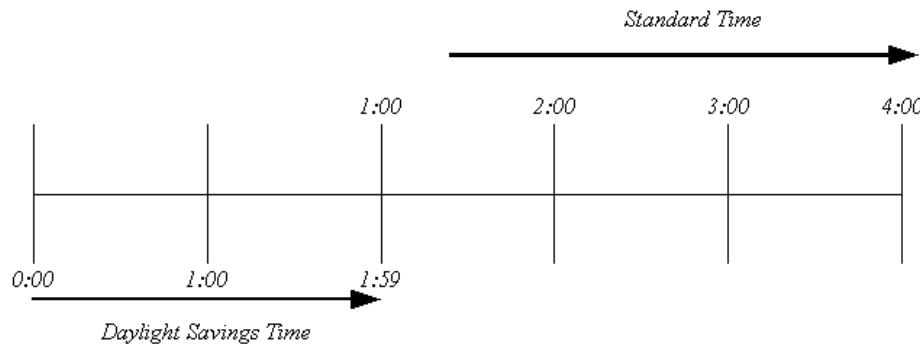
During the changes to and from daylight saving time, your operating system automatically changes the system clock to reflect the switch to either Standard Time (ST) or Daylight Time (DT).

In the spring, at 2 a.m., the clocks spring forward to 3 a.m. In most of the United States, this happens on the first Sunday in April. The following figure illustrates the Standard to Daylight Savings time change.



When this change occurs, time runs 1:58 ST, 1:59 ST, 3:00 DT, 3:01 DT, and the 2:00 to 2:59 hour is lost.

In the fall, at 2 a.m., the clocks fall back to 1 a.m. In most of the United States, this happens on the fourth Sunday in October. The following figure illustrates the Daylight Savings to Standard time change.



When this change occurs, time runs 1:58 DT, 1:59 DT, 1:00 ST, 1:01 ST,..., 2:00 ST, 2:01 ST, and the 1:00 to 1:59 hour is repeated.

Behavior During Time Change

Jobs that are time dependent may have their scheduling shifted to adjust for the time change. Jobs that are not time dependent, but have other starting conditions, will run as normal.

There are two types of time dependencies: absolute, and relative. Absolute times are defined to occur at a particular time of day, for example 9:30 on Thursday, or 12:00 on December 25. Absolute time dependent job attributes include:

- days_of_week
- exclude_calendar
- run_calendar
- run_window
- start_times

Relative times are specified with respect to either the current time, or relative to the start of the hour. For example, start a job at 10 and 20 minutes after the hour, or terminate a job after it has run for 90 minutes. Relative time dependent job attributes include:

- auto_delete
- max_run_alarm

- min_run_alarm
- start_mins
- term_run_time
- watch_interval

During the time change, absolute time attributes will behave differently than relative time attributes, as described following.

Spring Time Change

During the change to daylight saving time in the spring, the “2:00-2:59” hour is “lost,” therefore Unicenter AutoSys JM cannot schedule any jobs during that nonexistent hour.

The solution is to schedule jobs with absolute time dependencies for the missing hour to start within the first minute of the 3:00 DT hour. For example, a job scheduled to run on Sundays at 2:05, will run at 3:00:05 that day; a job scheduled to run everyday at 2:45 will run at 3:00:45. Although it may not be possible to start a large number of jobs within the first minute of the hour, this feature does somewhat preserve the scheduling order.

If you scheduled a job to run more than once during the missing hour, for example, 2:05 and 2:25, only the first scheduled job would run. Any additional start times for the same job in the missing hour will be ignored.

Relative time dependencies, such as start_mins, will run as you would expect. For example, a job specified to run at 0, 20, and 40 minutes after the hour will be scheduled for 1:00 ST, 1:20 ST, 1:40 ST, 3:00 DT, 3:20 DT, and 3:40 DT. Relative interval calculations, such as max_run_alarm, min_run_alarm, term_run_time, and watch_interval are still calculated in minutes out from when the job started. For example, if our Sunday at 2:05 job has a term_run_time of 90 minutes, the job will start shortly after 3:00, the term_run_time will be at 4:30.

Therefore, the behavior between two jobs that appear to have the same times specified, but use start_times versus start_mins, will not be the same. For example, job “Jrel” has start minutes of 10 and 20 minutes after the hour, and job “Jobs” has start times of 1:10, 1:20, 2:10, 2:20, 3:10, and 3:20. “Jrel” will run at 1:10, 1:20, 3:10, and 3:20. “Jobs” will run at 1:10, 1:20, 3:00, 3:10, and 3:20.

Run Windows

Run windows are treated a bit differently. If the specified closing of the run window falls within the missing hour, its recalculated closing time will be bumped up an hour, so that the effective duration of the run window remains the same. For example, a run window of 1:00 - 2:30 will have the closing time move to 3:30, so that the run window still remains open for an hour and a half.

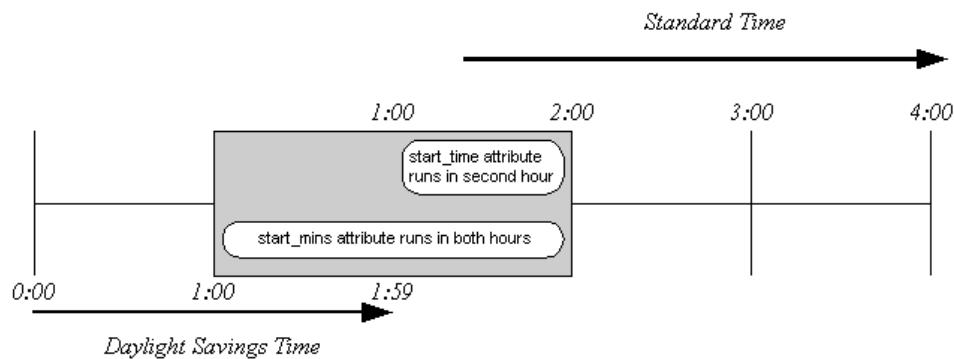
If the specified opening of the run window falls within the missing hour, its opening time is moved to 3:00. The closing time does not get altered, therefore the run window is foreshortened. For example, a run window of 2:45 - 3:45 will become 3:00 - 3:45, and the actual run window elapsed time will be 15 minutes shorter.

If both the specified opening and closing of the run window is within the missing hour, its opening time is moved to the first minute after 3:00, and its closing time is pushed forward one hour. Therefore, the resultant run window may be lengthened. For example, a run window of 2:15 - 2:45 will become 3:00 - 3:45, or 15 minutes longer.

Fall Time Change

During the change from daylight saving to standard time in the fall, there are two “1:00-1:59” hours. Jobs with start_times set between 1:00 and 1:59 will run only in the second, or Standard Time hour. Jobs with start_mins settings will run in both hours.

For example, a job scheduled to run on Sundays at 1:05, will run only at the second 1:05. A job scheduled to run every 30 minutes will run at 1:00 DT and 1:30 DT, then again at 1:00 ST and 1:30 ST, and so on (as shown in the following figure).



Jobs that are not time-based, but have other dependencies, will still run during the first hour.

Relative interval calculations, such as max_run_alarm, min_run_alarm, term_run_time, and watch_interval are still calculated in minutes out from when the job started. For example, if a job is scheduled to run on Sunday at 0:30, and has a term_run_time of 120 minutes, the job would normally be terminated at 2:30. On the day of the fall time change, it will terminate at 1:30 Standard Time, which is 120 minutes after the job started.

Testing the Fall Time Change

During the fall time change from Daylight Savings time to Standard Time, your operating system automatically falls back one hour from 2 a.m. to 1 a.m., causing the hour from 1 a.m. to 2 a.m to be repeated.

When testing this time change, you must set the clock to a time before 1 a.m. and allow the entire hour to pass before you can observe the time change. If you manually set the time to a period within the 1 a.m to 2 a.m. window, the system will assume that the time change has already occurred and will not reset at 2 a.m.

Run Windows

Run windows are treated a bit differently. If the specified opening of a run window is before the time change, and its specified closing falls within the repeated hour, it will close during the daylight saving, or first hour. For example, a run window of 11:30 - 1:30 will have the closing time of 1:30 DT, not 1:30 ST, which means that the run window remains open for its specified two hours. This may be a problem if there are also associated start times on the job that occurs during the repeated hour. In the example above, if the job also had a start time of 1:15, the start time would be calculated for 1:15 ST, and the job would not run on the day of the time change.

If the specified opening of the run window falls within the repeated hour, its opening time is moved to the second, Standard Time hour. The closing time does not get altered, therefore the length of the run window will remain the same. For example, a run window of 1:45 - 2:45 will become 1:45 ST to 2:45, or the same hour in length.

If both the specified opening and closing of the run window is within the repeated hour, the run window will be open during the second, Standard Time hour.

This chapter explains how box jobs work, including default box behavior and how to override the default behavior. It also explains what types of jobs should and should not be placed in a box. To illustrate box logic, several examples of box job definitions and job streams are provided in Examples.

Basic Box Concepts

A box is a container of jobs with like starting conditions (either date/time conditions or job dependency conditions). Use boxes to group jobs with like scheduling parameters, not as a means of grouping jobs organizationally. For example, if you have a number of jobs that run daily at 1:00 a.m., you could put all these jobs in a box and assign a daily start condition to the box. However, a variety of account processing jobs with diverse starting conditions should not be grouped in the same box.

Default Box Job Behavior

Some important rules to remember about boxes are as follows:

- Jobs run only once per box execution.
- Jobs in a box will start only if the box itself is running.
- Boxes should be used primarily for jobs with the same starting conditions. A box used to group sequential jobs is limited to 1,000 jobs.
- As long as any job in a box is running, the box remains in RUNNING state; the box cannot complete until all jobs have run.
- By default, a box will return a status of SUCCESS only when all the jobs in the box have run and the status of all the jobs is success.

Default SUCCESS is described in Default Box Success and Box Failure.

- By default, a box will return a status of FAILURE only when all jobs in the box have run and the status of one or more of the jobs is failure.
Default FAILURE is described in Default Box Success and Box Failure.
- Unless otherwise specified, a box will run indefinitely until it reaches a status of SUCCESS or FAILURE. For a description of how to override this behavior, see Box Job Attributes and Terminators.
- Changing the state of a box to INACTIVE (through the sendevent command) changes the state of all the jobs in the box to INACTIVE.

When You Should Not Use a Box

The fact that all jobs in a box change status when a box starts running has led some to use boxes to implement job cycle behavior. Be aware that placing jobs in a box to achieve this end may bring undesired behavior due to the nature of boxes.

Avoid the temptation to put jobs in a box as a shortcut for performing events (such as ON_ICE or ON_HOLD) on a large number of jobs at once. You will most likely find that the default behavior of boxes inhibits the expected execution of the jobs you placed in the box.

Likewise, you should not place jobs in a box solely because you want to run reports on all of them. When you run autorep on a box, you will get a report on the box and all the jobs in the box (unless you use the -L0 option). In addition, if you use wildcarding when specifying a job name, you could get duplicate entries in your report. For example, suppose you have a box named acnt_box containing three jobs named acnt_job1, acnt_job2, and daily_rep. If you specify acnt% as the job name for the autorep report, the report will have an entry for the box acnt_box and an entry for each job in the box. Then autorep will continue searching for all job names matching the wildcard characters and will list acnt_job1 and acnt_job2 a second time.

What Happens When a Box Runs

As soon as a box starts running, all the jobs in the box (including sub-boxes) change to status ACTIVATED, meaning they are eligible to run. (Because of this, jobs in boxes do not retain their statuses from previous box cycles.) Then each job is analyzed for additional starting conditions. All jobs with no additional starting conditions are started, without any implied ordering or prioritizing. Jobs with additional starting conditions remain in the ACTIVATED state until those additional dependencies have been met. The box remains in the RUNNING state as long as there are activated or running jobs in the box.

If a box is terminated before a job in it was able to start, the status of that job will change directly from ACTIVATED to INACTIVE.

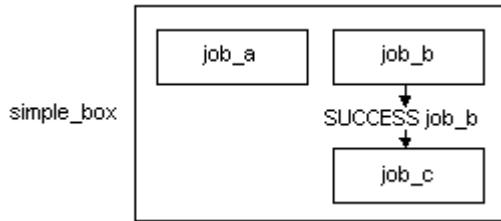
Note: Jobs in a box cannot start unless the box is running. However, once the job starts running, it will continue to run even if the box is later stopped for some reason.

Once all the jobs in a box have completed successfully, the box completes with a status of SUCCESS. The status of the box and the jobs in the box remain unchanged until the next time the box runs. (Some exceptions to this are explained in How Job Status Changes Affect Box Status, discussed later in this chapter.)

If a box changes to TERMINATED state, for example, if a user sends a KILLJOB event, it will stay in TERMINATED state until the next time it is started, regardless of any later state changes of jobs within the box.

Simple Box Job

In this example, a box named simple_box contains three jobs: job_a, job_b, and job_c. job_a and job_b have no starting conditions; the starting condition for job_c is the success of job_b.



When simple_box starts running, all jobs change to state ACTIVATED. Because job_a and job_b have no additional starting conditions, they will start running. After job_b completes successfully, job_c will start. When job_c completes successfully, the box completes with status of SUCCESS.

If job_b fails, job_c will not start; it will remain in ACTIVATED state. Because no contingency conditions have been defined, simple_box will continue running indefinitely, waiting for the default completion criteria to be met, namely that all jobs in the box ran.

Box Job Attributes and Terminators

The following sections describe how to use various job attributes to control the behavior of box jobs and their contained jobs.

Attributes in a Box Job Definition

Two box job attributes override the default success or failure of a box. They are `box_success` and `box_failure`. If included in a box job definition, they determine what conditions must be met to put the box in a state of SUCCESS or FAILURE. If you specify conditions for success of a box you should also specify failure conditions; otherwise the default failure conditions are applied.

Example of a Non-Default Success Condition

Using the previous simple box example, assume you defined the following success condition for `simple_box`:

```
box_success: success(job_a)
```

If `job_a` runs successfully, and `job_b` is still running, `job_c` would pass from ACTIVATED state directly to INACTIVE state without ever running because the box it is in would no longer be running.

When overriding default box terminators, be careful that you do not define conflicting success and failure conditions.

Attributes in a Job Definition

Two attributes you can add to the definition of a job within a box to force either the job or the box to stop running, are `box_terminator` and `job_terminator`.

The `box_terminator: y` attribute specifies that if the job fails, the box the job is in should terminate. If you use this attribute, be sure you have defined conditions for the other jobs in the box in the event that the box is terminated. For more information, see the figure in Using the Box Terminator Attribute in this chapter.

The `job_terminator: y` attribute specifies that if the box the job is in fails, this job will terminate. If you want every job in a box to terminate upon box failure, you must add this attribute to every job definition. For more information, see the figure in Using the Job Terminator Attribute in this chapter.

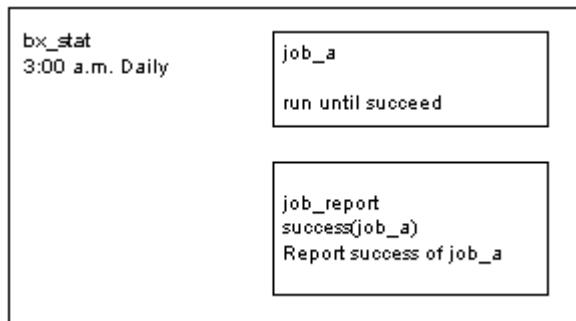
Time Conditions in a Box

Each job in a box will run only once per box execution. Therefore, you should not define more than one time attribute for any job in a box because the job will only run the first time. If you want to put a job in a box, but you also want it to run more than once, you must assign multiple start time conditions to the box itself, and define no time conditions for the job.

Remember also that the box must be running before the job can start. Do not assign a start time for a job in a box if the box will not be running at that time. If you do, the next time the box starts the job will start immediately.

The following example illustrates a scenario that would not work properly if placed in a box.

job_a is defined to run repeatedly until it succeeds. job_report has one starting condition—the success of job_a.



At 3:00 a.m., bx_stat starts running, which causes job_a to start running. If job_a is successful, job_report runs and all goes as expected. However, if job_a fails, it will not be able to run again until the next time the box starts, because jobs run only once per box execution. job_report will still be ACTIVATED waiting for the success of job_a, and the status of the box will be RUNNING. The box will remain in this state indefinitely.

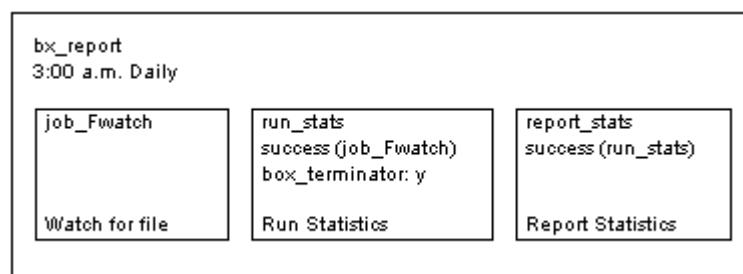
Force Starting Jobs in a Box

The FORCE_STARTJOB command forces a job to start even if its starting conditions have not been met. The command has the form as follows:

```
sendevent -E FORCE_STARTJOB -J job_name
```

You can also execute this by selecting the Force Start Job button in the Job Activity Console.

If you force start a job in a box, the state of the box influences whether or not other jobs in the box will run as expected, as shown in the following example.



In the previous figure, if the job `run_stats` fails, the `bx_report` box job will terminate because `run_stats` has a `box_terminator` attribute. If you force start `run_stats`, and it completes successfully, `report_stats` would still not start because the box it is in is not running.

The next section discusses how job status changes influence the status of the container box.

How Job Status Changes Affect Box Status

If a box that is not running contains a job that changes status, as a result of a FORCE_STARTJOB or CHANGE_STATUS event, the new job status could change the status of its container box. A change of status of the box could trigger the start of downstream jobs that are dependent on the box.

If a box contained only one job, and the job changed status, the box status would change as shown in the following table:

Current Box Status	New Job Status	New Box Status
SUCCESS	TERMINATED or FAILURE	FAILURE
SUCCESS	SUCCESS	Box status does not change
FAILURE	SUCCESS	SUCCESS
FAILURE	FAILURE	Box status does not change
INACTIVE	SUCCESS	SUCCESS
INACTIVE	TERMINATED or FAILURE	FAILURE
TERMINATED	Any change	Box status does not change

If another job is dependent on the status of the box, the status change could trigger the job to start. If the box status does not change, dependent jobs are not affected.

If the box contains other jobs in addition to the job that changed status, the status of the box will be evaluated again according to the success or failure conditions assigned to the box (either the default or user-assigned). Any jobs in the box with a status of INACTIVE are ignored when the status of the box is being reevaluated. For example, consider an INACTIVE box that contains four jobs, all with a status of INACTIVE (typical of a newly created box). If one of the jobs is force started and completes successfully, the status of the box will change to SUCCESS even though none of the other jobs ran.

Examples

Spend some time studying the examples in this section. They will help explain the logic of job flow in a box and reduce your chances of creating unexpected box behavior.

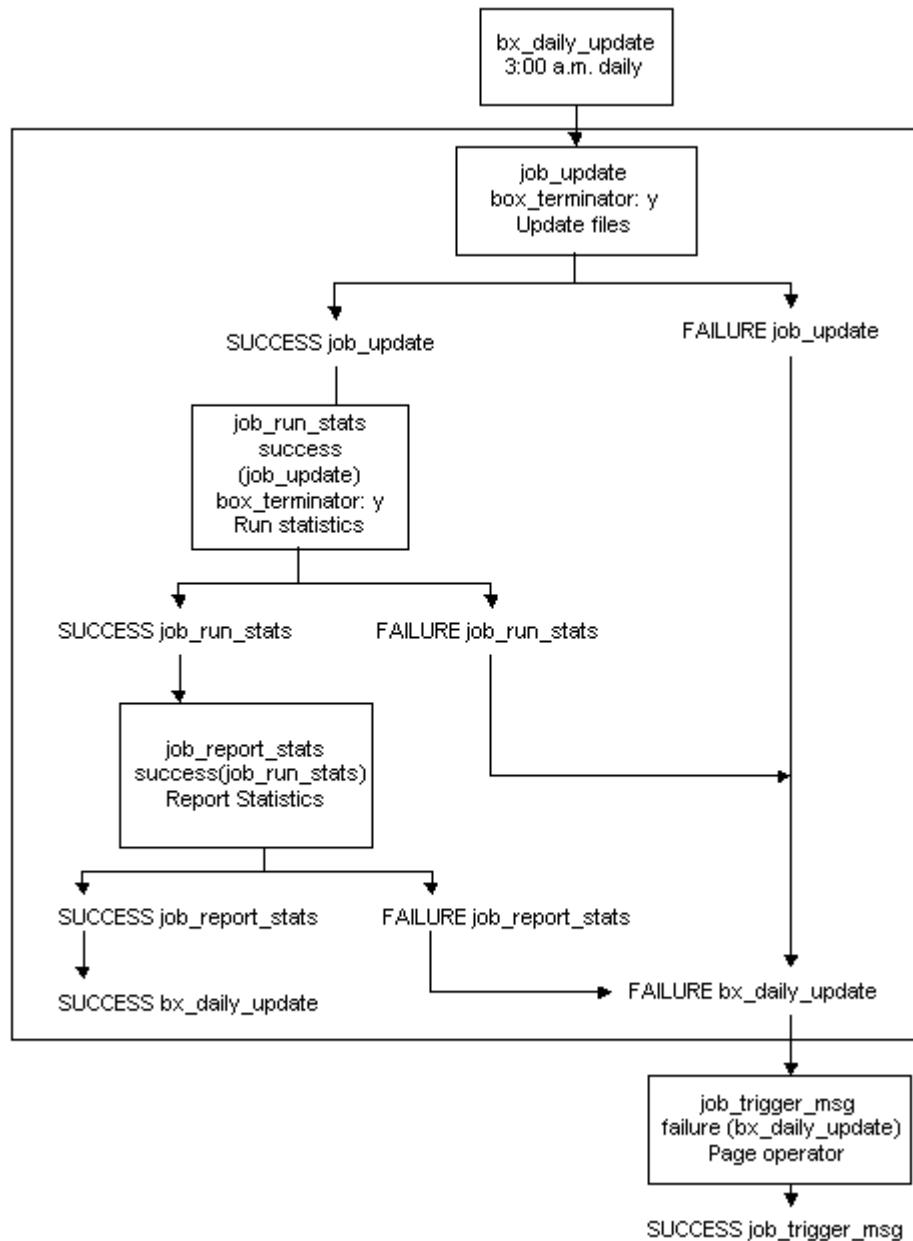
Advanced Conditions in Box Jobs

In the following example, conditions are defined to address different situations where the box may complete with a FAILURE status. The logic of this job flow is as follows:

- The box job named `bx_daily_update` has date and time conditions specified for its starting conditions; it runs every day of the week at 3:00 a.m. This box contains three command jobs whose overall purpose is to update files and generate a report.
- The command job named `job_update` updates a set of files. It is defined as being inside `bx_daily_update`. It will run as soon as `bx_daily_update` starts because it has no other starting conditions. This job has a `box_terminator` attribute; therefore, if this job fails, the box containing this job will be terminated.
- The command job named `job_run_stats` runs statistics on the updated files. It is defined as being inside `bx_daily_update`. It will run only on the successful completion of the job named `job_update`. This job has a `box_terminator` attribute; therefore, if this job fails, the box containing this job will be terminated.
- The command job named `job_report_stats` reports on the statistics generated by `job_run_stats`. It is defined as being inside `bx_daily_update`. It has a job dependency condition specified for its starting parameter. It will run only on the successful completion of the job named `job_run_stats`.

- The command job named job_trigger_msg has a job dependency condition specified for its starting parameter. It will run only on the FAILURE of the box job named bx_daily_update. This job will page an operator in order that the problem is investigated.

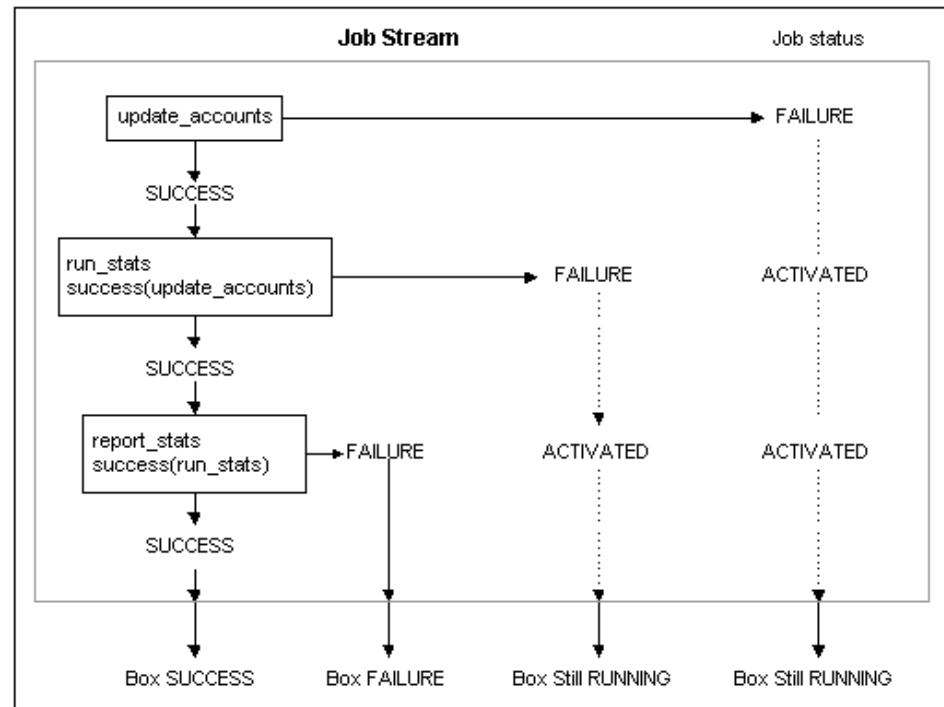
The following figure illustrates this job stream:



Default Box Success and Box Failure

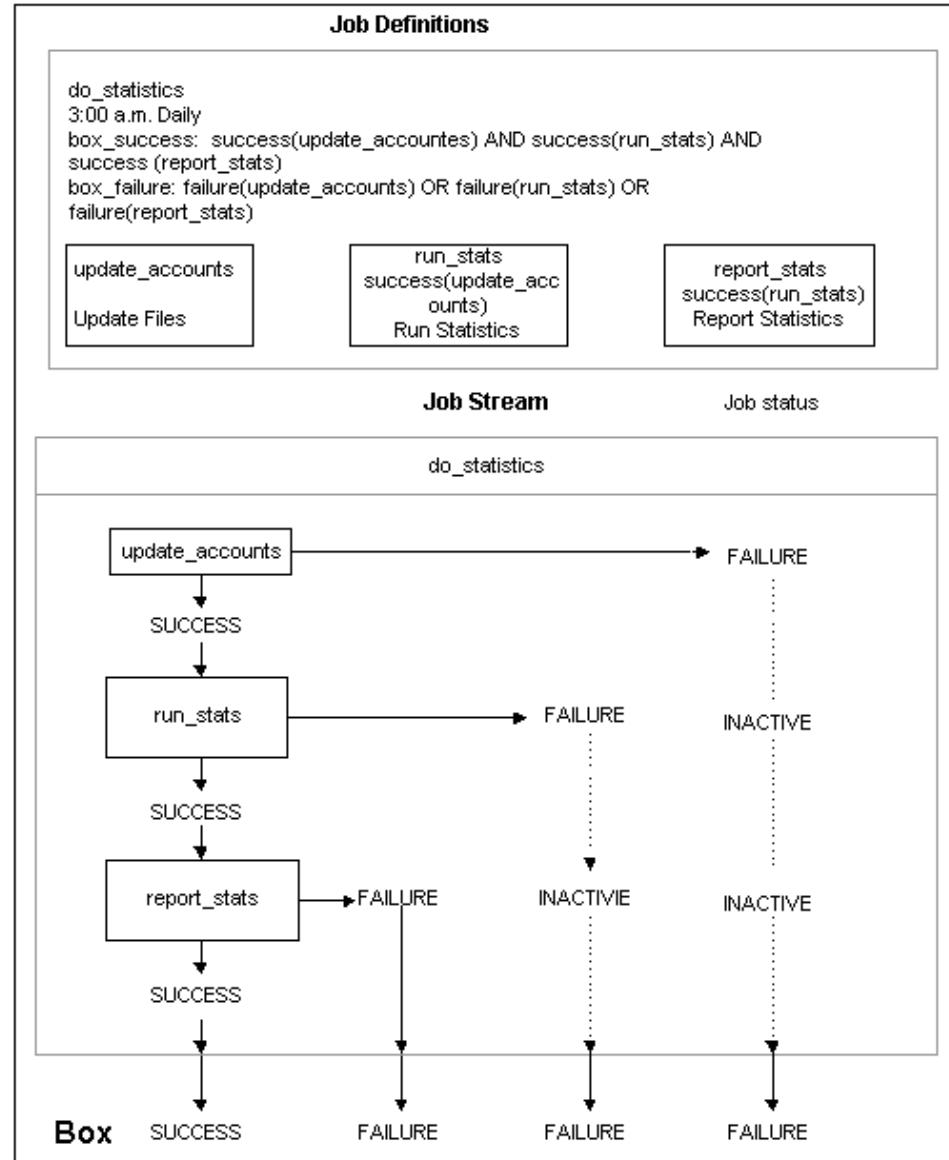
The box job do_statistics runs every day at 3:00 a.m. It contains three jobs. update_accounts updates files; it has no other starting conditions so it starts as soon as the box starts running. run_stats runs statistics; its only starting condition is the success of update_accounts. report_stats reports statistics; its only starting condition is the success of run_stats. No conditions for success or failure of the box have been defined; therefore the default conditions are applied. That is, box success is when all jobs in the box have run and successfully completed; box failure is when all jobs in the box have run and at least one has failed. The box will remain in the RUNNING state until all jobs in the box have run.

The following figure illustrates this job stream logic:



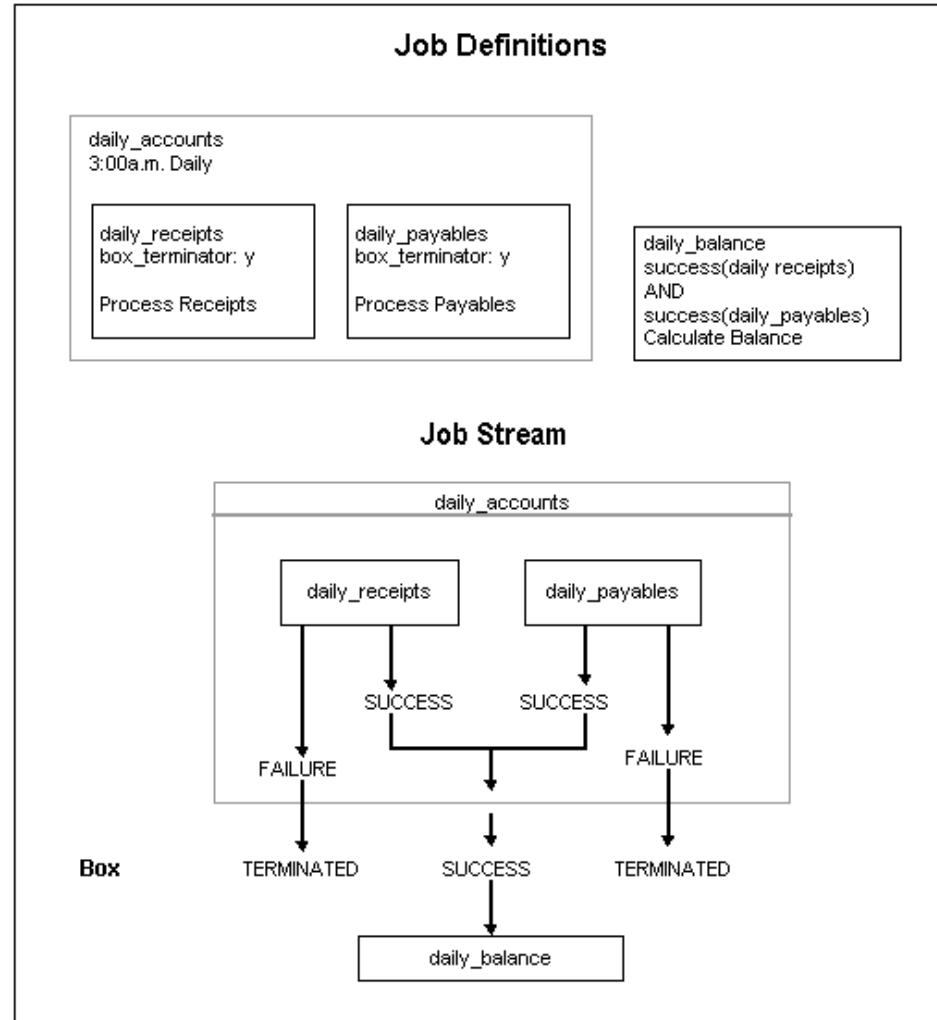
Explicit Box Success and Box Failure

The following figure illustrates box success and failure:



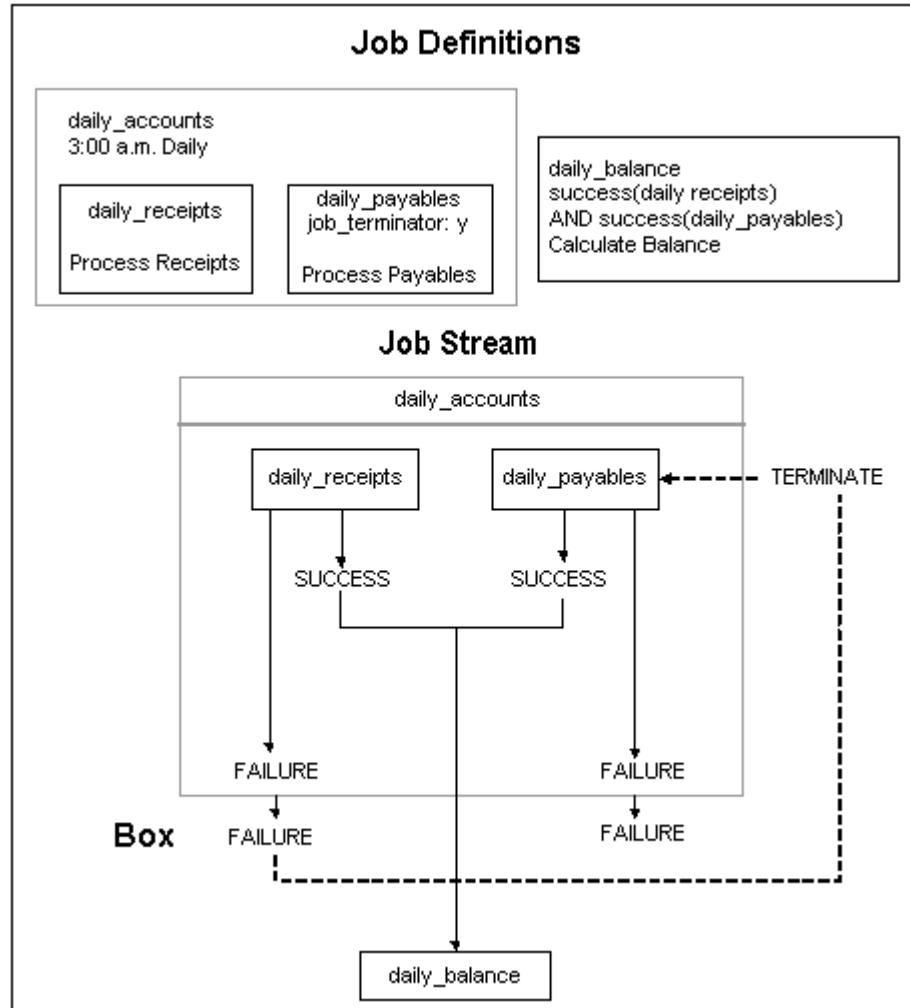
Using the Box Terminator Attribute

The following figure illustrates using box_terminator:



Using the Job Terminator Attribute

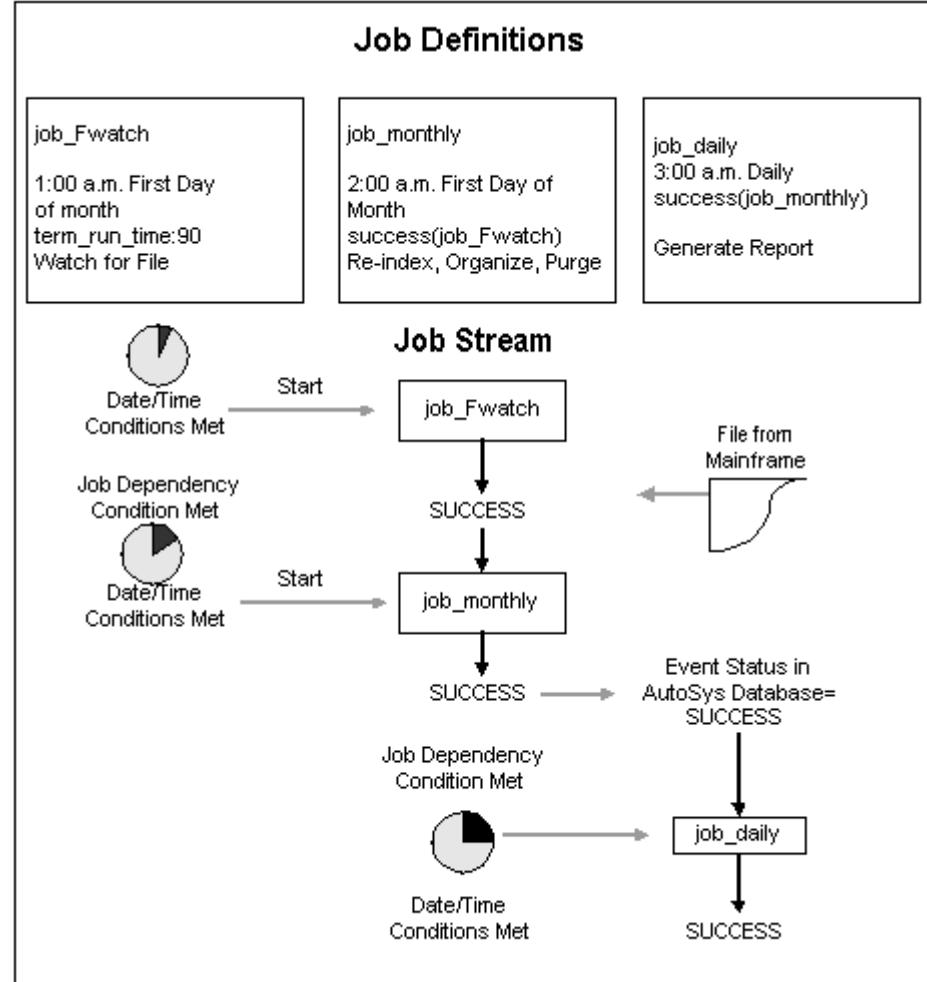
The following figure illustrates using job_terminator:



Advanced Job Streams

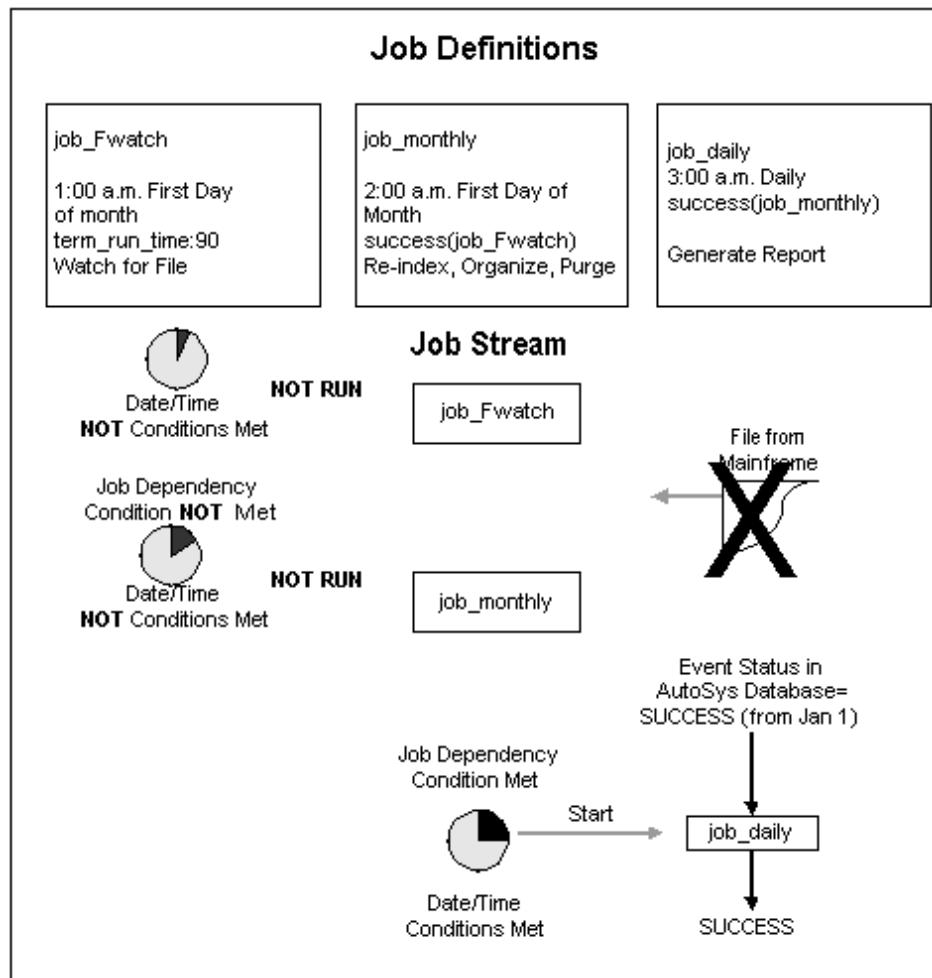
Scenario on the First of the Month

The first time the cycle is run (for example, January 1), statuses behave as expected.



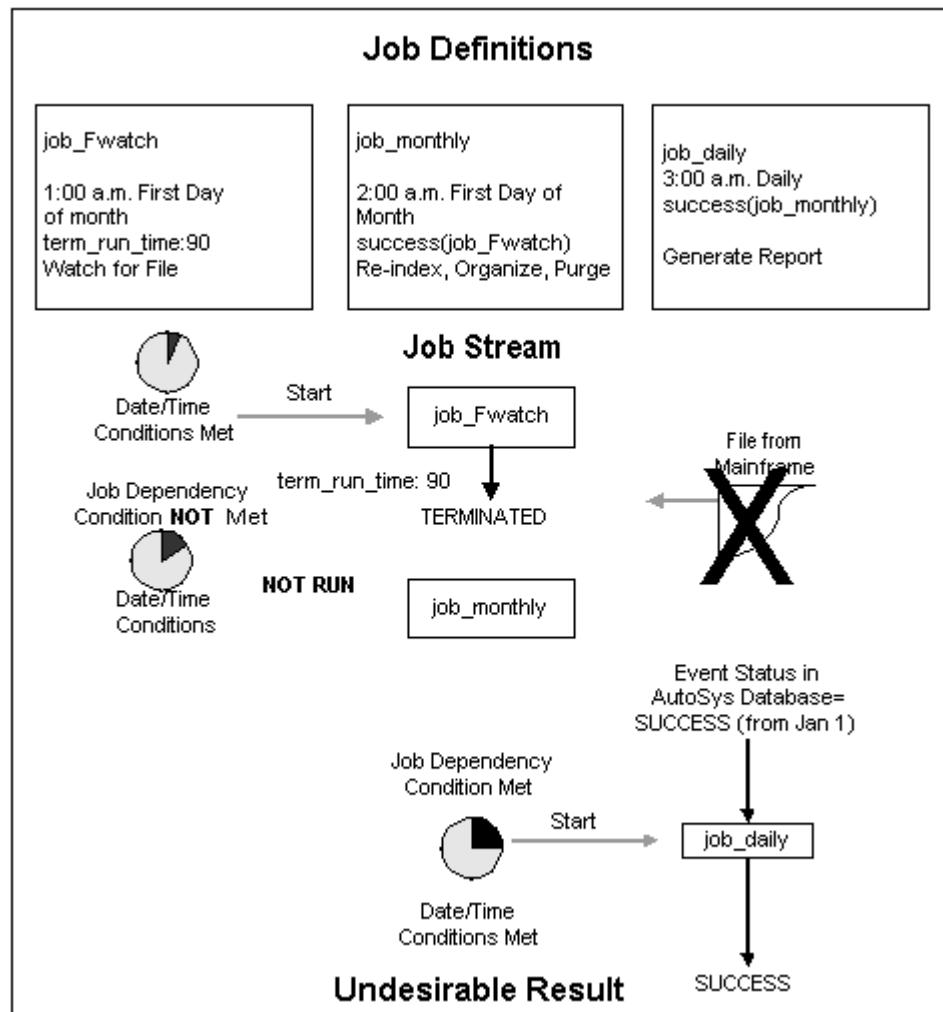
Scenario on the Second of Month

On days of the month other than the 1st, job_Fwatch and job_monthly do not run. They still have a status of SUCCESS in the database from the previous run on the first of the month. As a result, job_daily will still run.



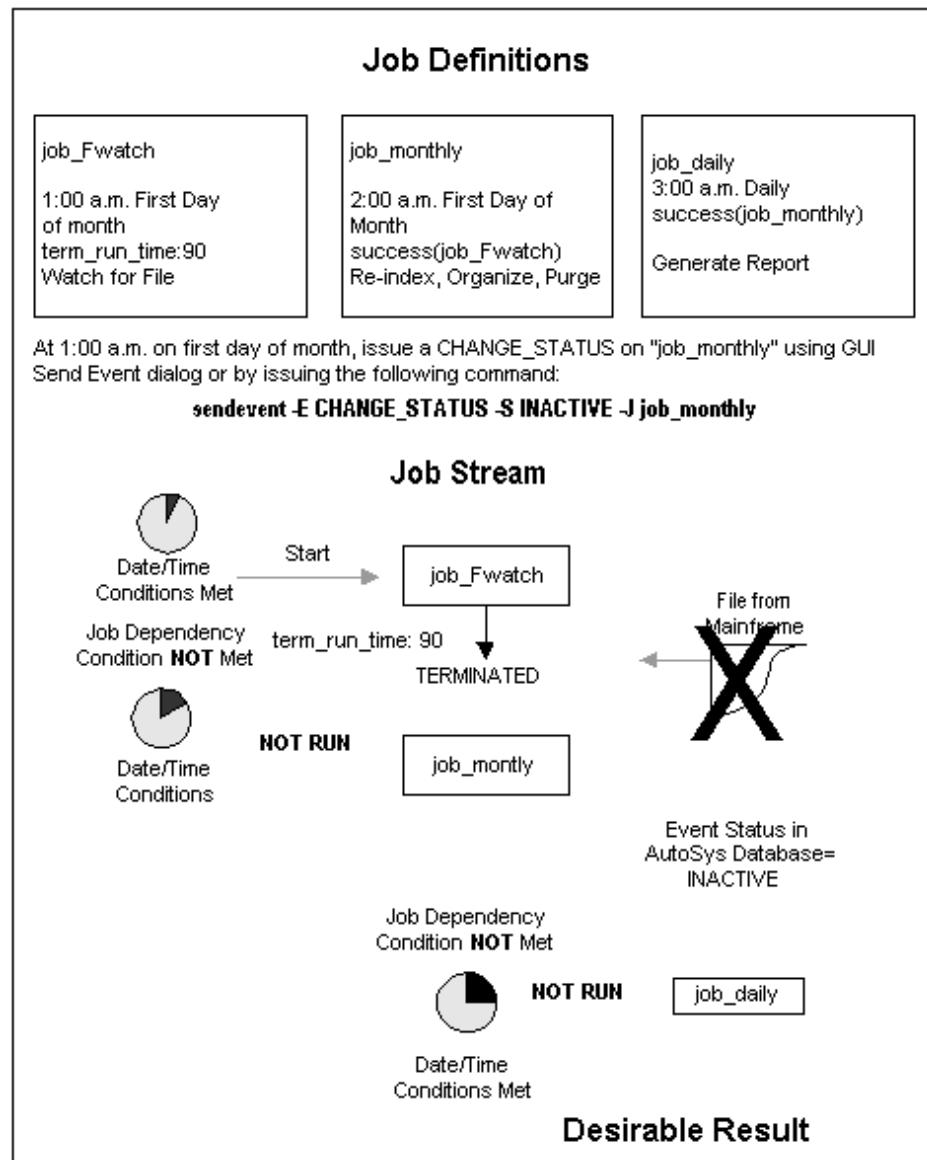
Scenario I on First of the Month

On the first of the next month (for example, February 1), the file from the mainframe fails to arrive; therefore, job_monthly does not run for the month. However, its event status in the database is still SUCCESS from the previous month, and as a result, job_daily runs in error.



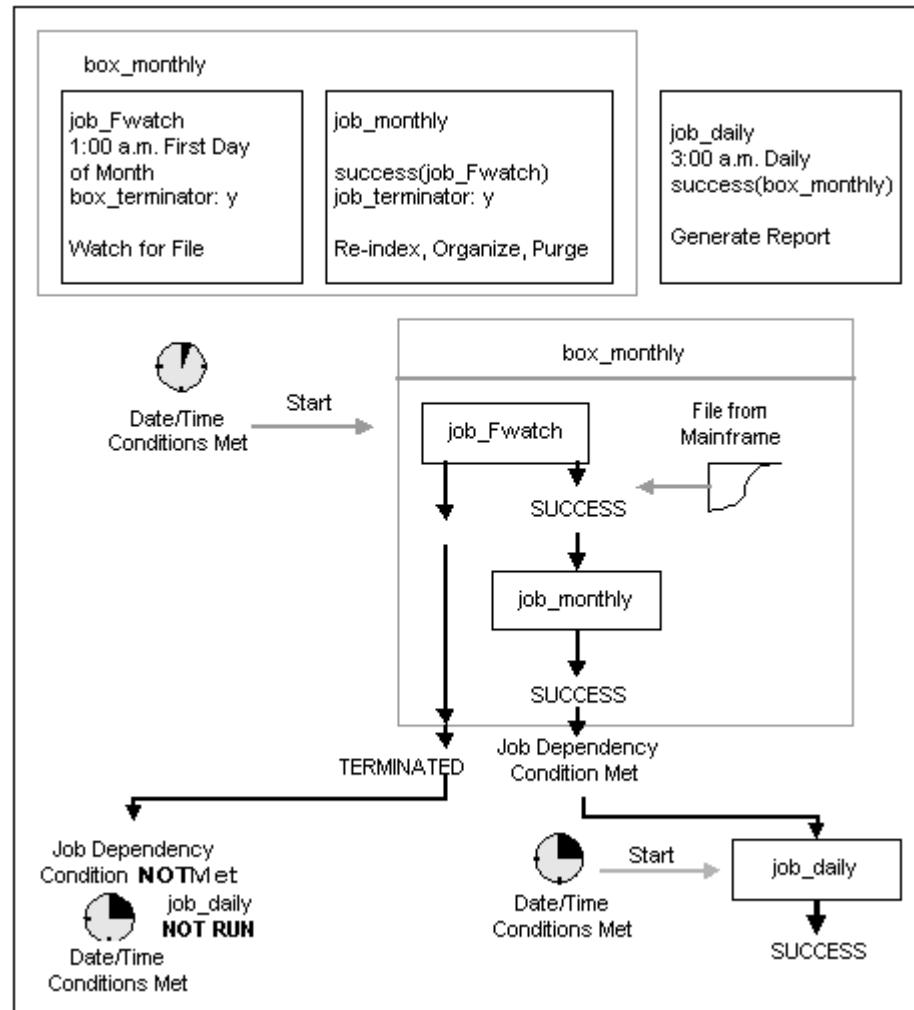
Scenario II on First of the Month

To fix statuses that are time-related, you can use a sendevent command to change them to INACTIVE at the end of their valid time period. You can create another job to do this automatically.



Scenario III on First of the Month

Instead of issuing a sendevent command to change the status of the jobs, you could put the monthly process in a box, and set box_failure or box_terminator appropriately.



Defining Jobs Using the GUI

This chapter describes how to define jobs using the graphical user interface or GUI. It also provides information about creating various types of jobs. It discusses changing and deleting a job and how to set time dependencies.

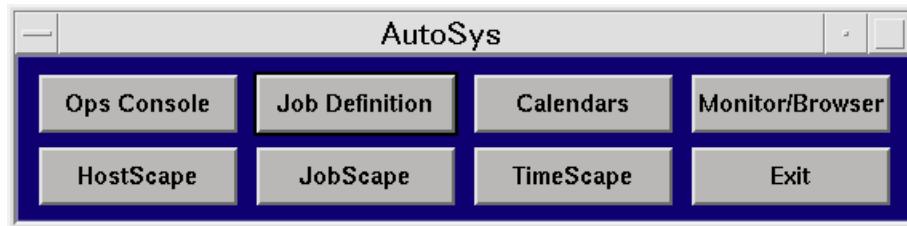
Starting the GUI

To start the GUI:

Enter the following command at the UNIX prompt:

```
autosc &
```

The GUI control panel appears:



GUI Control Panel

The control panel buttons and the actions they perform:

Ops Console

Displays the Job Activity Console, used to monitor jobs and alarms.

Job Definition

Displays the Job Definition dialog, used to define jobs.

Calendars

Displays the Calendar Definition window, used to define run and exclude calendars.

Monitor/Browser

Displays the Monitor/Browser dialog, used to define and run monitors and reports (or browsers).

Exit

Exits the GUI.

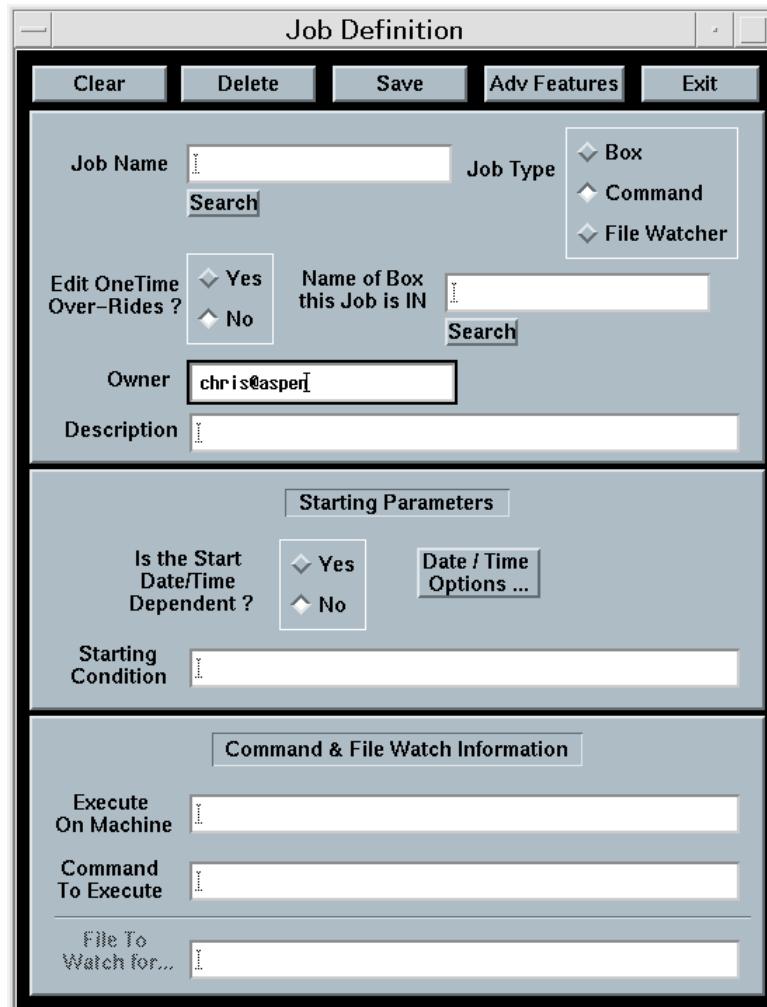
Job Definition Dialogs

Unicenter AutoSys JM provides several dialogs that you can use for defining jobs. You may use up to three, or as few as one dialog, depending on the complexity of the job. At the top of each dialog box is its title, such as Job Definition. This chapter is organized around these dialogs.

The Job Definition Dialog

The Job Definition dialog contains fields representing all the basic information you need to define a job.

When you click the Job Definition in the control panel, the Job Definition dialog appears:



The buttons at the top of the dialog perform the following actions:

Clear

Clears the dialog without affecting the database. Use this button to clear all fields in the dialog (and in memory).

Delete

Deletes the currently displayed job from the database.

Save

Stores the currently displayed job in the database, either modifying a pre-existing job, or creating a new one. It also clears the dialog in preparation for another job definition.

Adv Features

Displays the Job Definition Advanced Features dialog, which is used for all but the simplest of job definitions.

Exit

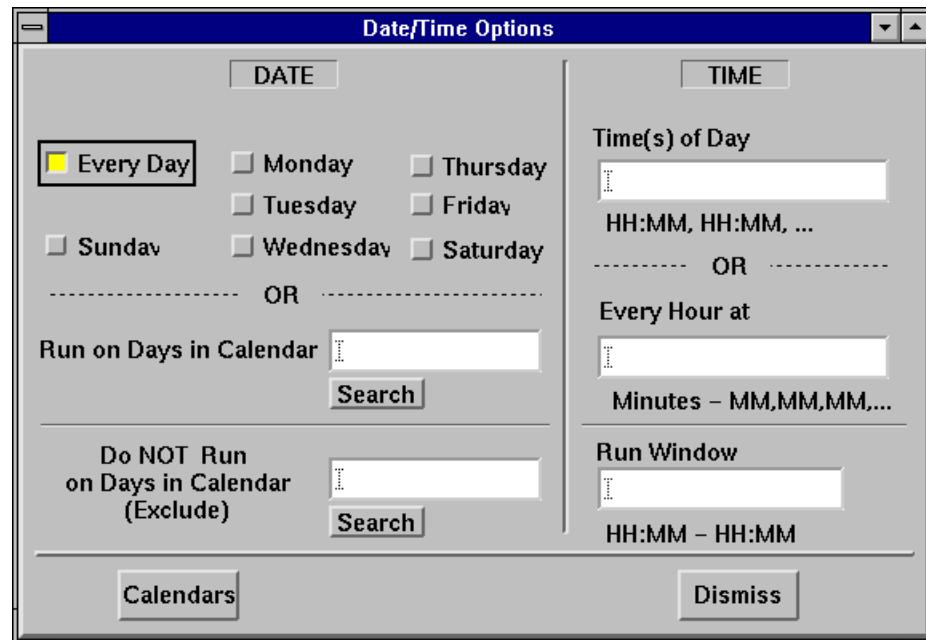
Closes the Job Definition dialog. If Exit is pressed without first pressing Save, the latest changes will not be saved.

The fields in the Job Definition dialog are context sensitive based on the type of job being defined. When you select a Job Type, only the fields appropriate to that type of job are displayed and activated, and the other fields are disabled.

In this dialog, there is a special field named Edit One-Time Overrides? which is used to specify that certain job attributes be applied for the very next run of the job. When the Yes radio button in this field is selected, only those fields that can be used for overrides are active, and the remaining fields are disabled.

Date/Time Options Dialog

In the Starting Parameters region (middle section) of the Job Definition dialog, there is a Date/Time Options button. When you click this button it displays the Date/Time Options dialog, as the following shows:



The buttons at the bottom of the dialog perform the following actions:

Calendars

Accesses the Graphical Calendar facility.

Dismiss

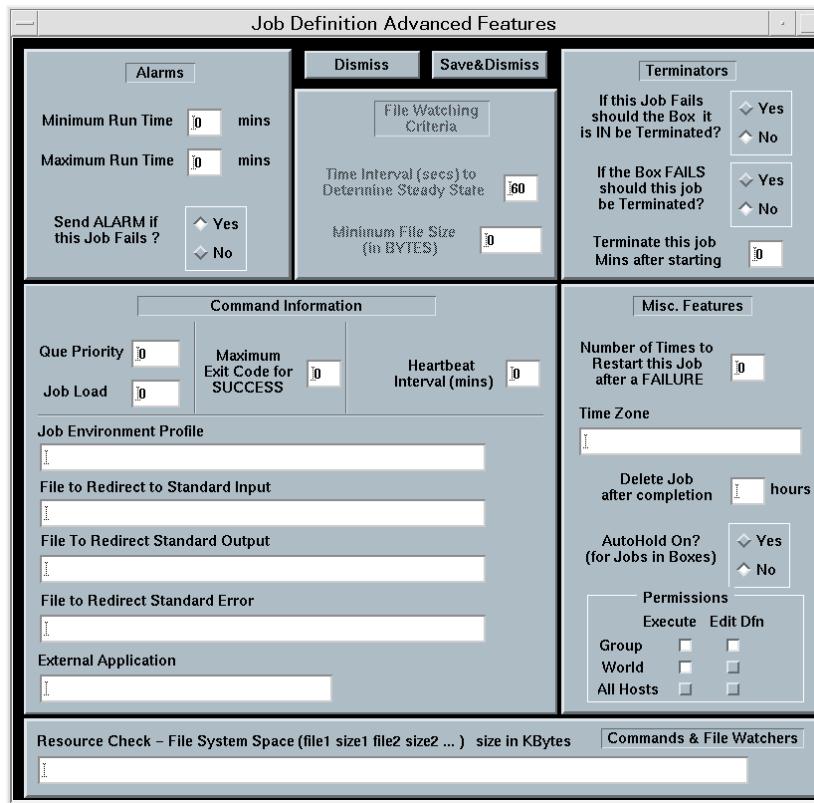
Closes the Date/Time Options dialog.

In addition to setting specific date and time starting conditions, at this dialog you can specify (and search for) any run or exclude calendars that have been defined for the job. To do this, you use one of the provided Search buttons.

All entries made in the dialog are maintained in memory after you close the dialog. They are saved only when you select Save at the Job Definition dialog.

Job Definition Advanced Features Dialog

When you click the Adv Features control button in the Job Definition dialog, it displays the Job Definition Advanced Features dialog, as the following shows:



The Job Definition Advanced Features dialog has fields for all the additional features that can be specified for a job. Many of these fields are organized by job type, and they only pertain to the type of job on which you are working.

Note: The External Application field is disabled. It is a placeholder for future functionality.

The control buttons in this dialog are as follows:

Dismiss

Closes the dialog; any entries made in the dialog are maintained in memory, so that a subsequent Save will save this information.

Save&Dismiss

Closes the dialog and saves the job, including the advanced features information to the database.

Creating a Simple Command Job

The most basic command job requires only a few fields to be entered in the Job Definition dialog. To demonstrate this, we will walk you through the process of defining a simple job named test_run which has no starting parameters. When manually started, this job will only echo a message to standard output.

Create an example job as follows:

In the GUI Control Panel, single-click on the Job Definition button. This action opens the Job Definition dialog.

To define the example job:

1. In the Job Name field, enter the job's name:

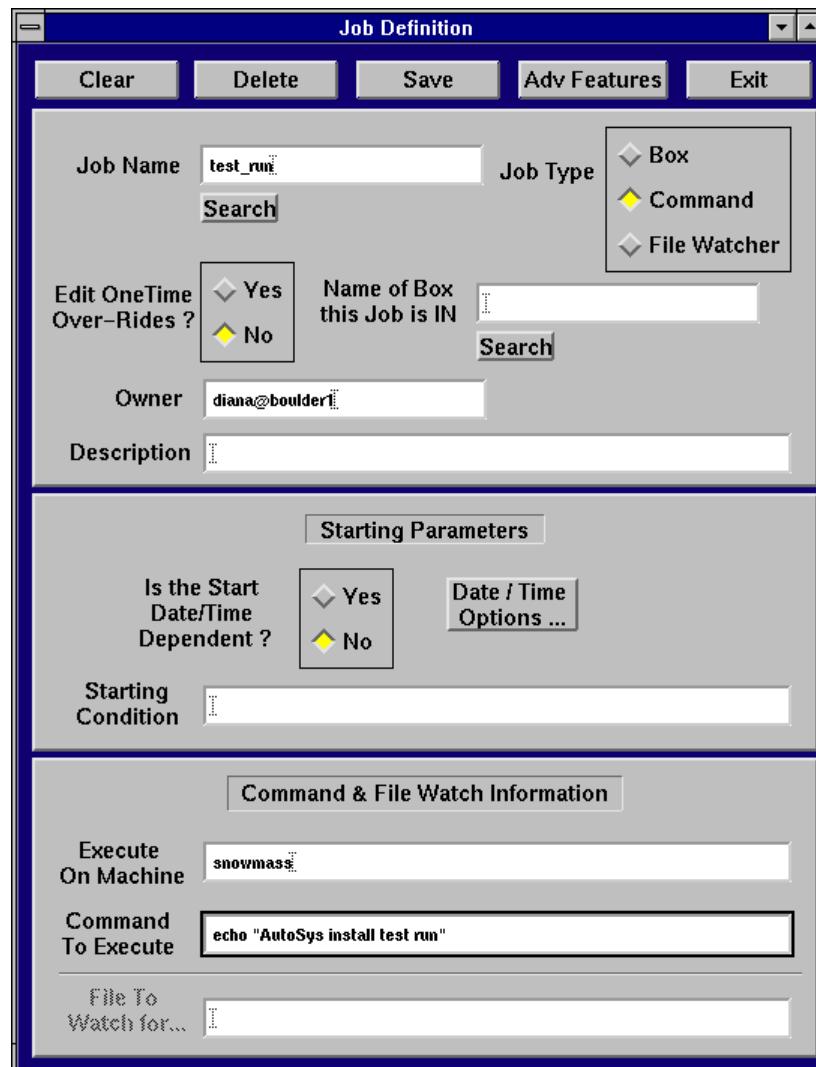
```
test_run
```

2. In the Job Type field, click the Command radio button.
3. In the Execute On Machine field, enter the name of the machine on which the command will be executed. You should enter your own valid, licensed client machine name.
4. In the Command to Execute field, enter the following command to be executed:

```
/bin/echo AUTOSYS install test run
```

Your entries in the Job Definition dialog should look like the following:

Note: The Owner field for the job defaults to the currently logged on user—not the user shown in these examples.



Save the example job by clicking Save.

Leave the Job Definition dialog on your screen to use for the next example.

Creating a File Watcher Job

Next, you will create a File Watcher Job. This job will watch for an end-of-day transaction file called EOD_trans_file. File Watcher jobs, themselves, do not actually execute commands; they are used to signal the arrival of files, and typically set off the execution of a command job.

Using the Job Definition dialog, follow these steps to create a File Watcher Job:

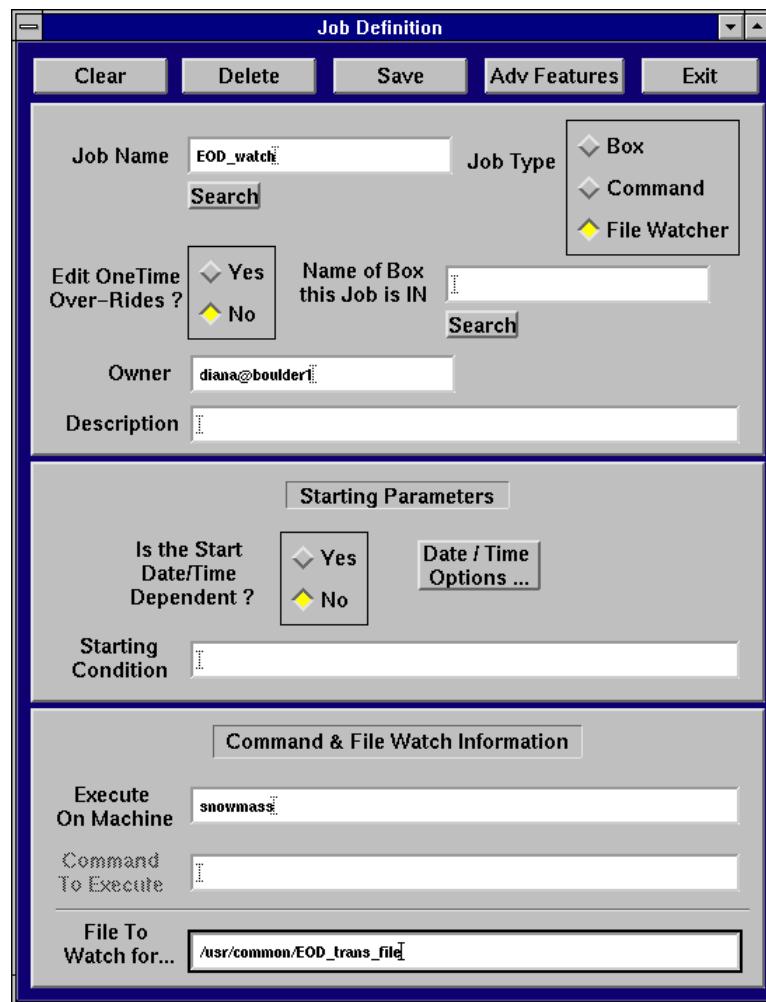
1. In the Job Name field, enter the job name:

EOD_watch

2. In the Job Type field, click the File Watcher radio button.
3. In the Execute on Machine field, enter the name of the machine on which the command will be executed. You should enter your own valid, licensed client machine name.
4. In the File To Watch For field, enter the file name:

/usr/common/EOD_trans_file

Your entries in the Job Definition dialog should look like the following:



Set the file watching criteria as follows:

Click Adv Features at the top of the Job Definition dialog, and the Job Definition Advanced Features dialog appears.

At the top-center of the dialog are the File Watching Criteria. In this region of the dialog, enter the following information:

1. In the Time Interval (secs) to Determine Steady State field, enter the time interval:

60

AutoSys will check for the file's existence every 60 seconds, and it will check if the file has grown between checks—if it has not changed in size, this is called a steady state.

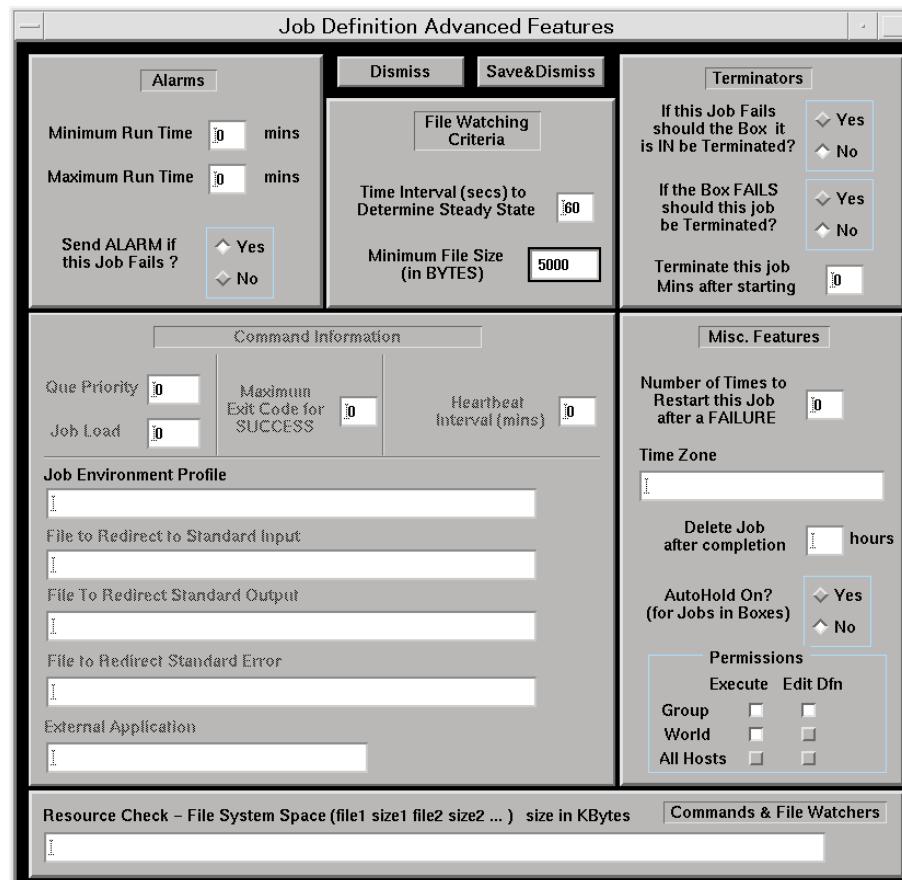
2. In the Minimum File Size (in BYTES) field, enter the minimum file size, which should be reached before the file can be considered complete:

50000

The file must have reached this minimum size, and must have reached a steady state before the file watcher job will complete with a SUCCESS condition.

3. To save the job and dismiss the Job Definition Advanced Features dialog, click the Save&Dismiss button.

The completed dialog should look like the following:



Creating a Dependent Command Job

Now, you will create a command job that is dependent on the successful completion of the file watcher job you just created. The only difference between this command job and the simple command job you created earlier, is that this one will be dependent on another job.

Using the Job Definition dialog, follow these steps:

1. In the Job Name field, enter the job's name:

EOD_post

2. In the Job Type field, click Command.
3. In the Starting Condition field, enter the only starting condition, in this case the successful completion of the file watcher job:

success (EOD_watch)

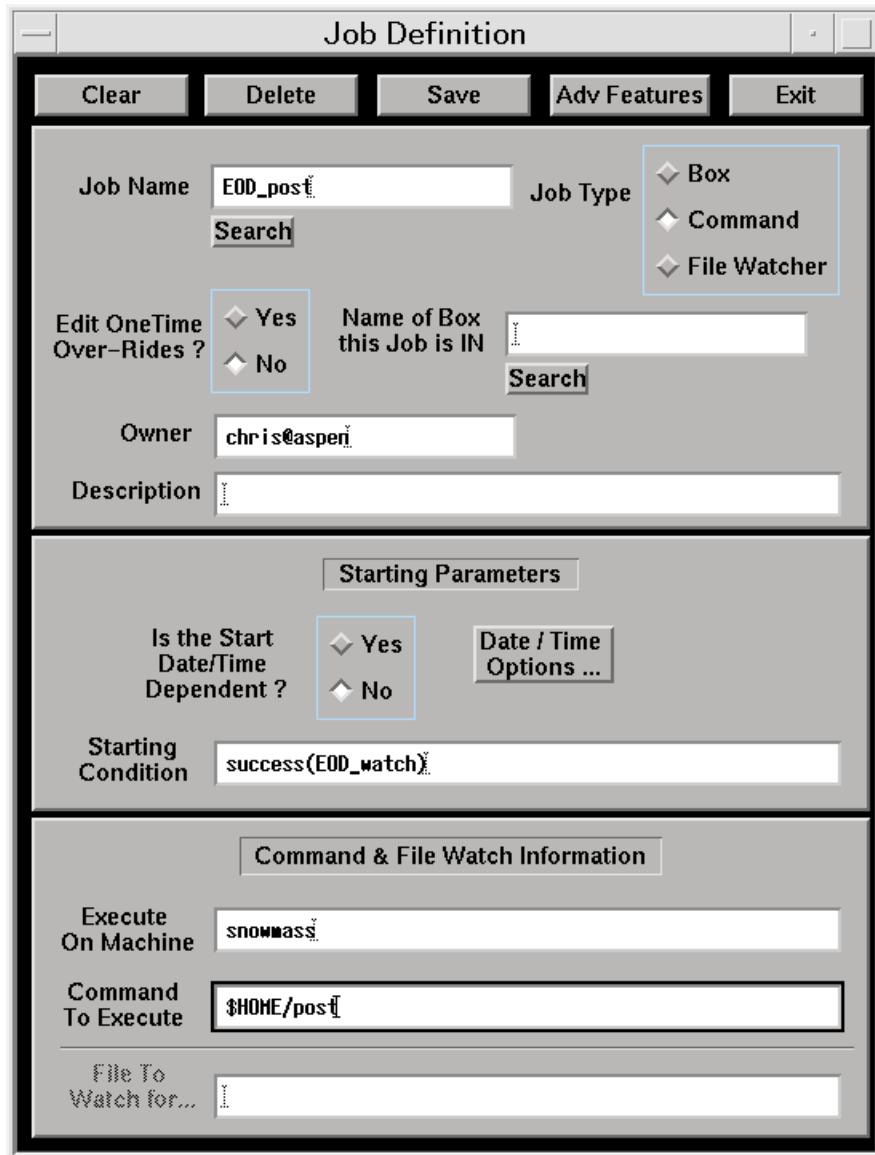
4. In the Execute On Machine field, enter the name of the machine on which the file watcher will run. You should enter your own valid, licensed client machine name.
5. To Execute field, enter the command that is to run when the file watcher completes as follows:

\$HOME/post

The environment variable \$HOME means that the post executable is located in the job owner's home directory.

Note: The job's execution environment is determined exclusively by the profile, which is sourced immediately before the job is started. By default, the file /etc/auto.profile is sourced. This can be overridden by specifying another profile in the File to Define Job Environment field in the Job Definition Advanced Features dialog. For information on the profile job attribute, see the chapter JIL/GUI Job Definitions in the *Unicenter AutoSys Job Management Reference Guide*.

The completed Job Definition dialog should look like the following:



Save the job by clicking Save.

Creating a Box Job

Assume that you want to schedule a group of jobs to all start running once the file watcher completes successfully. Rather than make each job dependent on the file watcher, you can create a box that is dependent on the file watcher, and place all of the jobs in the box. (For detailed information on box jobs, see the chapter [Box Job Logic](#), in this guide.)

Now you will create a box, then change the job you just created to put it in the box, and then make it no longer individually dependent on the file watcher.

In the Job Definition dialog, enter the following information:

1. In the Job Name field, enter the job name:

EOD_box

2. In the Job Type field, click Box.

When you select the box job type, the lowest section of the Job Definition dialog changes from Command & File Watch Information to Box Completion Conditions.

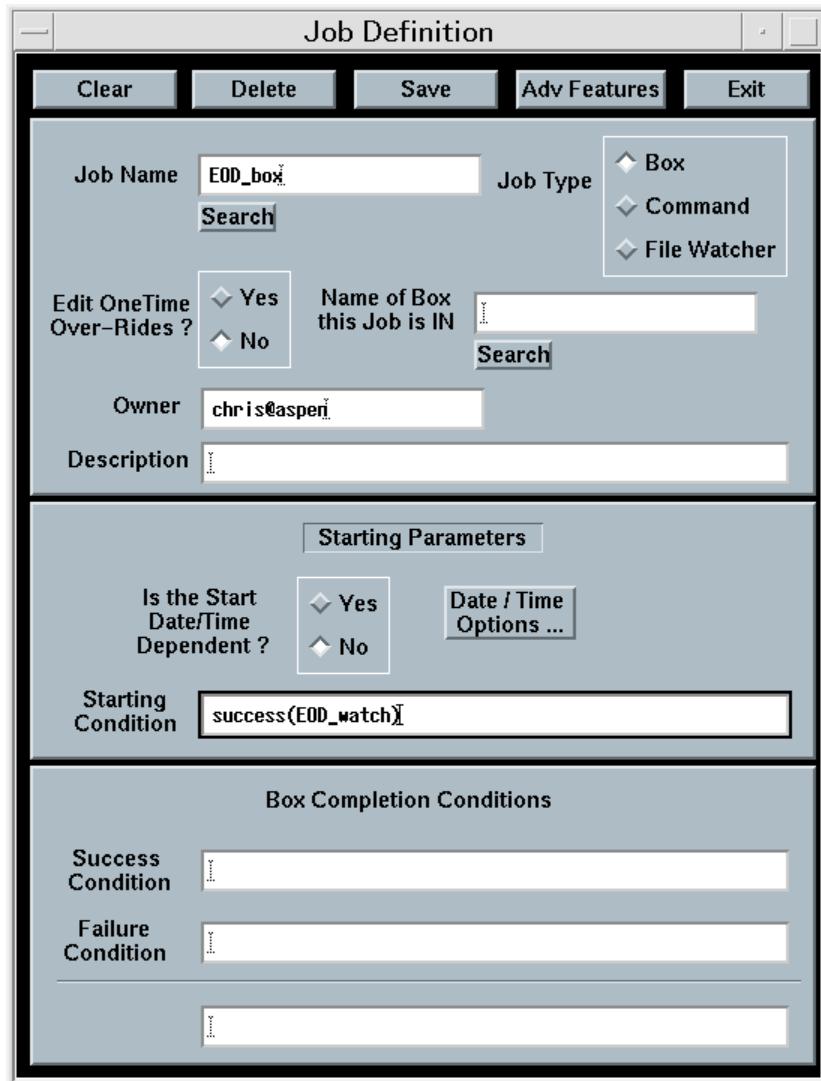
3. In the Starting Condition field, enter the only starting condition, in this case the successful completion of the file watcher job:

success (EOD_watch)

To save the job:

Click the Save button at the top of the Job Definition dialog.

The completed dialog should look like the following:



Changing a Job

This exercise will change an existing job. You should make sure a job is not running before you modify or delete it.

In this exercise, you will place the EOD_post job, created previously, in the newly created box. To load an existing job into the Job Definition dialog, you can either enter its name explicitly in the Job Name field and click the Search button, or you can use the Search Facility. For this exercise, you will use the search facility. You can enter some portion of the job name, followed by the percent (%) wildcard character. The percent (%) character will match any string of one or more characters in the job name. For instance, %box% will match any job name with the string box anywhere in the name.

Use the search facility by:

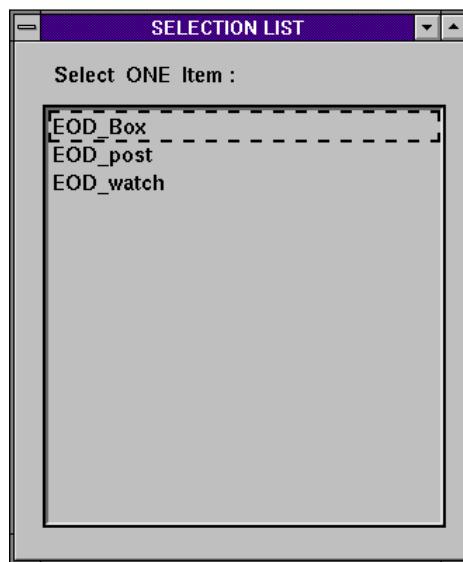
1. In the Job Name field, enter:

EOD%

2. Click Search below the Job Name field.

A Selection List Box similar to the one shown in the following graphic appears containing all the jobs currently defined in the database that start with the string EOD. (The list box shown below may not match yours exactly, since you can have other jobs defined.)

3. Typing just the percent (%) wildcard character will display all the jobs defined in the database.



4. Double-click the desired job's name, in this case, EOD_post.

This will automatically dismiss the Selection List Box and display the requested job in the Job Definition dialog. If the job you wanted was not in the list, you could click the Cancel button to dismiss the Selection List Box without making a selection.

Place the EOD_post job in the box:

1. In the Name of Box this Job is IN field, enter the box name:

EOD_Box

This field also has a search facility, which works the same as the Job Name search facility, complete with wildcarding using the percent (%) character.

2. In the Starting Condition field, delete the string:

success (EOD_watch)

This starting condition has been assigned to EOD_Box. Now that this job is in a box, it will inherit the starting condition of the box.

3. Click Save at the top of the dialog to save the changes.

Setting Time Dependencies

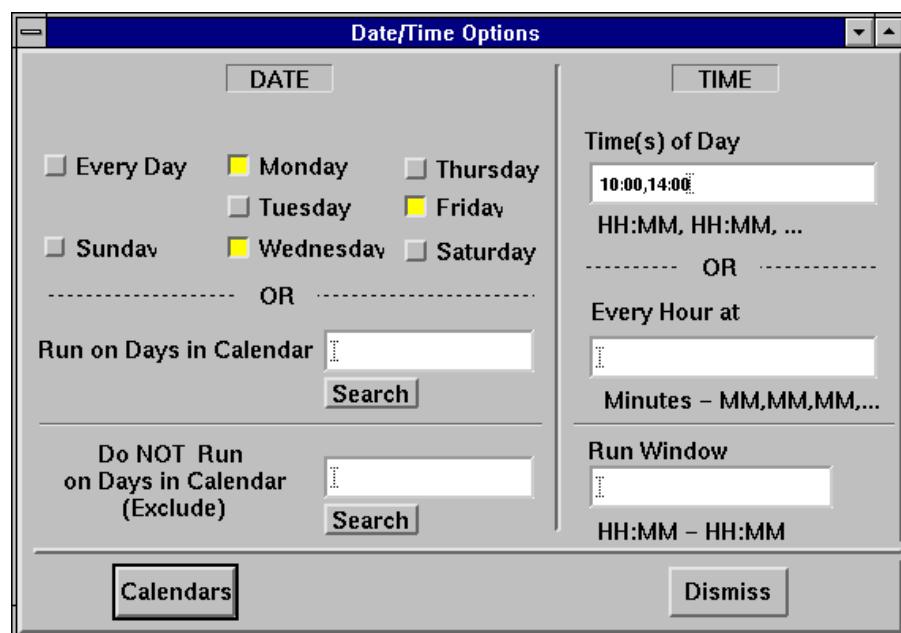
The test_run job you created at the beginning of the chapter will only be executed if it is started manually.

Set the job to run on certain days at certain times, such as 10:00 a.m. and 2:00 p.m. on Mondays, Wednesdays, and Fridays:

1. In the Job Name field, enter the job name to be modified:

test_run

2. Click the Search button to display the specified job.
3. In the Starting Parameters region of the Job Definition dialog, locate the Is the Start Date/Time Dependent? field and click Yes.
4. Click the Date/Time Options button.
5. The Date/Time Options dialog appears:



6. In the Date region of the dialog, select the days on which the job is to run. In this case, click the Monday, Wednesday, and Friday buttons.

These buttons can be toggled on and off and are not mutually exclusive.

7. In the Time region of the dialog, select the times when the job is to be run, in this case, click the Times of day field, then enter:

10:00, 14:00

Unlike in JIL, the times do not have to be enclosed in quotes when they are entered in the GUI.

8. To close the Date/Time Options dialog, click Dismiss.

The information is retained in memory until the job is either saved to the database or cleared.

To save the new start date/time information for this job in the database click Save in the Job Definition dialog.

Note: If a job runs daily at the same time (example: 12:00) and you EDIT this job definition and save this job at (11:59), the job will not run today but will run tomorrow at (12:00).

When a start time job definition is written to the database within one minute of the current runtime, the start time will be placed in the future, meaning tomorrow. However, if the start time is two minutes or greater from the current save time the job will run today.

Additional Time Setting Features

You can base the time settings for a job on a specific time zone. To do this, you enter a valid time zone in the Time Zone field in the Job Definition Advanced Features dialog. For details on specifying a time zone in a job definition, see *timezone* in the chapter *JIL/GUI Job Definitions* in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

To have a job run at specific minutes past every hour, as opposed to specific times of day, specify the minutes past every hour in the Every Hour at field.

If you want to run a job every day, rather than only on specific days, you click the Every Day button instead of the individual buttons for each day.

If you want to schedule a job for specific dates, rather than specific days of the week, you can specify a custom calendar name in the Run on Days in Calendar field. If the custom calendar does not already exist, create it by clicking the Calendars button, which displays the Graphical Calendar Facility.

You can specify a custom calendar defining days on which the job must not run. You specify this calendar in the Do NOT Run on Days in Calendar (Exclude) field.

To view the calendars defined, use the Search buttons beneath each of these fields. These buttons display a list box from which you can select a pre-defined calendar. To define a calendar on the fly, click Calendars, which displays the Graphical Calendar Facility.

Deleting a Job

Now you will delete the test_run job, which you just modified. You delete all jobs, regardless of type, in exactly the same way. To delete a job, you can either enter its name explicitly in the Job Name field, or use the search facility. In this final exercise, you will use the search facility. You can enter some portion of the job name, followed by the percent (%) character, or just enter the (%) character alone, for a global search.

Delete a job by doing the following:

1. In the Job Name field, enter:

%

2. Click Search to initiate the search.

A Selection List Box appears, allowing you to select the desired job.

3. Double-click the desired job's name, in this case, test_run. This will automatically dismiss the Selection List Box and display the requested job in the Job Definition dialog.
4. Verify that the test_job job is displayed in the Job Definition dialog.
Note: No confirmation dialog appears when you delete a job using the GUI, so ensure that you are deleting the right job.
5. To delete the job from the database, at the top of the Job Definition dialog, click Delete.

Deleting a Box Job

When using the GUI to delete a box job, Unicenter AutoSys JM will delete the box, and will recursively delete all jobs within that box.

Using the GUI, there is no way to delete just the box itself and not its contents.

However, with JIL, you can use the `delete_job` sub-command to delete the box while leaving its contents intact (see Deleting a Job in the chapter Defining Jobs Using JIL).

The GUI includes more fields than are discussed in this chapter. All fields are discussed in detail in the chapter JIL/GUI Job Definitions in the *Unicenter AutoSys Job Management Reference Guide*.

Specifying One-Time Job Overrides

Using the GUI, you can specify a job override for the next run of a particular job; that is, the next time a job runs, you can change its behavior. When using the GUI to specify job overrides, you enter new values into the GUI fields, or enter NULL in a GUI field to turn off a value.

Job overrides are applied for the next run of the job only. If a RESTART event is generated because of system problems, Unicenter AutoSys JM will reissue a job override until the job actually runs once, or until the maximum number of retries limit is met. After this, the override is discarded.

Note: The maximum number of job restarts after system or network failure is specified in the `MaxRestartTrys` parameter in the configuration file. One-time job overrides will be applied to jobs restarted due to system problems, but will not be applied to jobs restarted because of application failures. System problems include such things as machine unavailability, media failures, or insufficient disk space. Application failures include such things as inability to read or write a file, command not found, exit status greater than the defined maximum exit status for success, or various syntax errors.

You cannot submit an override if it results in an invalid job definition. For example, if a job definition has only one starting condition, `start_times`, you cannot set the `start_times` attribute to NULL because removing the start condition makes the job definition invalid (no start time could be calculated).

The GUI dialogs, regions, and fields that you can use in a job override specification (definition) are listed in the following table:

Dialog	Region	Field
Job Definition	Job Definition	Edit Onetime Over-Rides?
	Starting Parameters	Starting Condition
		Is Start Date/Time Dependent?
Advanced Features	Command & File Watch Information	Execute On Machine
		Command To Execute
		File To Watch for
File Watching Criteria	Alarms	Minimum RunTime
		Maximum RunTime
		Time Interval (secs) to Determine Steady State
Terminators		Minimum File Size (In BYTES)
		Terminate this job Mins after starting
Command Information	Command Information	Job Environment Profile
		File To Redirect to Standard Input
		File To Redirect to Standard Output
		File To Redirect Standard Error
		External Application
Misc. Features	Misc. Features	Number of Times to Restart this Job after a FAILURE
		Delete Job hours after Completion
		AutoHold On?

Dialog	Region	Field
Date/Time Options	DATE	(all date settings)
		Run on Days in Calendar
		Do NOT Run on Days in Calendar (Exclude)
TIME		(all time settings)
		Run Window

Setting Job Overrides

Set job overrides with the following:

1. At the Job Definition dialog, enter the name of the desired job and click Search.
2. In the Edit OneTime Over-Rides? field, click Yes. Now, the dialog is in job override mode, and only the available fields for specifying job overrides are active.
3. At the appropriate dialogs and fields, specify the overrides you want.

Save the job overrides with the following:

Click Save at the top of the Job Definition dialog. This action returns you to job definition mode.

Delete the job overrides:

1. At the Job Definition dialog, enter the name of the job for which you want to delete the overrides and click the Search button.
2. In the Edit OneTime Over-Rides? field, click Yes.
3. Click Delete. This action deletes the job overrides.

Customizing the Job Definition GUI

A set of X resources you can use to customize the appearance and behavior of the Job Definition GUI. In particular, these resources allow you to control:

- The time interval after which the Monitor/Browser GUI will drop the connection to the database.
- The Monitor/Browse GUI icon text and the Monitor/Browse title bar text.

Descriptions of the resources which can be customized are given following. All of these can be set by modifying the X resource file autosc. The X resources files reside in the local app-defaults directory, which varies across platforms. It is usually in /usr/lib/X11/app-defaults or /usr/openwin/lib/app-defaults. If you are not sure which directory these files are in, ask your system administrator.

Individual users can have their own copy of the X resources files in their \$HOME directory, which will take precedence over the app-defaults files.

For most operating systems, if you are exporting the display to another machine you must edit the appropriate files in the app-defaults directory on the local machine.

For Sun Solaris, you must edit the files in the /usr/lib/X11/app-defaults and /usr/openwin/lib/app-defaults directories. The files in /usr/lib/X11/app-defaults control the resources when you export the display.

Database Connection Time-Out Interval

The following resource controls the time interval after which the Job Definition GUI will drop the connection to the database.

```
! TimeOut to drop DB connections (in minutes)
*DBDropTime: 400
```

If DBDropTime is set to zero, the connection is dropped immediately after the database query has completed. A value greater than or equal to 5 means that the GUI will automatically drop all database connections if the database has not been accessed in the last DBDropTime minutes. (Values of 1 to 4 are invalid). A new database connection will subsequently be established when required. If DBDropTime is greater than 360, the connection to the database is maintained until the GUI screens are exited.

Job Definition Title Bar Text and Icon Text

The following resources specify the title bar text and the icon text. By default the title bar text is Job Definition and the icon text is JobDef.

```
Autosc.shellTitleJobDef:  
Autosc.iconTitleJobDef:
```

Note: When changing icon text, be sure the length of the new text string does not exceed the recommended maximum length for icon title text for your windowing system. Some window managers can display long icon text strings, while others will truncate them. Ensure the text string you specify for your icons displays appropriately. Also, some window managers allow you to change the size of icons and icon text font.

Defining Jobs Using JIL

This chapter describes how to define jobs using the Job Information Language or JIL. It also provides information about creating various types of jobs. It discusses changing and deleting a job, and how to set time dependencies. An example JIL script is provided.

Job Information Language (JIL)

Job Information Language (JIL) is a scripting language, which provides a way to specify how jobs should behave. JIL scripts contain one or more JIL sub-commands and one or more attribute statements; these elements constitute a job definition.

JIL Syntax Rules

When writing a JIL script, you must follow the syntax rules listed following.

Rule 1

Each sub-command uses the following form:

sub_command: job_name

where:

sub_command

Indicates one of the sub-commands listed in the table in JIL Sub-commands in this chapter.

job_name

Specifies the user-specified name of the job to receive action.

Rule 2

Each sub-command can be followed by one or more attribute statements. These statements can occur in any order, and are applied to the job specified in the preceding sub-command. A subsequent sub-command begins a new set of attributes for a different job. The attribute statements have the following form:

attribute_keyword: value

where:

attribute_keyword Indicates one of the legal JIL attributes.

value Specifies the setting to be applied to the attribute.

Rule 3

Multiple attribute statements can be entered on the same line, but the lines must be separated by at least one space.

Rule 4

A box must be defined before the jobs can be placed in it.

Rule 5

Legal *value* settings can include any of the following characters: uppercase and lowercase letters, hyphens, underscores, numbers, colons (if the colon is escaped with quotes or a preceding backslash), and the at character (@).

Rule 6

Any colons used in an attribute statement's *value* setting must be escaped, because JIL parses on the combination of keyword followed by a colon. For example, to specify the time to start a job, specify 10:00. The colon can also be escaped with a preceding backslash (\) as in 10\:00.

Rule 7

Comments are indicated using one of the following two methods:

- An entire line can be commented by placing a pound sign (#) in the first column.
- The C programming syntax used for beginning a comment with a slash star /*) and ending it with a star slash (*)/ can be used; this allows comments to span multiple lines. The following command is an example:

```
/* this is a comment */
```

JIL Subcommands

JIL subcommands are used to create, modify, override, or delete a job definition. These subcommands are listed in the following table.

Subcommand	Action
insert_job	Add a new job.
insert_machine	Add a new machine.
update_job	Edit fields on an existing job.
delete_job	Delete an existing job from the database.
delete_box	Delete an existing box job, and recursively delete all the jobs, which are contained in the box.
override_job	Apply overrides on indicated job attributes for the next run of this job.

Submitting Job Definitions

As stated earlier, a completed JIL script is called a job definition. This job definition must be submitted to the database before the job it defines can be run. You can submit a job to the database using one of the following methods:

- Submit it by redirecting a JIL script file to the `jil` command, for example:

```
jil < my_jil_script
```

- Interactively submit it by issuing the `jil` command and pressing Enter; then entering JIL statements at the provided command prompts:

```
jil>>
```

To exit interactive mode, enter `exit` at the prompt, or press `Ctrl+D`.

Both of these methods are analogous to saving a job definition in the GUI, using the Job Definition dialog.

To specify the instance to which definitions are to be sent and applied, you can use the `-S autoserv_instance` argument to the `jil` command. For single instance environments, the command will default to the only available instance.

For the `jil` command to work properly, the correct environment variables must be assigned. For more information about these variables, see the *Unicenter AutoSys Job Management for UNIX Installation Guide*. For more information about the `jil` command, see its definition in the chapter Commands in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

Running JIL

After a job definition has been submitted to the database, it will be started according to the starting parameters specified in its JIL script. That is, the event processor will continually poll the database and when it determines that the starting parameters have been met, it will run the job.

If a JIL script does not specify any starting parameters for a job, the job will not be started automatically by the event processor; it will start only if you issue the `sendevent` command. For example, assume a job named `test_install` has no starting parameters specified in its JIL script. The only way to start it would be to issue the following command:

```
sendevent -E STARTJOB -J test_install
```

This command tells the event processor to start the job named `test_install`.

For more information about the sendevent command, see its definition in the chapter Commands in the *Unicenter AutoSys Job Management Reference Guide*.

Creating a Simple Command Job

To create the most basic command job, you only need to specify a few attributes. For example, the JIL script required to define a simple command job named test_run as follows:

```
insert_job: test_run
job_type: c /*(optional, it is the default) */
machine: tibet
command: /bin/touch /tmp/test_run.out
```

This JIL script instructs:

- To add a new job named test_run.
- That the new job is a command job.
- To run the job on the client machine named tibet.
- To execute the UNIX /bin/touch command on the file named /tmp/test_run.out.

Creating a File Watcher Job

File watcher jobs do not execute commands themselves; they are used to signal the arrival of files, and typically set off the execution of a command job. For example, the JIL script required to define a file watcher job named EOD_watch as follows:

```
insert_job: EOD_watch
job_type: f
machine: tibet
watch_file: /tmp/EOD_trans_file
watch_interval: 60
watch_file_min_size: 50000
```

This JIL script instructs:

- To add a new job named EOD_watch.
- That the new job will be a file watcher job.
- To run the job on the client machine named tibet.
- To watch for an end of day transaction file named EOD_trans_file in the /tmp directory.
- Check the file every 60 seconds.
- Determine if the file has reached the minimum file size of 50,000 bytes.

Until the minimum file size of 50,000 bytes has been reached, the file will not be considered as complete. When the file reaches this minimum size and does not change between check intervals (60 seconds in this example) it is considered complete (also known as steady state). When this occurs, the file watcher job will end with a SUCCESS condition.

Creating a Dependent Command Job

Command jobs can be dependent on the successful completion of other jobs, such as the file watcher job created in the previous section. The only difference between a dependent command job and a simple command job is its dependency on another job.

For example, the JIL script required to define a dependent command job named EOD_post is given following. EOD_post will be specified to run on the same machine as the file watcher job created in the previous section, since it presumably will need the watched-for file to process. And, it will be dependent on the success of the file watcher job.

```
insert_job: EOD_post
job_type: c
machine: tibet
condition: success(EOD_watch)
command: $HOME/post
```

This JIL script instructs:

- To add a new job named EOD_post.
- That the new job will be a command job.
- To run the job on the client machine named tibet.
- To run the job only if the file watcher job named EOD_watch completes with a SUCCESS status.
- To source the /etc/auto.profile file (sources this file by default), and then, to run the job named post located in the job owner's home directory.

Note: The job's execution environment is determined exclusively by the profile, which is sourced immediately before the job is started. By default, the file /etc/auto.profile, on the client machine, will be sourced. This can be overridden by specifying another profile by using the profile attribute.

For information on the profile job attribute, see its entry in the chapter JIL/GUI Job Definitions in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

Creating a Box

Box jobs are a convenient way to start multiple jobs. When you put jobs in a box, you only have to start a single job (the box) in order for all the jobs in the box to start running.

Assume you want to schedule a group of jobs to all start running once the file watcher completes successfully. Rather than make each job dependent on the file watcher, you can create a box that is dependent on the file watcher, and place all of the jobs in the box.

Now you will create a box, then change the job you just created to put it in the box, and then make it no longer individually dependent on the file watcher. The JIL script required to define a box job named EOD_box as follows:

```
insert_job: EOD_box
job_type: b
condition: success(EOD_watch)
```

This JIL script instructs:

- To add a new job named EOD_box.
- That the new job will be a box job.
- To run the job only if the file watcher job named EOD_watch completes with a SUCCESS status.

For information on box jobs, see the chapter [Box Job Logic](#), in this guide.

Adding Machines

The `insert_machine` sub-command adds a new machine definition to the database for one of the following:

- Real machine
- Virtual machine
- NSM
- Universal Job Management Agent
- Unicenter AutoSys JM Connect

The machine type can be specified as either `r` for real, `v` for virtual, `n` for Windows, `t` for NSM or a Universal Job Management Agent, and `c` for Unicenter AutoSys JM Connect. The component real machines in a virtual machine definition must be all of the same type, for example, all UNIX machines or all Windows machines (not a mix).

If the machine being defined is a virtual machine, the `insert_machine` sub-command is followed by one or more machine attributes that specify real machines.

```
jil
insert_machine: (machine_name bocovic)
Type: t
Exit:
Database change was successful.
```

`machine_name`

Specifies the unique name of the machine to be defined. It can be from 1-30 alphanumeric characters, and is terminated with white space; embedded blanks and tabs are illegal.

The default type is UNIX (`n` or `v`), if no type is specified.



Any machine accessible through the TCP/IP protocol can be specified in the machine attribute of a job; it need not be explicitly defined using the `insert_machine` command. However, any undefined machine will have a default factor of 1.0 and no `max_load`, meaning that there will be no limit on the job load assigned to it. ■



Any machine defined in the /etc/hosts file on the machine running the Event Processor can be specified in the machine attribute of a job; it need not be explicitly defined using the insert_machine command. However, any undefined machine will have a default factor of 1.0 and no max_load, meaning that there will be no limit on the job load assigned to it. ■

Changing a Job

To place an existing job in a box, you need to change the EOD_post command job that was created previously. You will place the EOD_post job in the newly created box.

You should make sure a job is not running before you modify or delete it.

To change a job, you can either use the update_job subcommand, or you can delete the job definition, using the delete_job subcommand, then redefine the job using the insert_job subcommand. The latter scenario is particularly useful when many non-default attributes have been specified, and you want to unset them rather than reset them; in other words, you want to deactivate them. However, you will have to respecify any of the attributes that need to remain the same.

So, in the example following, you will use the update method. The JIL script required to change the EOD-post job and to put it in the EOD_box as follows:

```
update_job: EOD_post
condition: NULL
box_name: EOD_box
```

This JIL script instructs:

- To update the job named EOD_post.
- Remove the starting condition from the job definition, because the job will inherit the starting condition of the box in which it is placed.
- Put the job named EOD_post in the box named EOD_box.

The EOD_post command job is now in the EOD_box box job, and has inherited starting parameters of the box.

Setting Time Dependencies

The test_run job you specified at the beginning of the chapter has no starting conditions. Therefore, it will only run if it is started using the sendevent command. To set the job to run automatically on certain days at a certain time, such as 10:00 a.m. and 2:00 p.m. on Mondays, Wednesdays, and Fridays, you would modify the job using the JIL script as follows:

```
update_job: test_run
date_conditions: y
days_of_week: mo, we, fr
start_times: 10:00, 14:00
```

This JIL script instructs:

- To update the job named test_run.
- Activate the conditions based on date.
- Set the job to run on Mondays, Wednesdays, and Fridays.
- On each of these three days, start the job at 10:00 a.m. and 2:00 p.m.

The times shown in the script previously are quoted, since they contain a colon. They could also have been escaped by using backslashes, as follows:

```
start_times: 10\:00, 14\:00
```

Additional Time Setting Features

If you wanted the time settings for the job based on a specific time zone, you would use the timezone attribute. If you specify a time zone that includes a colon, you must quote the time zone name like:

```
timezone: IST-5:30
```

If you do not quote a time zone that contains a colon, the colon will be interpreted as a delimiter, producing unexpected results.

If you had wanted to run the job every day, rather than only on specific days, you could have specified the all value, instead of listing the individual day values. Or, if you had wanted to schedule the job for specific dates, rather than specific days of the week, you could have specified a custom calendar. First, you would have had to define the calendar, using the Graphical Calendar Facility, or the autocal_asc command. Then, you would specify the calendar name, weekday_cal, using the following JIL statement:

```
run_calendar: weekday_cal
```

You could have specified a custom calendar specifying the days on which the job was not to be run, holiday_cal, using the following JIL statement:

```
exclude_calendar: holiday_cal
```

If you wanted the job to run at specific times every hour, as opposed to specific times of day, the minutes past every hour could have been specified. For example, to run a job at a quarter after and a quarter before each hour, use the following JIL statement:

```
start_mins: 15, 45
```

Note: If a job runs daily at the same time (example: 12:00) and you edit this job definition and save this job at (11:59), the job will not run today but will run tomorrow at (12:00).

When a start time job definition is written to the database within one minute of the current runtime, the start time will be placed in the future, meaning tomorrow. However, if the start time is two minutes or greater from the current save time the job will run today.

Deleting a Job

Now you will delete the test_run job, which you specified at the beginning of this chapter. To delete the test_run job, enter the following JIL sub-command:

```
delete_job: test_run
```

The delete_job sub-command checks the job_cond table and notifies you if dependent conditions for the deleted job exist. This functionality only works when JIL is in job verification mode (which is the default mode).

Deleting a Box Job

To delete a box, and recursively delete every job in that box:

Use the delete_box subcommand as follows:

```
delete_box: EOD_box
```

To delete a box, but leave its contents intact:

Use the delete_job subcommand on the box as follows:

```
delete_job: EOD_box
```

You can configure a number of other attributes using JIL. These attributes are described in detail in the chapter JIL/GUI Job Definitions in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*. This reference chapter provides complete information on all job attributes specified using JIL.

Specifying One-Time Job Overrides

Using JIL, you can specify a job override for the next run of a particular job. In other words, the next time a job runs, you can change its behavior. Job overrides are applied only once. If a RESTART event is generated because of system problems, Unicenter AutoSys JM will reissue a job override until the job actually runs once, or until the maximum number of retries limit is met. After this, the override is discarded.

Notes:

The maximum number of job restarts after system or network failure is specified in the MaxRestartTrys parameter in the configuration file.

One-time job overrides will be applied to jobs restarted due to system problems, but will not be applied to jobs restarted because of application failures. System problems include such things as machine unavailability, media failures, or insufficient disk space. Application failures include such things as inability to read or write a file, command not found, exit status greater than the defined maximum exit status for success, or various syntax errors.

The following attributes can be modified in a job override:

Attributes

auto_hold	min_run_alarm	std_in_file
command	n_retrys	std_out_file
condition	profile	term_run_time
date_conditions	run_calendar	watch_file
days_of_week	run_window	watch_file_min_size
exclude_calendar	start_mins	watch_interval
machine	start_times	
max_run_alarm	std_err_file	

JIL will not accept an override if it results in an invalid job definition. For example, if a job definition has only one starting condition, start_times, JIL will not allow you to set the start_times attribute to NULL because removing the start condition makes the job definition invalid (no start time could be calculated).

Setting Job Overrides

To set job overrides, you use the override_job subcommand; you only need to specify those attributes that you want to override. Using this command, you can also temporarily delete a job attribute.

For example, if you wanted to run a job named RunData with no conditions (where some had been previously specified) and you wanted to output the results to a different output file, you would enter a JIL script as follows:

```
override_job: RunData
condition: NULL
std_out_file: /usr/out/run.special
```

To cancel the job overrides specified in the script previous, you would enter the following JIL script:

```
override_job: RunData delete
```

Note: Once you have submitted a JIL script to the database, you cannot view the JIL script and edit a job override. If you want to change the override values, you must submit another JIL script with new values, or use the GUI. However, the original override (that is, the first over_num) remains stored in the overjob table in the database.

Example JIL Script

The following is a full example of an JIL script. It incorporates the creation and use of a command job, a file watcher job, and a box job. The processing scenario is described following:

- A file named / download / mainframe / sales.raw is expected to arrive from the mainframe sometime after 2:00 a.m.
- When the file arrives, it is processed by the command file named filter_mainframe_info, and the results are placed in the file named / download / mainframe / sales.sql.
- When the previous functions are completed, the file named / download / mainframe / sales.sql (containing SQL statements) is executed.

```
# Example of Jobs
insert_job: Nightly_Download
job_type: b
date_conditions: yes
days_of_week: all
start_times: 02:00

insert_job: Watch_4_file
job_type: f
box_name: Nightly_Download
watch_file: /download/mainframe/sales.raw
machine: gateway

insert_job: filter_data
job_type: c
box_name: Nightly_Download
condition: success(Watch_4_file)
command: filter_mainframe_info
machine: gateway
std_in_file: /download/mainframe/sales.raw
std_out_file: /download/mainframe/sales.sql
std_err_file: /log/filter_mainframe_info.err

insert_job: update_DBMS
job_type: c
box_name: Nightly_Download
condition: success(filter_data)
machine: gateway
command: isql -U jeff -P jeff
std_in_file: /download/mainframe/sales.sql
```

An example of the output generated by the autorep command for the previous job definition is provided in Examples for the autorep command in the chapter Commands of the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

The Graphical Calendar Facility

This chapter describes the Graphical Calendar Facility and how to use it to create, view, and maintain calendars. It also describes how to preview a calendar before applying its dates and how to apply custom rules to a calendar.

A calendar is simply a collection of dates grouped as a single entity. The Calendar Facility lets you define and maintain calendars using a point and click approach on graphical displays of a conventional calendar. Once these dates are defined in a calendar, you can use the calendar to schedule jobs. In the job definition, you can specify that a job start (or not start) on the dates defined in a calendar. Calendars do not in themselves convey any rules about when a job should or should not start; this meaning is assigned exclusively through the job definition.

The Calendar Facility lets you:

- Define calendars.
- Apply custom rules to a calendar, such as the “first weekday of every month,” rather than selecting the individual dates by hand.
- Block out certain dates, such as holidays, when editing calendars.
- Select options that will automatically reschedule conflicting dates when applying a rule. A “conflicted” date is a date that is blocked out but also meets the qualifications of the rule being applied. A number of alternatives for rescheduling are provided.
- Build a new calendar by overlaying multiple, pre-existing calendars, and allowing you to further customize the new calendar manually.
- Preview a calendar before applying it to another calendar.
- Import and export text definitions for calendars.

Note: There is a command-line based calendar editor utility. It is called autocal_asc.

Scheduling Jobs with Calendars

In the job definition, use one of the following methods to apply a calendar:

- With JIL, enter a calendar name in the run_calendar or exclude_calendar attribute.
- In the Job Definition Date/Time Options dialog, enter a calendar name in the Run on Days in Calendar field or the Do NOT Run on Days in Calendar (Exclude) field.

For example, you could create a calendar called “holidays” containing the dates of all corporate holidays. For jobs that you want to start on holidays, you would define this using the attribute:

```
run_calendar: holidays
```

For jobs that you do not want to start on holidays, you would define this using the attribute:

```
exclude_calendar: holidays
```

Jobs scheduled with the run_calendar attribute are scheduled to start on every day specified in the calendar, at the times specified in the calendar or in the start_times or start_mins attribute. If present in the job definition, the start_times or start_mins attribute overrides the times specified in a run calendar. If no start time is specified, calendar-scheduled jobs start at midnight, by default.

Note: Times can be assigned to calendars only when using the command-line calendar definition tool, autocal_asc.

Jobs scheduled with the run_calendar attribute are scheduled on the next available date from that calendar. Dates previous to the current date are ignored.

Jobs scheduled with the exclude_calendar attribute can make use of other start conditions in the job definition. In this case, Unicenter AutoSys JM evaluates the start conditions and, if they are true, checks if the date is set in the exclude_calendar. If it is in the exclude_calendar, the job will not be started, and its status will be changed to INACTIVE.

For more details on these job attributes, see their reference pages in the chapter “JIL/GUI Job Definitions” in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

Starting the Calendar Facility

To start the Calendar Facility:

Do one of the following:

- Single-click on the Calendars button in the GUI Control Panel.
- Enter the following command at the UNIX prompt:

```
autocal &
```

The Calendar Definition window appears, as shown in Calendar Definition Window in this chapter.

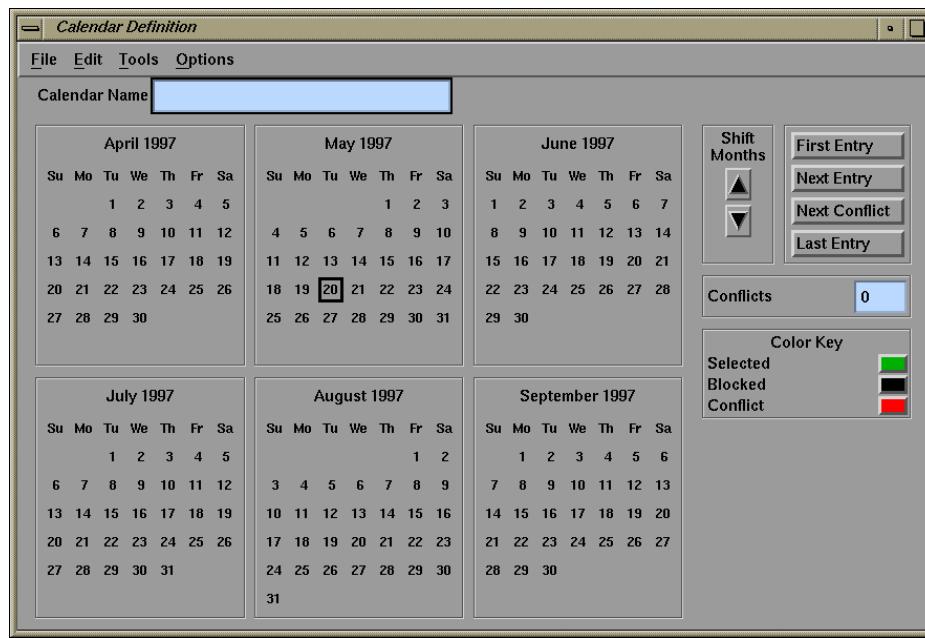
Calendar Facility Screens

The Calendar Facility consists of the following primary screens:

Screen Name	Function
Calendar Definition	This window is used to define a calendar and specify the dates the calendar should include. It also provides access to all other calendar functions and screens.
Calendar Selection	This dialog is used to select a predefined calendar to view, modify, or delete.
Term Calendar Rule	This dialog is used to specify a rule to apply to the calendar currently being created or modified.
Term Calendar Viewer	This window is used to preview a preexisting calendar before applying its dates to the calendar currently being created or modified. It can also be used to browse calendars that have already been defined.
Import/Export File Name	This dialog is used to specify a calendar text file (ASCII format) to import from or export to.

Calendar Definition Window

The first screen that displays when the Calendar Facility starts up is the Calendar Definition window:



The Calendar Definition Window is divided into the following regions:

Menu Bar

At the top of the window.

Calendar Display

At the center of the window.

Navigation and Legend Controls

At the right of the window.

Menu Bar

At the top of the Calendar Definition window is the menu bar, containing four pull-down menus: File, Edit, Tools, and Options.

File Menu

The File menu contains the following options:

File Menu Option	Action
New	Displays the New Calendar Name dialog for specifying a name for a new calendar to be created.
Open	Displays the Calendar Selection dialog for choosing an existing calendar to be edited.
Save	Saves the calendar currently being edited, using its current name.
Save As	Displays the Save As dialog, asking you for the new name under which the calendar currently being edited will be saved.
Delete	Displays a verification dialog, asking you to confirm that you want to delete the calendar currently being edited.
Rename	Displays the New Calendar Name dialog, asking you to specify a new name for the calendar currently being edited.
Import	Displays the Import File Name dialog so you can select the directory and filename of the text file containing calendar definitions that you want to import.

File Menu Option	Action
Export	Displays the Export File Name dialog so you can select the directory and name of the file to which you want to save all the calendars in the database, in text form.
Print	Prints the calendar currently being edited, using the print command specified in the application defaults or the print command specified using the Set Print Command option from the Options menu.
Exit	Displays a verification dialog, asking you to confirm that you want to exit the application. If you have made changes that have not been saved, you will be notified and given an opportunity to save your changes. If you indicate that you want to exit, the application is exited.

Edit Menu

The Edit menu contains the following options:

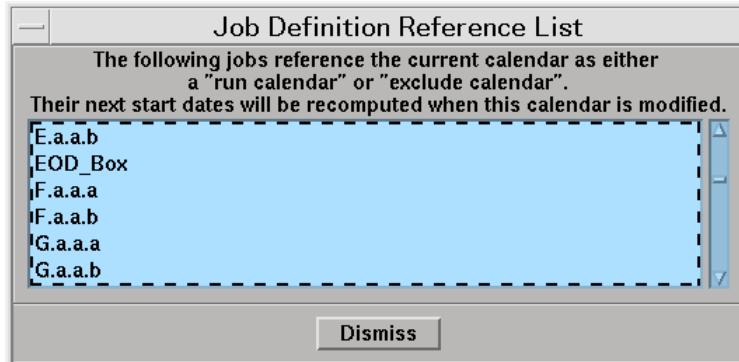
Edit Menu Options	Action
Apply Rule	Displays the Term Calendar Rule dialog, which allows you to set multiple dates at once using a variety of rule options.
Revert	Resets the state of all the dates in the current calendar to those last saved to the database.
Clear	Resets the state of all the dates in the current calendar to the Unset state, regardless of their current states.

Tools Menu

The Tools menu contains the following options:

Tool Menu Options	Action
Term Calendar Viewer	Displays the Term Calendar Viewer window, which allows you to preview various calendars prior to applying their dates to the calendar currently being edited.
Job Definition Reference List	Displays a list of all the jobs that reference the calendar currently being edited, either as their “run calendar” or “exclude calendar.” This list indicates which jobs will be affected by any changes you make to the current calendar.

The following is the Job Definition Reference List:



Note: Whenever a calendar is updated, Unicenter AutoSys JM re-computes the starting times for all jobs that use that calendar.

Options Menu

The Options menu contains the following options:

Option Menu Options	Actions
Date Range	<p>Displays the Date Range pull-down menu, through which you may specify the date range of the calendar currently being edited. The default date range is the prior year through the current year (Current Year). You may extend that to include additional years, and these are the options:</p> <ul style="list-style-type: none"><li data-bbox="801 726 1090 756">■ Through Next Year<li data-bbox="801 783 1122 813">■ Three Calendar Years<li data-bbox="801 840 1101 870">■ Four Calendar Years<li data-bbox="801 897 1096 927">■ Five Calendar Years<li data-bbox="801 954 1093 984">■ Ten Calendar Years
Set Print Command	<p>The Date Range option limits how far into the future you can set dates in the current calendar. You can increase the date range of a calendar, and you can decrease it through the previous year.</p> <p>When you open an existing calendar, either the date range of the calendar or the current date range, whichever is greater, becomes the new date range for the current Calendar Facility session.</p>
	<p>Displays a dialog box prompting you to specify the full path to the print command to be executed when printing the calendar currently being edited. For example, the print command might be “/bin/qprt -Plp3.” The print command could also be set using the application default settings, as described in Print Command in this chapter.</p>

Calendar Display

The Calendar Display region of the window displays six months of the calendar currently being edited. By default at startup, two quarters of the current year display, one of which contains the current day (indicated by a box). Using the navigation controls at the right side of the window, you can advance or move backward through the calendar, one quarter at a time.

The Calendar Name field at the top left of the Calendar Display region displays the name of the calendar currently being edited. You can change this name using the Rename option from the File menu. You cannot edit the Calendar Name field.

Date States

Each date in the calendar is a selectable button. By using the mouse and by clicking, you can set each date to one of the following states:

Unset

The date is not set.

Set

The date is set.

Blocked

The date is ineligible for setting when applying a rule.

You can cycle through these three states by clicking the mouse button additional times. The current state of each date is indicated by its color, as listed in the Color Key area of the Navigation Controls region.

Note: Calendars are stored in the database. Only the Set dates are stored. Dates designated as Blocked are only in effect while editing a calendar. The Blocked state is only useful while applying rules. Blocked dates are not saved in the database, nor are the rules.

In addition to the three user-selectable states above, there is a fourth, system-generated state called the “Conflict” state. This state occurs when both of the following are true:

- A rule is applied to a calendar.

- A date that qualifies for setting by the rule has been previously set to the Blocked state and rescheduling has not occurred. (Rescheduling may not have occurred if either a reschedule rule has not been applied, or an applied reschedule rule cannot find a non-conflicting date to move to.)

For example, you may have marked all holidays as Blocked, one of which was January 1. Now you want to apply a rule to the first day of each month, which would conflict with the January 1 Blocked date. If there is no reschedule rule in effect, such as “move to the next weekday,” or if the reschedule rule specifies to move backwards, which would end up on a date in the previous year, a Conflict state would result. In this case, you must manually correct the situation before attempting to save the calendar to the database.

For more information about Rescheduling Rules, see Rescheduling Rule in this chapter.

Navigation Controls

The Navigation Controls region of the window contains controls that do the following:

- Let you specify which six months (or two quarters) are to be displayed in the window.
- Display additional information relevant to the calendar currently being edited.

Shift Months Area

The Shift Months area has two push buttons (with up and down arrows) to advance or move backward through the calendar, one quarter at a time. You can change the calendar display to any two quarters within the calendar's date range. (For information about the date range, see Menu Bar in this chapter.) When shifting backward, you can move from the current year into the prior year. Only one prior year is viewable through the Graphical Calendar Facility (that is, if the current year is 2002, you could shift back to 2001, but not to 2000).

Note: Rules cannot be applied to dates prior to the current date.

Skip Button Area

To the right of the Shift Months area is a row of push buttons to “skip” to a particular date whose state has been set. The following buttons are available:

First Entry

Changes focus to the first date in the calendar that has been set.

Next Entry

Changes focus to the next date in the calendar that has been set relative to the date that currently has the focus.

Next Conflict

Changes focus to the next date in the calendar that has been set to the Conflict state by the system.

Last Entry

Changes focus to the last date in the calendar that has been set.

Note: The date with the focus is designated with a darkened box around the date.

When you use the skip buttons, if the focus is changed to a date that is not currently displayed, the calendar display will shift to bring that date into view.

Number of Conflicts Area

Below the Shift Months area is the Number of Conflicts area displaying how many dates with the Conflict state currently exist in the calendar. This field is updated each time you apply a rule, or manually change a Conflict date to another state.

Note: You cannot save a calendar containing unresolved conflicts.

Color Key Area

Immediately below the Number of Conflicts area is the Color Key area, which displays the colors that indicate the states of each date. These colors are user-configurable, as described in Object Color in this chapter. The following colors are used by default:

Unset dates

Background color of the Graphical Calendar Facility

Set dates

Green

Conflict dates

Red

Blocked dates

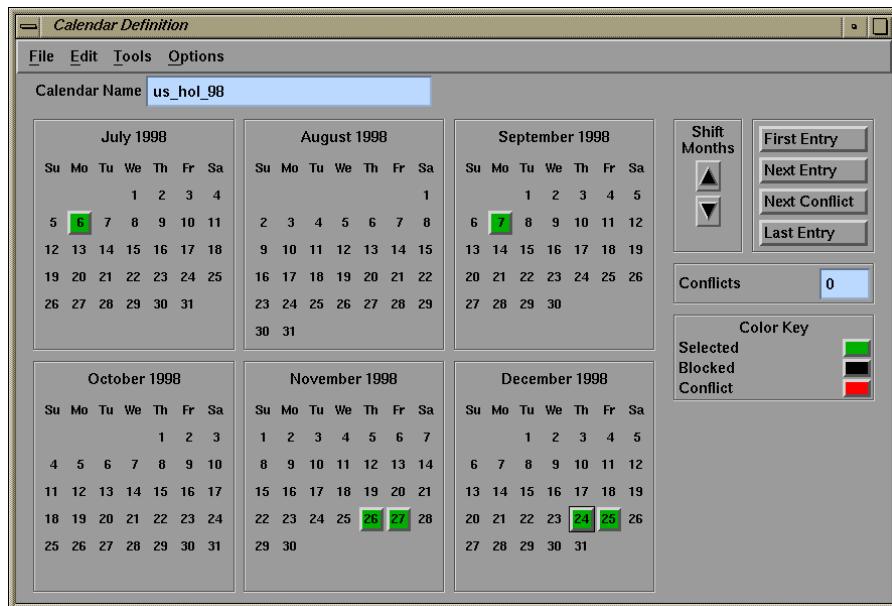
Black

Creating a Simple Calendar

To create a simple calendar containing all 1998 holidays:

1. Choose File, New.
2. In the New Calendar Name dialog, enter:
`us_hol_98`
3. Press Enter.
4. In the Calendar Definition window, set the holiday dates by clicking the left mouse button on each date. If the desired dates are not displayed, use the Shift Months arrows to bring them into view.

5. Choose File, Save. Your calendar will resemble the following:



Dates Prior to Today's Date

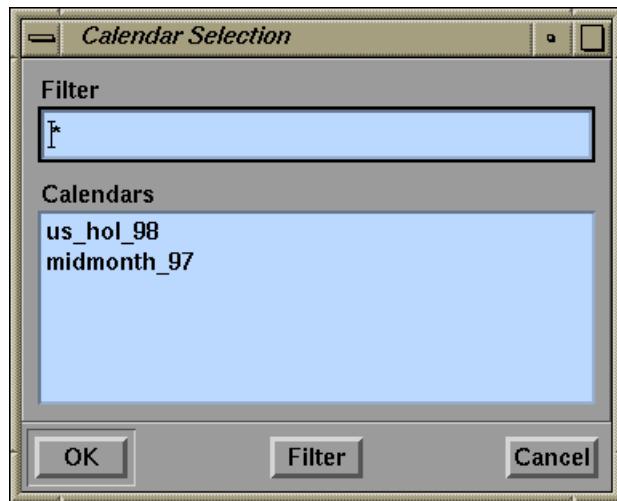
Calendars exist to simplify the scheduling of job runs; therefore, only current and future dates are applicable. While the Graphical Calendar facility allows you to set dates in the past, they will be ignored. Moreover, when you merge calendars to create a new calendar, any dates prior to today will be dropped from the new calendar.

Calendar Selection Dialog

The Calendar Selection dialog lets you specify which calendar you want to open for editing, or which one to use for a rule specification (see Term Calendar Rule Dialog in this chapter).

To access the Calendar Selection dialog:

Choose File, Open. The Calendar Selection dialog appears:



When this dialog appears, it contains a scroll list of all the calendars currently in the database. In the Filter field of this dialog, you may specify any string, including the asterisk (*) wildcard character, to show only those calendars whose names include the string. The default filter, (*)" matches all calendar names. For example, to display only those calendar names starting with the string "us_hol," such as "us_hol_97" and "us_hol_98," you would specify the filter "us_hol*," then click the Filter button. The list will display only the matching calendar names.

You can select any calendar in the list by clicking the mouse. Click OK to open the calendar and dismiss the dialog. Click Cancel to dismiss the dialog without selecting a calendar.

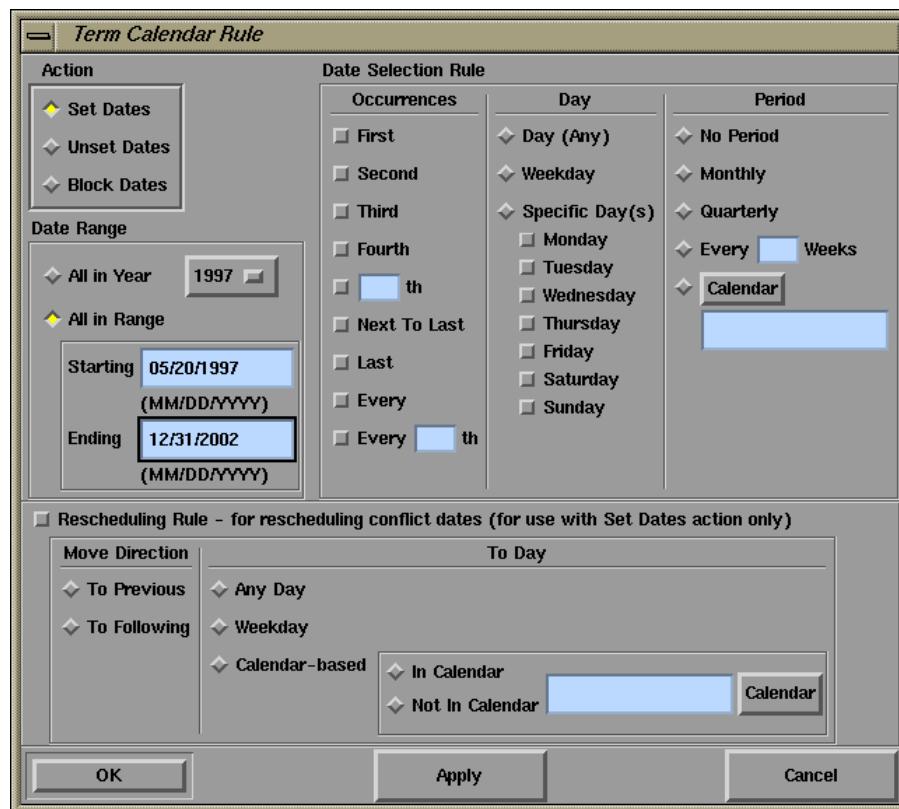
Term Calendar Rule Dialog

The Term Calendar Rule dialog lets you set a state for multiple dates simultaneously, rather than selecting each date individually.

To access the Term Calendar Rule dialog:

Choose Edit, Apply Rule.

The Term Calendar Rule dialog appears.



The Term Calendar Rule dialog is divided into the following three regions:

Rule Specification

The top portion of the dialog.

Rescheduling Rule

Center of the dialog.

Control

The bottom portion of the dialog.

Rule Specification

The Rule Specification region provides a variety of options for specifying which dates in the currently selected calendar you want to affect, and what state to set for those dates. This region consists of the following three areas:

- Action Area

- Date Range
- Date Selection Rule

Action Area

The Action Area lets you apply one of the following states on the selected dates:

Set Dates

Change the state to Set.

Unset Dates

Change the state to Unset.

Block Dates

Change the state to Blocked. Blocked dates will not be set during any subsequent rule applications.

Note: These actions are mutually exclusive; only one of these actions can be selected.

Date Range Area

In the Date Range area, you specify the date range over which the rule should apply. By default, the date range of the currently selected calendar is in effect. However, you can change the date range by selecting an option from either the All in Year pull-down menu, or by specifying a date range in the All in Range edit fields. When entering a date range, the dates must fall within the range set in the Options Date Range pull-down menu.

Note: Rules are not applied to any date prior to today's date.

Date Selection Rule Area

The Date Selection Rule area contains the following three options:

Date Selection Options	Action
Occurrences	Lets you specify the "occurrence of a day" for which the rule should be applied. Select one or more options by pressing the corresponding toggle button.

Date Selection Options	Action
Day	Lets you specify the days of the week to which the rule should be applied. Select one of the following: Day (Any), Weekday, or Specific Days. If Specific Days is selected, one or more of the specific days of the week must be chosen as well.
Period	Lets you specify the period during which the rule should be applied. Only one of the following options can be chosen: No Period, Monthly, Quarterly, Every "n" weeks, or the days in a specified Calendar. The No Period option is used for non-repeating periods, such as Every/Monday—this option is generally used with the Every or Every "th" option in the Occurrences sub-area.

If the Calendar option is selected, only the dates specified in the indicated calendar will be used when applying the rule. You can either enter the calendar name directly, or press the Calendar button to display the Calendar Selection dialog, from which you can choose a calendar for the period.

Note: The rules used to set a calendar are not saved after the calendar has been created and saved.

Examples of Date Selection Rules

The following examples illustrate the use of the Date Selection Rule options. We recommend you experiment with these rules. You can always revert to the last saved version of a calendar by using the Revert option from the Edit menu, or you can clear everything using the Clear option from the Edit menu.

Setting Dates

To set the 3rd Tuesday of every month throughout the entire currently selected calendar:

1. In the Action area, select Set Dates.
2. Keep the default date range, which is the currently selected calendar's entire range.
3. In the Occurrences sub-area, select either the Third option or enter 3 in the "th" option.
4. In the Day sub-area, select Tuesday (which automatically selects the Specific Days option).
5. In the Period sub-area, select Monthly.
6. Click OK or Apply to apply the rule to the calendar you are currently editing.

Blocking Dates

To block every holiday date and prevent those days from being scheduled:

Follow these steps (assuming the holiday dates already exist in a calendar named "us_hol_98"):

1. In the Action area, select Block Dates.
2. Keep the default date range, which is the currently selected calendar's entire range.
3. In the Occurrences sub-area, select Every.
4. In the Day sub-area, select Day (Any).

5. In the Period sub-area, press the Calendar button. When the Calendar Selection dialog appears, select the calendar named “us_hol_98,” and click OK. The calendar name will appear in the Calendar field in the Period sub-area.
6. Click Apply to apply the rule to the calendar you are currently editing.

Following on with this example, assume that you want to change the state of the first day of every month to Set.

To change the state of the first day of every month to Set:

1. In the Action area, select Set Dates.
2. Keep the default date range, which is the currently selected calendar's entire range.
3. In the Occurrences sub-area, select First.
4. In the Day sub-area, select Day (Any).
5. In the Period sub-area, select Monthly.
6. Click Apply to apply the rule to whatever calendar you are currently editing.

When you apply this rule, a Conflict state will be assigned to January 1, since this date is defined in “us_hol_98,” and was marked as Blocked in the previous example. To address this you must either “unset” it manually and, if desired, set another date in its place, or you could specify a Rescheduling Rule to accommodate this type of conflict. Rescheduling Rules are described in the next section.

Rescheduling Rule

The Rescheduling Rule region lets you control how date conflicts should be resolved when applying a rule. To specify a rescheduling rule, you must indicate a “move direction” and the day to which the newly scheduled Set state should be moved when a conflict occurs. You do this in the Move Direction and To Day areas.

Note: Conflicts can also occur if a date is Set, then Blocked. Whenever a conflict occurs, the Set state is moved when rescheduling.

Move Direction

You can select one of the following Move Direction options:

To Previous

Moves the Set state backward in the calendar.

To Following

Moves the Set state forward in the calendar.

To Day

In addition to specifying the Move Direction, you must specify one of the following To Day options:

Any Day

Next available date.

Weekday

Next available weekday date.

Calendar-based

Next available date in the specified calendar. You must choose either In Calendar or Not In Calendar, and specify a calendar name, either by entering it directly, or by pressing the Calendar button, then selecting a calendar from the Calendar Selection dialog that appears.

Example

To remedy the conflict situation in the last example from the previous section, assume that you want to apply a Rescheduling Rule that specifies “any day following the date in conflict that is not also a holiday.”

To apply a Rescheduling Rule that specifies “any day following the date in conflict that is not also a holiday”:

1. In the Action area, select Set Dates.
2. Keep the default date range, which is the currently selected calendar's entire range.
3. In the Occurrences sub-area, select First.
4. In the Day sub-area, select Day (Any).
5. In the Period sub-area, select Calendar. When the Calendar Selection dialog appears, select the calendar named “us_hol_98,” and click OK. The calendar name will appear in the Calendar field.
6. In the Move Direction area, select To Following.
7. In the To Day area, select the Not in Calendar option and enter the calendar named “us_hol_98.”
8. Click OK or Apply to apply the rule to the calendar you are currently editing.

Notes:

- Multiple conflict dates can be rescheduled to the same new date. For example, if you block all weekend dates, then apply a rule to set every day, with rescheduling to the previous weekday, both the Saturday and Sunday conflicts will be resolved to the preceding Friday. If you want separate runs for Saturday and Sunday, turn off the Rescheduling Rule and resolve the conflicts manually.
- If you intend to use a Rescheduling Rule, you should set it up at the same time you set up the Date Selection Rule, rather than wait for conflicts to occur.

Control

At the bottom of the Term Calendar Rule dialog, there are three buttons:

Ok

Applies the current rule and dismisses the dialog.

Apply

Applies the rule but does not dismiss the dialog

Cancel

Dismisses the dialog without applying the rule.

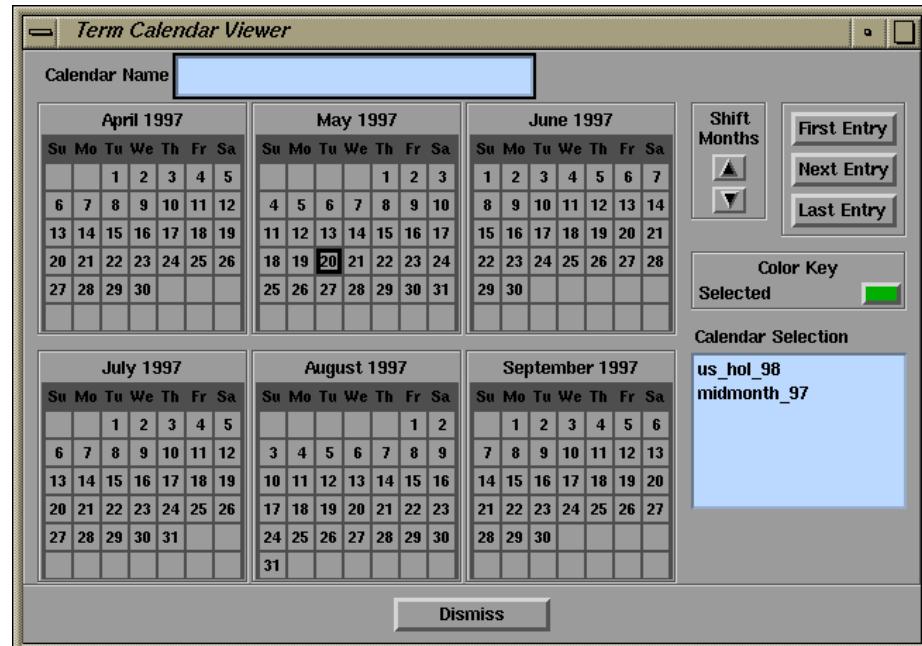
Term Calendar Viewer

The Term Calendar Viewer allows you to preview a calendar before using it in a rule you are about to apply. You can easily browse any existing calendars to see which dates they contain.

To access the Term Calendar Viewer:

Choose Tools, Term Calendar Viewer.

The Term Calendar Viewer appears:



The Term Calendar Viewer is divided into the following regions:

Calendar Display

At the left and center of the window. The dates in this calendar are read only—they cannot be edited. You can only edit dates using the Calendar Definition window.

Navigation Controls

At the right of the window.

Note: Unlike the Calendar Definition window, there is no Next Conflict button in the Navigation Controls region of this window. This is because calendars cannot be stored in the database if they contain conflicts. Also note that Blocked days are not stored in the database—all dates saved in a calendar are “Set” dates. However, calendars may be used to block out dates by way of the Term Calendar Rule dialog.

When the Term Calendar Viewer first appears, it displays six months of the current year. You must then select a calendar by clicking its name in the Calendar Selection list in the lower-right corner of the window. Navigation is exactly the same as in the Calendar Definition window.

As each calendar is selected, it completely replaces the previously viewed calendar, and its name is displayed in the Calendar Name field at the top of the window.

When you are finished viewing calendars, click the Dismiss button to close the window.

Combining Calendars

Calendars can be combined in a number of ways. For example, you can create a calendar that includes all the dates that are in either one calendar or another. Or you can create a calendar that includes all the dates that are in one calendar but not in another. In fact, you can combine any number of calendars in these ways.

For example, suppose you have a calendar named “us_hol_98” containing all United States holidays, and you have a calendar named “corp_hol_98” containing all your corporation holidays. In this scenario, you want create a new calendar for all combined holidays called “all_hol_98.” This new calendar will include all the dates that are in either of the previously defined calendars. The best way to accomplish this is described in the following steps.

To combine calendars:

1. Choose File, New.
2. In the New Calendar Name dialog, enter:
`all_hol_98`
3. Click OK.
4. Choose Edit, Apply Rule.
5. In the Term Calendar Rule dialog, select the Set Dates Action, the Every Occurrence, and the Day (Any) Day settings.
6. In the Period sub-area, click the Calendar button, then select “us_hol_98” from the Calendar Selection dialog, and click OK.
7. In the Term Calendar Rule dialog, click Apply.
8. Repeat steps 5 and 6, selecting the calendar “corp_hol_98.”
9. Choose File, Save.

This process of combining calendars can be repeated for any number of calendars.

Note: When combining calendars, any dates prior to the current date will be dropped from the new calendar.

Printing Calendars

You can print calendars to obtain a hard copy for your files. Before you attempt to print a calendar, be sure the print command is correctly specified, either through your X resources file, or by way of the Set Print Command option from the Options menu. The print command must include the full path to the print facility to be used, plus all appropriate arguments. Once this is set, you can open an existing calendar, or create a new calendar, then select the Print option from the File menu.

Import/Export File Name Dialog

You can import calendar text files and you can export calendars to text files. This is particularly useful for copying calendars from one instance to another. You perform calendar imports and exports using the Import (and Export) File Name dialog.

Importing Calendar Text Files

Calendars contained in ASCII text files can be imported into the database. These text files may contain multiple calendars, each of which must be delimited with the calendar: *calendar_name* attribute. A sample calendar text file is shown following:

```
calendar: Q1paydays
01/01/1998
01/15/1998
02/01/1998
02/15/1998
03/01/1998
03/15/1998
calendar: Q1holidays
01/01/1998
```

Comments may be included following a space after the calendar name or after each date.

If you export a calendar, the output will be in this same format.

To access the Import File Name dialog:

Choose File, Import.

The Import File Name dialog appears:



At this dialog, you can either enter the full path to the calendar in the Selection field, or use the Filter field, Directories list, and Files list to select the file.

When using a filter, the Filter field must contain a directory path name, which is everything up to the last slash (/) followed by the file pattern containing an asterisk (*). For example, to search the directory /home/my_dir for all file names with the string cal, you would specify: /home/my_dir/*cal*. Then, click Filter, and the Files list will display all the files containing that string. Then, click on the file you wish to import, and the full path name will appear in the Selection field. Finally, click OK to import the file.

Note: If the text file being imported contains calendars with the same names as calendars already existing in the database, these calendars will not be imported. A warning dialog will notify you that a calendar name is duplicated, and that the import of this calendar will not occur. Also, after the import has completed, an information dialog will tell you how many calendars were imported.

Exporting Calendars

You can export all the calendars from the database to an ASCII text file. To export, select File, Export. The Export File Name dialog will display, allowing you to specify a file name for the calendar text file.

Note: When you use the export facility, all the calendars in the database are exported to a single ASCII text file. After the export has completed, an information dialog will tell you how many calendars were exported. You cannot export just a single calendar; however, you can edit the ASCII file after you export all the calendars.

The Export File Name dialog uses the same filtering as described in Importing Calendar Text Files in this chapter.

Customizing the Calendar Facility

There is a set of X resources that you can use to customize the appearance and behavior of the Calendar Facility. In particular, these resources allow you to control:

- The fonts that are used in the Calendar Definition window as well as the Calendar Facility dialogs. The choice of fonts will affect the size of the various windows and dialogs.
- The colors used to indicate whether dates are Set, Unset, or Blocked.
- The print command to be executed when you want to print a calendar.
- The Calendar GUI icon text and the Calendar title bar text.

Descriptions of each of the resources, which can be customized, are given below. All of these can be set by modifying the X resource file Autocal. The X resources files reside in the local app-defaults directory, which varies across platforms. It is usually in /usr/lib/X11/app-defaults or /usr/openwin/lib/app-defaults. If you are not sure which directory these files are in, ask your system administrator.

Individual users may have their own copy of the X resources files in their \$HOME directory, which will take precedence over the app-defaults files.

For most operating systems, if you are exporting the display to another machine you must edit the appropriate files in the app-defaults directory on the local machine.

For Solaris, you must edit the files in both the /usr/lib/X11/app-defaults and /usr/openwin/lib/app-defaults directories. The files in /usr/lib/X11/app-defaults control the resources when you export the display.

We have listed the various resource names here as a reference only. For the default values, see the Autocal file. The provided resource file values work well for a majority of platforms.

Font Selection Resources

The following resource specifications control font selection. They are completely independent of any other font settings.

```
! main font for all fields not otherwise specified
Autocal*fontList:
! font for dates within each month in Calendar
! Definition window
Autocal*month_form*fontList:
! main font for everything in Calendar Viewer
! except dates
Autocal*calViewForm*fontList
! font for dates in Calendar Viewer
Autocal*calViewform*month_form*fontList
! font for term rule dialog
Autocal*termRuleForm*fontList:
```

Object Color

The following resources affect the colors of the various objects within the Calendar Facility.

Note: Several of the following colors come in pairs, such as setColor and setLabelColor. The LabelColor is the color of the numbers that appear on the date buttons and should be chosen so that they are easily readable with the color of the button on which they display. For example, do not specify setColor as blue and setLabelColor also as blue—the numbers will not be visible on the dates. Select another color, such as black or white for the label.

```
! general application background color
Autocal*background:
! color to represent "set" dates
Autocal*setColor:
! color for label (foreground numbers) to represent
! "set" dates
Autocal*setLabelColor:
! color to represent "blocked" dates
Autocal*blockedColor:
! color for label (foreground) to represent
! "blocked" dates
Autocal*blockedLabelColor:
! color to represent "conflict" dates
Autocal*conflictColor:
! color for label (foreground) to represent
! "conflict" dates
Autocal*conflictLabelColor:
```

The following resource sets the color for a grid that appears between the dates in the Calendar Viewer window, to help differentiate it from the Calendar Definition window. If you do not want the grid to appear, select the same color as the background.

```
Autocal*viewerColor:
```

Date Range

The following resource sets the number of years in the date range of the calendar, as a default at start up. This can be overridden manually by way of the Date Range option from the Options menu, or by loading a calendar that has a larger date range than the default. These are the legal settings:

```
! 1 = prior year plus current year,  
! 2 = thru next year, 3 = thru third year,  
! 4 = thru fourth year, 5 = thru fifth year,  
! 10 = thru tenth year  
Autocal.dateRange:
```

Window Size

The following resources help keep the size of the window small, and it is recommend that you do not change their settings.

```
Autocal*calViewForm*month_form*marginHeight: 0  
Autocal*calViewForm*month_form*marginWidth: 0  
Autocal*calViewForm*month_form.marginWidth: 2
```

Print Command

The following resource specifies the print command, and its arguments, which will be executed when you request that a calendar be printed. Be sure to specify the full path name of the executable, or it may not be located and the print will fail.

```
Autocal.printCommand:
```

Calendar Title Bar Text and Icon Text

The following resources specify the title bar text and the icon text. By default the title bar text is "Calendar Definition" and the icon text is "CalDef."

```
Autocal.shellTitle:  
Autocal.iconTitle:
```

Note: When changing icon text, be sure the length of the new text string does not exceed the recommended maximum length for icon title text for your windowing system. Some window managers can display long icon text strings, while others will truncate them. Ensure the text string you specify for your icons displays appropriately. Also, some window managers allow you to change the size of icons and icon text font.

Load Balancing and Queuing Jobs

This chapter describes the use of real and virtual machines in the Unicenter AutoSys JM environment to provide load balancing and queuing functionality. It provides information about load balancing jobs across multiple machines, as well as queuing jobs to real and virtual machines.

Real Machines

In the environment, a real machine is any physical CPU that has:

- Been identified in the appropriate network database (for example, /etc/hosts) so that AutoSys can access it.
- Undergone a client software installation (and is licensed) so that Unicenter AutoSys JM can run jobs on it.

The above two conditions are required for a real machine to run jobs. However, to perform intelligent load balancing and queuing while executing jobs, it needs to know the relative processing power of the various real machines. Unicenter AutoSys JM provides both load balancing and queuing by way of the logical construct called virtual machines.

Virtual Machines

A virtual machine is comprised of one or more real machines, in whole or in part (or a combination of both). All real machines within a virtual machine must be of the same type, either Windows or UNIX. Virtual machines cannot be a mix of both UNIX and Windows machines.

By defining virtual machines, and then submitting jobs to run on those machines, you can specify:

- Runtime resource policies (or constraints) at a high level.
- That Unicenter AutoSys JM automatically execute those policies in a multiple machine environment.

Defining Machines

Define both real and virtual machines by using machine attribute statements within a JIL script. The following JIL subcommand defines a real or virtual machine:

```
insert_machine: machine_name
```

The following JIL machine attributes are used when defining machines:

Machine Attribute	Description
type	Specifies a machine type, which can be one of the following: <ul style="list-style-type: none">■ r for UNIX real■ v for UNIX virtual■ n for NT (real or virtual)
machine	Specifies a real machine name to be inserted in a virtual machine.
max_load	For real machines only, and used for load balancing.
factor	For real machines only, and used for load balancing.

Real machines only need to be defined if they meet one of the following criteria:

- Require a max_load or factor attribute to be set for them. (These attributes are discussed in the next section.)
- Are to be included in a virtual machine.

Virtual machines must be defined before you can use them.

Load balancing and queuing can be done only if real and virtual machines have been defined using these machine attribute statements. The following two attributes, used when defining real machines, are key for load balancing and queuing: max_load and factor.

Note: Real and virtual machines can only be defined using JIL. There is no GUI interface for defining machines.

For more information about the JIL subcommands and attributes pertaining to machines, see the chapter “JIL Machine Definitions” in the *Unicenter AutoSys Job Management Reference Guide*.

Specifying Machine Load (max_load)

The max_load attribute can be defined only for real machines. It describes how much of a load can be placed on a real machine, and it is specified with an arbitrary unit called a “load unit.” Any weighting scheme desired by the user can be used. For example, a load unit with a range of 10-100 would specify that machines with limited processing power are expected to carry a load of only 10, while machines with ample processing power can carry a load of 100. There is no direct relationship between the load unit value and any of the machine’s physical resources. Therefore, you should develop conventions that are meaningful to you. Zero and negative numbers cannot be used.

Job Attributes and Load Balancing and Queuing

For load balancing to work, every defined job that will impact the load on a machine must be assigned a job_load job attribute, which defines the relative load the job will place on a machine. Thus, a machine’s current load can be tracked, and overloading of a machine can be prevented. For example, if the max_load on a machine is “100” and the job_load for one job is “10,” then that job will use 10 percent of the machine’s resources.

In addition, for job queuing to take place, the priority job attribute must also be assigned in the job definition. The priority attribute specifies the relative priority of all jobs queued for a given machine. Without this attribute set, a job will run immediately on a machine, and it will not be placed in the queue.

Specifying Relative Processing Power (factor)

The factor machine attribute can be defined only for real machines. It is another arbitrary value that describes the relative processing power of a machine. This attribute's value is a real number that can contain a decimal. It is used to weigh available cycles on one machine against that of another machine. When AutoSys checks the available cycles on each machine, it multiplies the percent of free CPU cycles by the factor in order to determine which machine has more relative processing power available. Therefore, the factor value is typically a number between 0.0 and 1.0.

Using max_load and factor

The max_load attribute is primarily used to limit the loading of a machine. As long as a job's load will not cause a machine's max_load to be exceeded, the max_load attribute does not influence the decision of on which machine a job should be run. Conversely, the factor attribute is primarily intended to be used when deciding between machines for running a job, if more than one machine is available.

If these attributes are not specified in a real machine definition, they default to the values shown below:

```
max_load: none /* no limit */  
factor: 1.0
```

Note: A virtual machine is comprised of real machines. Therefore you do not specify max_load and factor attributes explicitly in a virtual machine definition. They are specified in the definitions of the real machines that make up the virtual machine.

Machine Definitions

Unicenter AutoSys JM can infer whether a machine being defined is a real or a virtual machine based solely on the attributes in the definition. Any machine definition containing a max_load or factor attribute must be a real machine definition, because only real machines can have these attributes. Any machine definition containing a list of machine attributes is a virtual machine definition.

Because of this, you can omit the type attribute when defining a UNIX machine. For Windows NT, however, the type attribute is required. Compare the following definitions:

Real UNIX	Real Windows
insert_machine: toad	insert_machine: tiger
max_load: 100	type: n
factor: .8	max_load: 100
	factor: .8

Virtual UNIX	Virtual Windows
insert_machine: pond	insert_machine: jungle
machine: toad	type: n
machine: frog	machine: tiger
	machine: monkey

To help you understand virtual machines and their capabilities, the following sections provide a series of examples that demonstrate the different combinations of real machines that can constitute a virtual machine. These examples include the JIL statements used to define these machines.

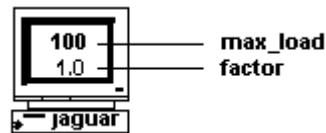
Defining a Real Machine

To define a real machine:

1. Assign a machine name, this must be its host name.
2. Assign a machine type: r for real UNIX or n for NT.
3. Optionally, assign a max_load and a factor attribute value.

The following demonstrates the definition of a real UNIX machine named “jaguar” with a max_load of 100 and a factor of 1.0.

```
insert_machine: jaguar
type: r
max_load: 100
factor: 1.0
```



To define a real Windows NT machine named “tiger,” enter the following JIL statements:

```
insert_machine: tiger
type: n
max_load: 100
factor: 1.0
```

Deleting Real Machines

The following JIL statement deletes the real machine definition for the machine named “jaguar”:

```
delete_machine: jaguar
```

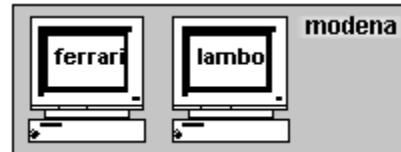
Defining a Virtual Machine

To define a virtual machine:

1. Assign a machine name.
2. Assign a machine type: v for virtual.
3. Specify the real machines that will make up the virtual machine.

The following demonstrates the definition of a virtual machine named “modena,” which is composed of two real machines named “ferrari” and “lambo.” Because the real machines do not specify a max_load and factor, they will have the default values for these attributes: a factor of 1.0 and unlimited load units.

```
insert_machine: modena
type: v
machine: ferrari
machine: lambo
```



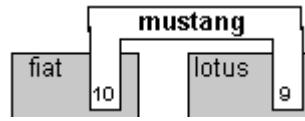
The following JIL statements define two real machines named “fiat” and “lotus,” and a virtual machine named “capri,” which is composed of the two real machines. The virtual machine is a superset of the two previously defined real machines. (Because the real machines are defined first, the virtual machine will use the max_load and factor attributes specified for them.)

```
insert_machine: fiat
type: r
max_load: 100
factor: 1

insert_machine: lotus
type: r
max_load: 80
factor: .9
insert_machine: capri
type: v
machine: fiat
machine: lotus
```

The following JIL statements define a virtual machine named “mustang” which is composed of slices, or subsets, of the real machines named “fiat” and “lotus.” Even though the real machines have been previously defined, only the reduced load portion (or slices) will be used in the virtual machine “mustang.”

```
insert_machine: mustang
type: v
machine: fiat
max_load: 10
machine: lotus
max_load: 9
```



Deleting Virtual Machines

To delete a real machine component of a virtual machine, you specify the virtual machine and the desired component. The following JIL statements delete only the real machine named “lambo” found within the virtual machine named “modena”:

```
delete_machine: modena
machine: lambo
```

If the machine “lambo” had been individually defined outside of the virtual machine, its individual definition still remains in effect.

To delete the entire virtual machine, you don’t have to specify any of the component real machines. The real machines are still defined—only the virtual machine they were in is deleted. The following JIL statement deletes the virtual machine named “mustang”:

```
delete_machine: mustang
```

Because the real machines “fiat” and “lotus” had been individually defined outside of the virtual machine, their individual definitions remain in effect.

Load Balancing

By specifying a virtual machine or a list of real machines in a job's machine attribute, rather than a single real machine, you can implement simple load balancing. That is, you can cause the workload to be spread across multiple machines, based on each machine's capabilities. In addition to load balancing, this feature is a useful way to ensure reliable job processing. For example, if one of the machines is down, load balancing will run the job on another machine.

When a job is ready to start, Unicenter AutoSys JM will determine which of the specified machines is best suited to run the job. The following JIL example shows the job definition statements for such a job:

```
insert_job: test_load
machine: modena
command: echo "Test Load Balancing"
job_load: 50
priority: 1
```

where:

modena

Is a virtual machine.

Alternatively, you can specify a list of real machines in the job's machine attribute, as shown below:

```
machine: ferrari, lambo
```

If the max_load attribute was not defined for either real machine (as in our example), or both machines had ample load units available, Unicenter AutoSys JM would choose the machine to run on based solely on available processing power. To accomplish this, Unicenter AutoSys JM does the following:

1. Determines the percentage of CPU cycles available on each real machine in the specified virtual machine. This is accomplished by one of the following actions:
 - Issuing a remote procedure call (RPC) to the real machine and requesting rstatd data.
 - Running vmstat on the real machine.
2. These methods are specified in the AutoSys configuration file using the MachineMethod parameter.

Note: rstatd is not currently supported on NCR or Pyramid client machines.
3. Multiplies it by the machine's factor value.

4. Chooses the machine with the largest result (that is, the machine with the most relative processing cycles available).

In the example machine list previously shown, the factor attribute is not specified for either machine, and thus the default factor value for each machine is 1.0.

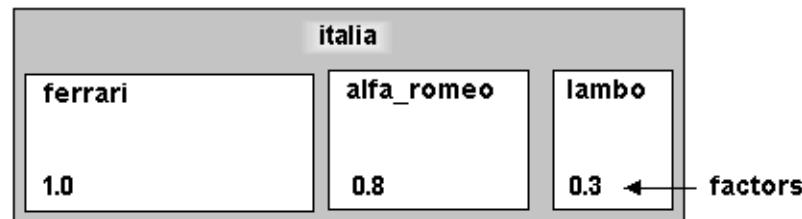
If the machines have equal max_load and factor values, it is equivalent to defining a job and specifying the following in the machine field:

```
machine: ferrari, lambo
```

The advantages of building a virtual machine are that it can be changed, and the new construct is immediately applied globally. Also, the values can vary between machines. Even when a set of real machines that have not been explicitly defined are specified in a job's machine attribute, the available CPU cycles are used to determine which machine will run the job.

In all likelihood, your system configuration will include machines of varying processing power, so you will need to specify the factor attribute value for each real machine.

The following illustrates three machines having different capabilities, which are grouped into a virtual machine.



The following JIL statements can be used to define this machine:

```
insert_machine: italia
machine: ferrari
factor: 1
machine: alfa_romeo
factor: .8
machine: lambo
factor: .3
```

To start a job on this virtual machine, simply specify “italia” as the machine attribute for the job. The event processor will perform the necessary calculations to determine on which machine to run the job, and reflect these calculations in its output log. The output is similar to this:

```
EVENT: STARTJOB JOB: test_mach
Checking Machine usages using RSTATD      :<ferrari=78*[1.00]=78>
<alfa_romeo=80*[.80]=64>
    <lambo=2*[.30]=6>
[ferrari connected]
EVENT: CHANGE_STATUS STATUS: STARTING JOB:
test_mach
```

Note that even though the “ferrari” usage was less than “alfa_romeo,” “ferrari” was picked because of the factors ($78 * 1.0 > 80 * 0.8$). Thus, the factors weigh each machine to account for variations in processing power.

Force Starting Jobs

If you force start a job, Unicenter AutoSys JM will start the job right away on the machine specified in the job definition, regardless of the current load on the machine or the job_load specified for the job. If the job was defined to run on a virtual machine, or a list of real machines, Unicenter AutoSys JM will determine which machine has the most processing power available and will run the job on that machine, even if the job_load of the job exceeds the max_load defined for the machine.

Note: If you FORCE_START a job that has a status of ON_ICE or ON_HOLD, upon completion (either success or failure), the status/condition does *not* change back to the previous condition.

For example: You scheduled Job-1 to run every Monday at 3:00 A.M, however, on Sunday you placed this job ON_HOLD. If you FORCE_START Job-1 on Wednesday at 2:00 P.M., Job-1 will run to completion (either success or failure), and then run again as scheduled on Monday at 3:00 A.M.

Load Balancing Using ServerVision

Unicenter AutoSys JM can use the machine performance metrics monitored by ServerVision to select, at runtime, the best machine on which to execute a job.

You supply a command that executes a specified algorithm to the machine job attribute, and this command returns the name of the best machine (based on the given algorithm).

Before using this feature, ensure the configuration has been completed as described in Configuring ServerVision Monitoring and Reporting in the chapter “Configuring.” For information on ServerVision, see the ServerVision documentation on the documentation CD.

To apply ServerVision load balancing to a job, specify the following string in the machine job attribute, or to the Machine to Execute field in the Job Definition screen:

```
'svload -a alg [-v virt | -l list] -p profile 2> filename'
```

Note: This string cannot exceed 80 characters.

where:

svload	Is the svload executable
alg	Is the method to be used by ServerVision to select the best machine. This algorithm must be defined in the profile file that you reference with the -p argument. For example, the algorithm you define could provide one method that would be used for I/O-bound jobs and a different method that would be used for cpu-bound jobs.
virt	Is a virtual machine that has been defined.
list	Is a comma-separated list of real machines.
profile	Is a file containing basic information and the performance metrics that apply to each possible algorithm that can be chosen. You can supply a path and filename for this option. To know what the ServerVision profile file may look like, see Examining the ServerVision Profile File.
2> filename	Specifies to redirect any standard error messages to a file.

Examining the ServerVision Profile File

The profile file for ServerVision load balancing may look like this example svload.prf file:

```
[ServervisionConfiguration]
Timeout=12
InstanceType=unix
honda=honda
toyota=toyota
jeep=jeep

[CPU]
cpu_t1,idle,maximum,1

[Memory]
memory,used,minimum,1

[CPUMemory]
cpu_t1,idle,maximum,2
memory,used,minimum,1
```

where:

[ServervisionConfiguration]	Is a literal that must appear exactly as shown in this example.
Timeout	Is the number of seconds ServerVision should attempt to connect to a machine. The recommended setting is 12 seconds.
InstanceType=unix	Indicates the instance type. For this implementation, the value should be <i>unix</i> . For more information.
honda=honda	Are examples of names of the real machines defined with the name of the corresponding ServerVision instance. It is in the form of <i>machine_name=ServerVision_instance</i> . Typically, the ServerVision instance will have the machine name also.
[CPU], [Memory], and [CPUMemory]	These specify examples of names and definitions of the algorithms. You can create your own definitions, using any name, and you can have as many definitions in this file as you want. Each algorithm definition must use the Scan Groups as defined by ServerVision. The syntax must be a colon-separated string with the following elements, in the following order: <ul style="list-style-type: none"> ■ Scan Type Indicates the Scan Type as defined by ServerVision. ■ Scan Object Indicates the Scan Object that is associated with the Scan Type as defined by ServerVision.

- **maximum or minimum**
Indicates the value that is desired, either the maximum number or the minimum number.
- **Weight**
Indicates the item's importance in relationship to other items that make up the algorithm. For example, in the CPUMemory definition above, the CPU item is indicated as having twice the weight of the memory item.

Testing the Algorithms

You can run the following command to test the algorithms you enter in the svload.prf file:

```
svload -a alg [-v virt | -l list] -p profile -y
```

This command prints to the screen the result metrics. You can use this information when you are creating the profile file and want to test your algorithm definitions.

WARNING! *The -y argument is for testing only. Do not use it when using this command as the value for the machine job attribute. When testing the algorithm, do not include the 2> filename argument.*

ServerVision Monitoring and Reporting

You can also use ServerVision to monitor AutoSys job resource usage in real time. Jobs will appear in the ServerVision GUI by job name. Unicenter AutoSys JM can also archive a job's resource usage, use ServerVision to generate reports for capacity planning and UNIX process auditing (charge back). This functionality requires that ServerVision integration be enabled as described in Configuring ServerVision Monitoring and Reporting in the chapter "Configuring." For details on monitoring job resource usage using ServerVision, see Monitoring Job Resource Usage in the same chapter.

Queuing Jobs

Queuing jobs in AutoSys is a mechanism for ordering jobs that are unable to be run immediately. You can also issue a "change priority" event to change the priority of a job in the queue. There is no actual "queue" entity. Instead, jobs are chosen based on queuing policies.

Queuing policies are established through the use, and subsequent interaction, of the two job attributes job_load and priority, and the two machine attributes max_load and factor.

The following sections discuss queuing jobs and give examples of how load balancing and queuing are used to optimize job processing in your environment.

Queuing and Simple Load Limiting

If a job has been assigned a job_load value (the load limiting feature), and a max_load attribute is assigned to every real machine comprising a virtual machine, Unicenter AutoSys JM will first determine whether each machine has sufficient available load units before running the job. If each real machine has sufficient load units, Unicenter AutoSys JM employs the load balancing and "factor" algorithms to determine on which machine it should start. However, if only one of the machines has sufficient load units, the job will be run on that machine. If no machines have sufficient load units, the job will be placed "on queue" for both (or all) machines. When one machine becomes available, the job is run on that machine, and removed from all other queues.

The words “in the queue” refer to an actual QUE_WAIT job status, and the job will stay in this state until the necessary load units become available.

When the necessary load units become available, Unicenter AutoSys JM again checks all the job’s starting conditions to ensure it is still okay to run the job. If any of the starting conditions are no longer true, the following message is generated:

```
Job: job_name Starting Conditions are no longer TRUE.  
De-Queuing this Job and setting to ACTIVATED.
```

Note: In order for any queuing to take place, all jobs must have their priority attribute set. By default, the priority attribute is set to 0 indicating that the job should not be queued, but be run immediately. When this is the case, even jobs whose job load would push the machine over its load limit will be run. However, it is important to note that even when jobs have a priority of 0, job loads will still be tracked on each machine. This is done so that jobs that do have nonzero priorities will still be queued.

Using a previously defined machine named “fiat” with a max_load of “100,” a simple queuing example would be as follows:

```
insert_job: jobA  
machine: fiat  
job_load: 80  
priority: 1  
insert_job: jobB  
machine: fiat  
job_load: 90  
priority: 1
```

If “jobA” was running when “jobB” started, “jobB” would be in a QUE_WAIT state until “jobA” completed and “jobB” could run.

Note: If a job is in the QUE_WAIT state and you want to run it immediately, do not force start the job. To change the job queue priority, use the sendevent command with the -E CHANGE_PRIORITY option.

Queuing with Priority

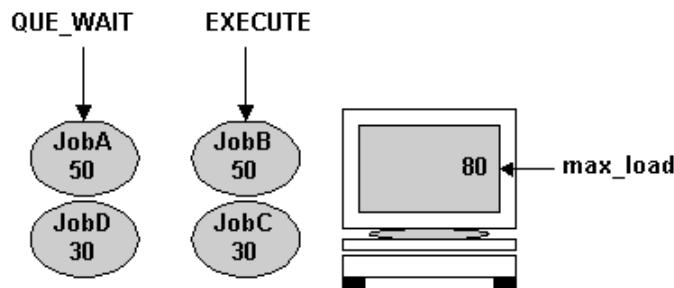
When more than one job is queued, the priority value is considered first when deciding which job to run next. If there are insufficient load units (job_load value) available to run the highest priority job, it will remain in the queue, and lower priority jobs will be considered subsequently.

When there is no priority value assigned to a job (default is 0), there is no queuing and Unicenter AutoSys JM starts the job immediately. Therefore, the job will never go into the QUE_WAIT state. If the job's priority was set to run immediately, the job would run regardless of the current load on the machine. Obviously, this interferes with the load-limiting feature, so when load limiting is in use, the priority attribute should always be set.

In the case where a job has its job_load attribute set, the load value will be reflected in the total load running on a machine. It is important to note that if there is no job_load value set for a job, it will not be figured into the total load units running on a machine.

Note: The constructs of job loads and machine loads are merely conventions that you set up, and that are enforced by Unicenter AutoSys JM. If you do not indicate what the load of a job is, it will not figure it into its queuing calculations. This is different from the load-balancing feature, which does inspect the CPU to determine its usage.

The following illustrates a situation where a machine has 80 load units, and multiple jobs are waiting to start. In this example, “JobB” and “JobC” are executing while “JobA” and “JobD” are “queued” (in the QUE_WAIT state), waiting for available load units. The numbers in the figure indicate the job_load assigned to each job, and the max_load of the machine. The JIL statements provided below define the machine and the jobs.



```
insert_machine: ferrari
max_load: 80
insert_job: JobA
machine: ferrari
job_load: 50
priority: 60
insert_job: JobB
machine: ferrari
job_load: 50
priority: 50
insert_job: JobC
machine: ferrari
job_load: 30
priority: 80
insert_job: JobD
machine: ferrari
job_load: 30
priority: 70
```

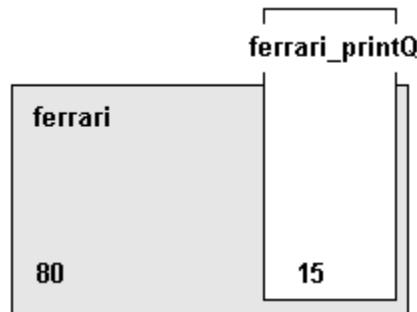
In the above scenario, “JobB” and “JobC” are already running because their starting conditions were satisfied first. After “JobB” or “JobC” are completed, “JobA” or “JobD” will start. Which job will start, “JobA” or “JobD,” is determined by a combination of the priority and job_load attributes of each job, and the max_load machine attribute. The resulting scenario will differ, based on which job finishes first.

If “JobB” finishes first, 50 load units become available, so either “JobA” or “JobD” could be run. Since “JobA” has a higher priority (lower value = higher priority), it will run first. However, if “JobC” finishes first, only 30 load units become available, so only “JobD” could be run.

Subsets—Individual Queues

One variety of virtual machine can be considered a subset of a real machine. Typically, this type of virtual machine is used to construct an individual queue on a given machine. One use for this construct might be to limit the number of jobs, of a certain type, that will run on a machine at any given time. For example, you have created three different print jobs, but you want only one job to run on a machine at a time. You can accomplish this by using a combination of the `max_load` attribute for the virtual machine and the `job_load` attribute for the jobs themselves.

The following illustration depicts a virtual machine functioning as a queue. The JIL statements to define the queue, called “`ferrari_printQ`” follow the graphic. Note that “`ferrari`” is a real machine.



To implement the schema in the previous illustration, you first create the virtual machine named “`ferrari_printQ`,” like:

```
insert_machine: ferrari_printQ
machine: ferrari max_load: 15
```

Next, you define the three print jobs, like:

```
insert_job: Print1
machine: ferrari_printQ
job_load: 15
priority: 1

insert_job: Print2
machine: ferrari_printQ
job_load: 15
priority: 1

insert_job: Print3
machine: ferrari_printQ
job_load: 15
priority: 2
```

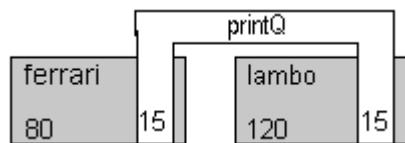
Using this definition, only one of the jobs would run on “`ferrari`” at one time, since each job requires all of the load units available on the specified machine.

Load Units and Virtual Machines

It is important to note that the load units associated with a virtual machine have no interaction with the load units for the real machine. In the previous example, this means that the virtual load of 15 does not subtract from the load units of 80 for the real machine. Load units are simply a convention that allows the user to constrict concurrent jobs running on any one machine.

Multiple Machine Queues

Virtual machines can also be constructed to allow subsets (or slices) of real machines to be combined into one virtual machine. A possible need for this would be if there were two machines that were print servers, on each of which only one print job was to run at a time. The following illustration demonstrates this situation.



To implement the previous schema, you would first create the virtual machine named “printQ,” then you would specify two real machines, “ferrari” and “lambo” as shown in the following example:

```
insert_machine: printQ
type: v
machine: ferrari
max_load: 15
machine: lambo
max_load: 15
```

As a job is logically ready to start on printQ, Unicenter AutoSys JM will determine if there are enough load units available on either machine. If there are not, it will place the job in the QUE_WAIT state, and start it when there are enough load units. If there are enough units on only one machine, it will start it on that machine. In the case that there are enough available load units on both machines, Unicenter AutoSys JM will determine the usage on each, and start the job on the machine with the most available CPU resources.

User-Defined Load Balancing

As an alternative to using the provided load balancing methods described in this chapter, you can write your own programs or scripts to determine which machine to use at runtime. If you specify the name of a program or script as the machine name in the job's machine specification, the event processor will execute the script at job runtime, and it will substitute its output for the machine name. For example, you might supply the following:

```
insert_job: run_free
machine: '/usr/local/bin/pick_free_mach'
command: $HOME/rm_stuff
```

At runtime, the script /usr/local/bin/pick_free_mach is run on the event processor machine. The standard output will be substituted for the name of the machine, and the job will be run on that machine.

Note: If you specify a user-defined load balancing script in the machine attribute, you cannot use the priority or job_load job attributes.

This chapter describes how to use the Operator Console to monitor and control job activity in real-time. It also describes the job selection and reporting features of the console, as well as the Alarm Manager. Customizing the Operator Console is also covered.

The Operator Console provides a sophisticated method of monitoring jobs in real-time. The Operator Console lets you view any jobs that are defined, whether they are currently active or not.

Job selection criteria, which you can dynamically change, allows you to control which jobs you want to view based on various parameters, such as the current job state, the job name (with wildcarding), and the machine on which the job runs. You can select any job and view more detailed information about it, including its starting conditions, dependent jobs, and autorep reports. You can even invoke the Job Definition dialog directly from this window and change the job, if the correct permissions are set.

Alarm Manager

In addition to its job monitoring capabilities, the Operator Console provides an Alarm Manager, which lets you monitor alarms as they are generated. You can manage alarms by doing the following:

- Entering responses directly at the Alarm Manager dialog.
- Setting the alarm's state to either acknowledged or closed. (If an alarm's state is "open" and you simply acknowledge it without closing it, it will be set to "acknowledged.")

Alarms and their responses are stored in the database, from which they can be retrieved for viewing, or for adding additional responses. You can dynamically select which alarms you want to view based on such criteria as alarm type, alarm state, and the date and time range in which the alarm was generated.

Operator Console Screens

The Operator Console consists of the following elements:

Operator Console Screens	Actions
Job Activity Console	The primary interface to the Operator Console. This console lets you monitor any jobs.
Job Selection	Dialog used to specify which jobs you want to view.
Alarm Manager	Dialog allows you to browse any alarms, which have occurred, and to acknowledge (and log a response to) those alarms.
Alarm Selection	Dialog used to specify which alarms you want to view.

Starting the Operator Console

If you are reading this chapter for the first time, you should start the Operator Console before continuing. This way you can follow along with the descriptions in the following sections using your own system.

To start the Operator Console, use one of the following two methods:

1. At the UNIX prompt, enter the following command:

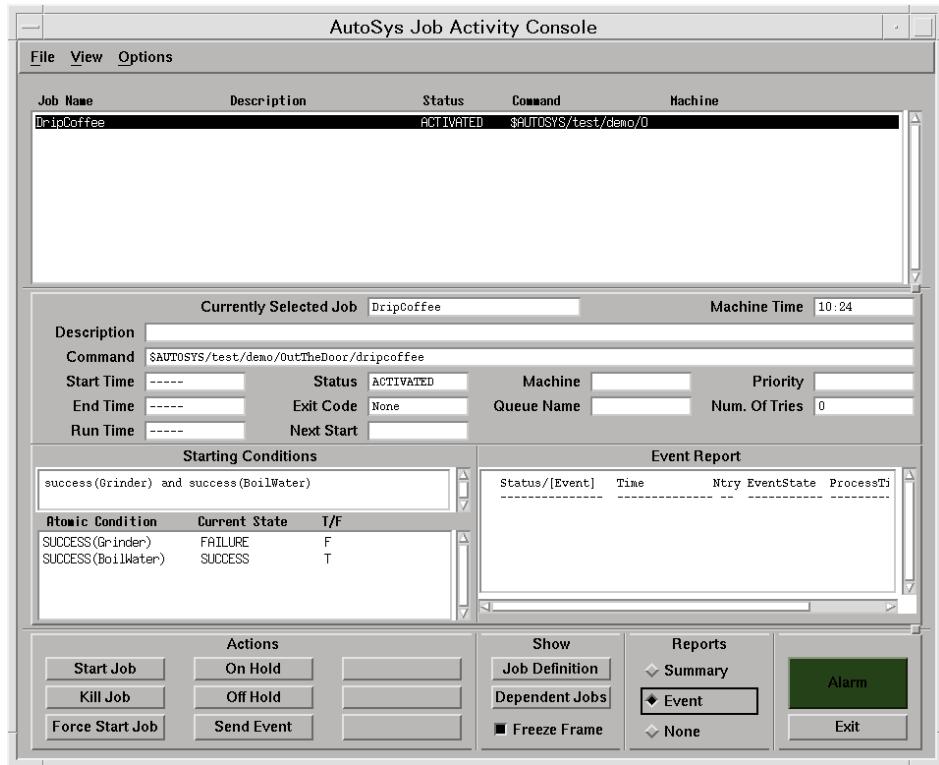
```
autocons &
```

2. At the GUI Control Panel, click Ops Console.

The Job Activity Console appears.

Job Activity Console

The first screen that displays when the Operator Console starts up is the Job Activity Console:



The Job Activity Console has a menu bar and the following three regions:

- **Job List**
Displays a list of all jobs stored in the database, subject to the job selection criteria currently in effect.
- **Currently Selected Job**
Displays more detailed information about the currently selected job.
- **Control Area**
The bottom portion of the Job Activity console is the Control area. The left side of this area contains buttons that act on the currently selected job; the middle contains buttons that act on the console screen; the right side contains the Alarm button, which displays the Alarm Dialog, and Exit to exit the Job Activity Console.

By default the Job Activity Console starts up in Freeze Frame mode, which prevents the display from regularly updating and refreshing. To observe changes as they occur, click Freeze Frame.

Menu Bar

At the top of the Job Activity Console is the menu bar, containing three menus: File, View, and Options.

File Menu

Contains the Exit option, which functions exactly like the Exit button in the Control area—it displays a verification dialog asking you to confirm the exit. If you confirm, the Job Activity Console is closed. If the Alarm Manager was open, it will be closed as well.

View Menu

Contains the Select Jobs option, which displays the Job Selection dialog, discussed in Job Selection Dialog in this chapter.

Options Menu

Contains the Console Clock Perspective option. You have three choices: Server Time, Current Job Time, or Local Machine Time. This option controls the time perspective of the display, as discussed in the [10](#).

Job List

The Job List region displays a list of all the jobs that are defined, subject to the job selection criteria currently in effect.

Each entry in the Job List contains the following information about a single job:

- Job name.
- Description.
- Current status.
- Command that is defined for this job, if it is a command job. If it is a file watcher job, the file to watch for appears in the Command column. If it is a box job, the Command column is empty.
- Machine on which the job ran or is currently running.

The entries in the Job List provide a snapshot of the entire system, across multiple machines.

When you select a job from the list, the highlighted job becomes the currently selected job, and more detailed information about the job appears in the Currently Selected Job region.

To select a job:

Click the job in the Job List display.

When you select multiple jobs, you can perform actions on all those jobs at the same time.

To select contiguous multiple jobs:

Press and hold the mouse button and drag to select a group of jobs.

To select noncontiguous multiple jobs:

Hold the Control key and click each job you want to select. Hold the Control key and clicking a selected job will deselect that job.

To deselect all the currently-selected jobs:

Click anywhere in the job list.

Note: The Job List region has a scroll bar along the right side for scrolling through the job list. Using the X resource file, you can configure the relative sizes of the columns in the Job List, as well as the length of each field and the spacing between fields.

Currently Selected Job

The Currently Selected Job region displays information about the most recent run (or the current run) of the currently selected job. If you have selected multiple jobs, the first job you selected will be the currently selected job. (Freeze Frame must be deselected in order for the information in this region to update in real time.) In this region, you will find the following fields:

Currently Selected Job

This field displays the name of the currently selected job.

Console Clock

The console clock initially displays the current time for the machine on which the Operator Console is running. You can change this display by using the Options menu (see Changing the Console Clock Perspective in this chapter) or by changing the setting in the resource file (see Console Clock Perspective in this chapter).

Description

If the job definition includes the description attribute, the text appears in this field.

Command

If the currently selected job is a command job, the command appears here. If it is a file watcher, the file it is watching for appears here. If it is a box job, this field is blank.

Start Time

The start time of the current or most recent run of the job.

End Time

The end time of the most recent run of the job. If the job is currently running, this field will be blank.

Run Time

How much time elapsed between the start and end of the most recent run of the job. If the job is currently running, this field will be blank.

Status

The current status of the job.

Exit Code

The exit code from the most recent run of the job.

Next Start

If the job has date and time starting conditions, this field shows when the next run of the job is scheduled to start.

Machine

The name of the machine on which the job ran or is currently running. If a job is defined to run on a virtual machine, the name of the real machine component on which it actually ran will appear here.

Queue Name

If the job is queued to start on a machine, the name of that machine appears here.

Priority

If the job is queued to start on a machine, its priority in the queue appears here.

Num. of Tries

If the job had to be restarted, the number of times it was started appears here.

Starting Conditions

The Starting Conditions area displays the job's entire starting condition, as specified in its job definition, as well as the "atomic" conditions—the most basic components of an overall condition. This information is very useful when troubleshooting a job.

For example, in the sample Job Activity Console, shown in Job Activity Console in this chapter, the job named "DripCoffee" has a starting condition called:

- SUCCESS(Grinder) and SUCCESS(BoilWater)

This starting condition is specified in the job's definition. However, this starting condition is actually composed of the following two atomic conditions:

- SUCCESS(Grinder)
- SUCCESS(BoilWater)

In the Starting Conditions area, each atomic condition is displayed with the Current State of the job upon which it is based; in our example, "Grinder" and "BoilWater," respectively. Also, a "True/False" flag is provided that indicates whether or not that atomic starting condition has been satisfied.

If a job has not run within the time frame it was expected to, you would select the job from the Job List and check its starting conditions to quickly determine what “upstream” job might be preventing it from running.

The atomic condition list is selectable. By clicking any one of the atomic conditions, the job associated with that condition will become the currently selected job, and its details will be displayed in the middle region of the screen. This feature allows you to quickly step through upstream dependencies, checking out each job along that path.

Reports

The Reports area displays a realtime report, and it is also included in the Currently Selected Job region. This report presents job run information in the same format as that produced by the autorep command. You can choose from the following report types:

Summary

A one-line synopsis of the last or current execution of the job showing the job name, timestamp of the last start and last end of the job, status, job run number and number of tries (separated by a slash), and priority (if the job is in QUE_WAIT status) or exit code (if the job completed).

Event

A detailed report listing all the events and statuses from the last or current execution of the job. The screen shown in Job Activity Console in this chapter shows an Event Report.

None

Does not display a report.

Summary and Event reports will be run automatically each time the dialog is refreshed. The default refresh interval is every five seconds, but the interval is user-configurable. If the Event report is chosen, you can watch the realtime progression of a job, observing, as they occur, the arrival of the various events, such as the job starting, running, completing, and restarting.

Control Area

The bottom region of the Job Activity Console is the Control area.

Action Buttons

On the left side of this region is a group of push buttons that can be pressed to initiate certain actions on the currently selected job or jobs. By pressing the appropriate button, you can issue an event that will:

- Start a job.
- Kill a job.
- Force a job to start.
- Place a job on hold.
- Take a job off hold.

In each of these cases, a dialog box asks you to confirm, after which the action is taken immediately, without requiring you to perform any further actions. If you have initiated an action on multiple jobs, the dialog will display the following:

"Ready to send event *event* for # jobs"

where:

event

Is the action you chose.

#

Is the number of jobs that are selected.

When you click Send Event, the Send Event dialog displays, and it allows you to send any type of event.

In the last column are buttons that are user configurable. You can associate any command with these buttons, and specify your own button labels. This process is explained in User-Configurable Action Buttons in this chapter.

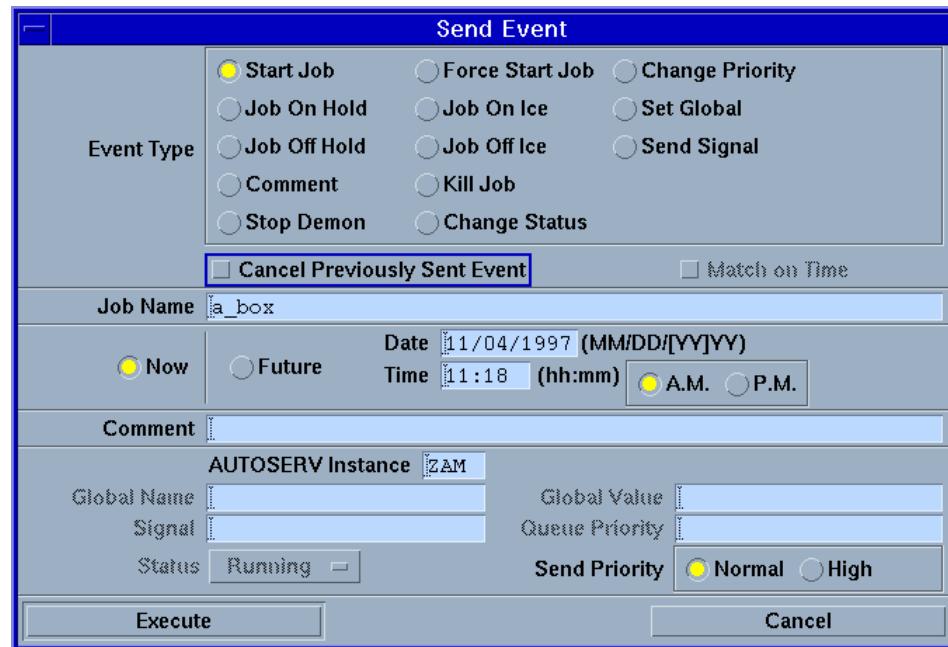
Send Event Dialog

The Send Event dialog box provides you the means to do the following:

- Send any event that can be sent manually in AutoSys.
- Select the various event parameters you want to specify when sending the event.
- Cancel an event that has been scheduled to occur in the future.

Note: The fields of the Send Event dialog correspond to sendevent command options. For a complete description of these options, see the description of the sendevent command in the chapter “Commands” of the *Unicenter AutoSys Job Management Reference Guide*.

The following is the Send Event dialog:



You specify an event using one of the radio buttons at the top of the dialog. Just below these buttons is the Job Name field, which by default contains the name of the currently selected job. You can change this field if desired.

You can specify when the event is to take effect, either Now (the default), or at some future time and date. (The current time and date are provided as examples of the required format.) Use the A.M. and P.M. radio buttons if you want to specify the time using a 12-hour format. If the time field contains an hours setting that is less than 13, it is considered a.m., while any larger value is considered p.m.

The Comment field is a free-form field in which you can enter any text you want to associate with this event in the database; this field is for documentation purposes only. For example, if you force a job to start, you might provide an explanation about why this was necessary.

The AUTOSERV instance field displays the current server identifier; only when events need to be sent to a different instance should this field be changed.

The Global Name and Global Value fields are used when you have specified a SET_GLOBAL event. Global Name and Global Value can each be a maximum of 30 characters.

The Signal field is used to specify the signal number if you specified a SEND_SIGNAL event.

The Queue Priority field is used only when you have specified a CHANGE_PRIORITY event. This affects the run priority of a job that is in QUE_WAIT state.

You can only make a selection from the Status pull-down menu if the Change Status event type (radio button) has been selected. This menu lets you select a new status for the currently selected job.

Use the Send Priority radio buttons to specify whether the event is to be sent with normal priority (the default), placing the event in the queue with all system-generated events, or with high priority, placing it at the top of the event queue. The latter is normally reserved for emergencies, such as killing a job.

The Execute button of the dialog executes, or sends, the event. The Cancel button cancels the event that was about to be sent. When either button is pressed, the Send Event dialog is dismissed.

Cancelling a Sent Event

At the Send Event dialog, you can cancel one or more events scheduled to occur sometime in the future. You can do this in one of two ways: by canceling a specific event or by canceling a specific event type for a specific scheduled time.

Note: You should use this feature to cancel events that you have sent from the Send Event dialog. If you want to override a scheduled starting condition for a job, you should use the one-time override job attribute, either from the Job Definition dialog or from JIL.

To cancel a specific event:

1. In the Event Type region, specify an event type by selecting one of the radio buttons.

Note: You can select multiple jobs in the Job Activity Console before you open the Send Event dialog. If you do so, the Send Event dialog will send this cancel event for all of the selected jobs that meet the Event Type criteria.

2. Select the Cancel Previously Sent Event radio button.
3. In the Job Name field, enter the job name.
4. Click the Execute button. This process cancels all pending events of the specified Event Type for the selected jobs.

To cancel a specific event by its scheduled time:

1. Make sure you have the appropriate job in the Job Name field.

Note: You can select multiple jobs in the Job Activity Console before you open the Send Event dialog. If you do so, the Send Event dialog will send this cancel event for all of the selected jobs that meet the Event Type and Time criteria.

2. In the Event Type region, specify an event type by selecting one of the radio buttons.
3. Select the Cancel Previously Sent Event radio button.
4. Select the Match on Time radio button.

5. In the Time field (of the Future region), specify the time the event is scheduled to occur.
6. Click the Execute button. This cancels all pending events of the specified Event Type at the specified Time for the selected jobs.

Notes on Canceling a Send Event

The Cancel Previously Sent Event feature is designed to be used primarily on events that you have sent from the Send Event dialog. If you want to override a scheduled starting condition for a job, you should use the one time override job attribute, either from the Job Definition dialog or from JIL.

If you cancel a future Start Job event for a time-dependent job with no other starting conditions, the job may never run again without manually starting it with a Send Event command. For example, “jobA” is scheduled to run daily at 11:00. “jobA” starts at 11:00 on Monday and completes at 11:30, at which time the next future Start Job event is sent for 11:00 Tuesday. At 9:00 on Tuesday, you cancel the 11:00 Start Job event. The job not only does not run at 11:00 on Tuesday, but it will not be scheduled to run again. To restart the job, you can either update its job definition, or manually issue a Start Job Send Event.

Control Buttons

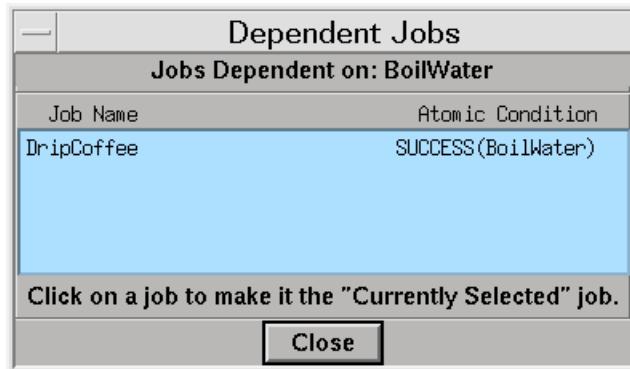
In the middle of the Control area, there are several push buttons that provide you with control functions for the Console screen. The Job Definition button displays the Job Definition dialog with the currently selected job already displayed. This allows you to quickly review the job’s definition, and change it if necessary (and if permissions allow).

The Dependent Jobs button displays the Dependent Jobs dialog that contains a list of all the jobs directly dependent on the currently selected job. This allows you to quickly see which jobs will be affected by the current job; in particular, which jobs will not run until the current job completes. This is useful if the currently selected job is running late and you need to determine which other jobs will be affected.

The following Dependent Jobs dialog is for the job “BoilWater.” As with the atomic conditions list, any job in this Dependent Jobs List can be selected, making it the currently selected job (and dismissing the dialog).

The Dependent Jobs dialog can be dismissed by pressing the Close button, or by selecting another job in the Job Activity Console.

The following is the Dependent Jobs dialog for the job ‘BoilWater’:



This feature allows you to arbitrarily follow the chain of job dependencies far downstream, and to determine which jobs are in some way dependent on another job.

The Freeze Frame button freezes the Console display, which otherwise is regularly updated. By default it is updated every five seconds; this refresh interval is user-configurable. In Freeze Frame mode, all processing continues normally, but the screen is not refreshed. This feature is useful, for instance, when you are viewing the Event Report’s output and the display has scrolled through some of the output. A refresh operation would reset the report display to the first line of output, forcing you to scroll back to the area that you were viewing. When the Freeze Frame button is toggled back off, the Console once again reflects the current state of the system.

There is a user-configurable X resource that causes the Operator Console to start in Freeze Frame mode—this is the installation default. For more information about customizing the Operator Console, see Customizing the Operator Console in this chapter.

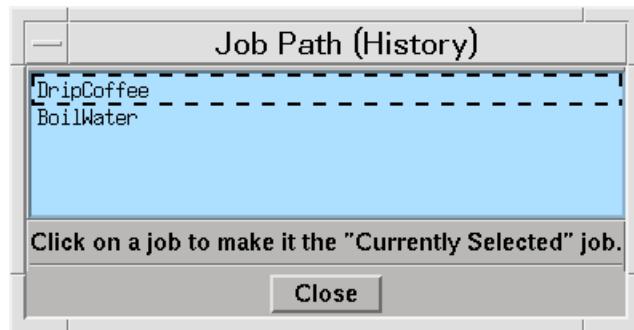
The three Report buttons let you choose the type of report you want to view, as described earlier in this section. You can specify a default report type using the X resources, see Default Report in this chapter.

Job Path (History) Dialog

When you single-click on a job in the atomic conditions list or in the dependent jobs list (thereby changing which job is the currently selected one), the Job Path (History) dialog appears. This dialog contains a list of all the jobs selected since the last time a job was selected directly from the Job List, in the order in which they were selected.

Assuming that you had clicked the atomic condition for “BoilWater” while displaying the “DripCoffee” job, the new currently selected job would be “BoilWater,” and the Job Path (History) dialog would display with the appropriate entries for both of the jobs you have traversed since your last selection from the Job List.

The following shows a Job Path (History) dialog box with these entries.



Using this dialog, you can quickly return to any previously selected job by clicking on the job's name.

Alarm Button

The large Alarm button serves both as an indicator that a new alarm has been detected and as way to display the Alarm Manager dialog. When a new alarm occurs, the Alarm button changes to the color red. When this happens, and you press the button, its color returns to green, and the Alarm Manager dialog is displayed. If the Alarm Manager dialog is already on the screen, but is obscured by the Job Activity Console, pressing the Alarm button will bring the Alarm Manager dialog to the top of the display. The Alarm button can also be used to update the Alarm Manager dialog, even if Freeze Frame is in effect.

Exit Button

The Exit button is used to close the Job Activity Console, including the Alarm Manager. When this button is pressed, a verification dialog displays asking you to confirm the exit.

Resizing Regions of the Job Activity Console

The resizing handles of the Job Activity Console let you move the boundary lines between the three regions of the window. You change these boundaries using the small square handle at the right edge of each region's horizontal separators; these separators are blue by default.

To adjust a boundary, you simply click and hold on a resizing handle and drag it up or down. By doing so, your screen display can be adjusted to reveal more jobs. For example, you can borrow display space from the Currently Selected Job region, and even the Control area, to view more jobs in the Job List. In fact, if the upper handle is pulled all the way down to the bottom of the Console screen, the entire Job Activity Console can be used to display the Job List. You can expose the buttons in the Control area later by pulling the lower handle back up. (You can alter the Job Activity Console display size using the resizing handles surrounding the dialog as well.)

Job Selection Dialog

The Job Selection dialog provides a way to specify which jobs you want to view, filtering out all other jobs. You specify the jobs you want to view by name, job status, and machine on which the job ran (or is currently running).

To display the Job Selection dialog:

From the Job Activity Console, choose View, Select Jobs.

The Job Selection dialog appears:



Specifying a Job by Name

If you select the All Jobs option, all jobs are selected, ignoring any entries in the Job Name or Box Name field. However, you can select jobs by name.

When specifying a job name in the Job Name field, you can enter either the entire name or a partial name with the asterisk (*) wildcard character representing one or more characters. You can use this wildcard in more than one character position. For example, specifying the job name “*a*” would match the jobs “Dad,” “Bead,” “Bat,” and so forth.

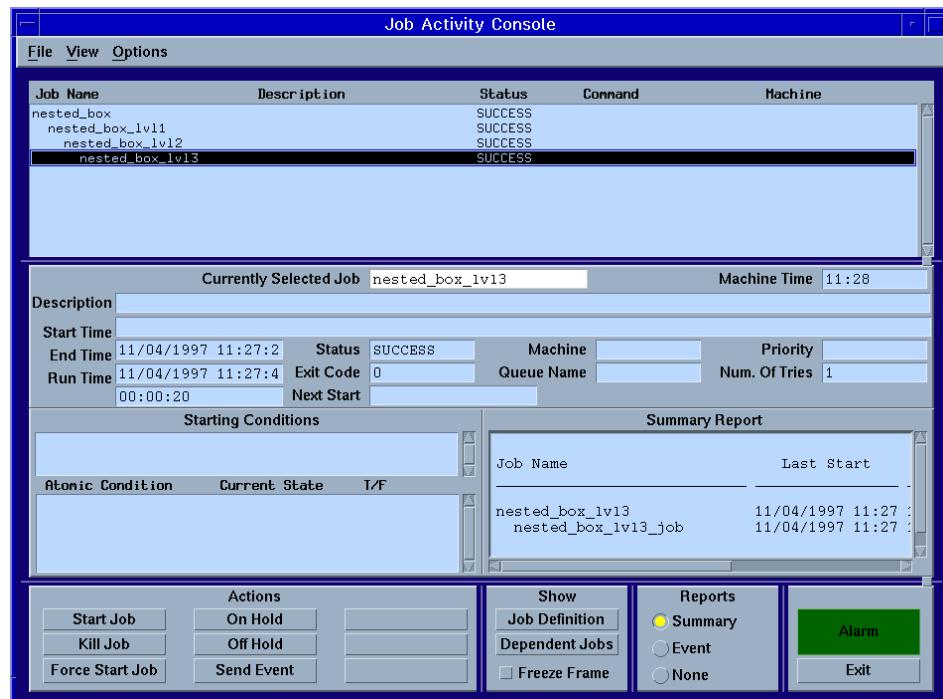
In a similar fashion, you can specify a box name in the “Box Name” field to select all jobs in the specified box.

Note: If you use the wildcard character to specify all box names, and a box has other boxes inside of it, the name of the nested box job will be listed multiple times in the Job List.

The Box Levels field lets you indicate how many levels of nesting you want to view for a box job. You can enter any valid positive number.

Nesting Level	Description
0	Indicates that only the top-level box specified in the Box Name field is to be displayed.
1	Indicates that the box specified in the Box Name field and only the first level nesting of its direct descendants are to be displayed.
All	Indicates that the box specified in the Box Name field and all of its direct descendants (including nested boxes and the jobs in those boxes) are to be displayed. This is the default.

When selecting jobs based on box name, each level of box/job will be indented two spaces to indicate the nesting. The following shows this convention in the Job List:



Specifying Jobs by Status

From the Job Selection dialog, you can select jobs based on their current status, such as STARTING, RUNNING, INACTIVE, and so forth. The default is to display all job statuses. You can select any combination of the statuses. All Statuses overrides any specific status selections.

Specifying Jobs by Machine

From the Job Selection dialog, you can select jobs based on the name of the machine on which it ran (or is currently running). On the right side of this dialog is a list of all the machines that are referenced in any job or machine definition, or which have run an AutoSys job. From this list, you can choose one, several, or All Machines (the default).

Note: Virtual machines will appear in this list, but cannot be used to select jobs because jobs can run only on real machines.

Selecting Machines

To choose a single machine:

Click on that machine.

To choose a range of machines:

Click and hold on the first machine name, drag the cursor to the last name, then release the mouse button.

To choose additional machines after the initial selection:

Hold down the Control key and perform the actions previously shown.

To choose all machines:

Select the All Machines option.

Sorting the Specified Jobs

The Job Selection dialog lets you specify the sort order in which the jobs should be listed. You can choose one of the following sort criteria:

Sort Order	Description
Start Time	The starting time for the most recent execution of the job.
End Time	The ending time for the most recent execution of the job.
Job Name	Jobs will be sorted by name in ascending alphanumeric order.
Job Status	Jobs will be sorted by their current status, in ascending alphabetical order.
Machine Name	Jobs will be sorted by the machine on which they run or have been run, in ascending alphabetical order.
Unsorted	Displays the jobs in the order in which they were created; that is, the order in which they exist in the database. You should choose this option when selecting jobs by box name, because their creation order usually has some significance; especially as this pertains to the running order of jobs inside of boxes. If any sorting is done, the indenting of the various nesting levels has no meaning, nor is there any indication of which jobs are in which box.

Setting the Job Selection Criteria

To make the job selection criteria take effect:

Do one of the following:

- Click OK, which also dismisses the dialog.
or:
■ Click Apply, which does not dismiss the dialog.

Clicking the Cancel button dismisses the Job Selection dialog without changing the selection criteria.

Changing the Console Clock Perspective

The Options menu of the menu bar contains a single option, Console Clock Perspective. You have three choices: Server Time, Current Job Time, or Local Machine Time. This option controls the time perspective of the display.

Note: The Console Clock Perspective does not change the Start Time and End Time displays in the Currently Selected Job region.

Server Time displays the time based on the time zone of the AutoSys server machine. Current Job Time displays the time using the time zone specified in the job definition (the `timezone` attribute); if no time zone is set for the job, then the server machine time is used. Local Machine Time displays the time in the time zone of the machine on which the Operator Console is running.

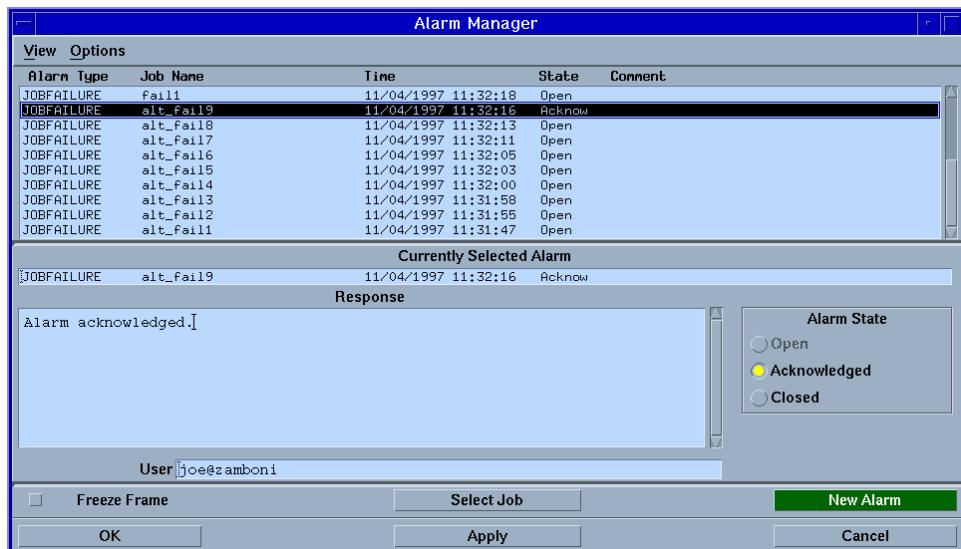
Alarm Manager Dialog

The Alarm Manager lets you view alarms as they arrive, acknowledge them, and change their status from Open to Acknowledged or Closed. This feature provides a useful tracking mechanism for all the alarms issued on your system.

To display the Alarm Manager dialog:

Press Alarm in the lower-right corner of the Job Activity Console.

The Alarm Manager dialog appears:



The Alarm Manager dialog has a menu bar and the following three regions:

Alarm List

At the top of the dialog.

Currently Selected Alarm

In the middle of the dialog.

Control

At the bottom of the dialog.

Alarm Manager Menu Bar

The Alarm Manager menu bar, at the top of the dialog, contains two menus: View and Options.

Alarm Manager Menu	Action
View Menu	Provides two options: Select Alarms and Reset to Defaults. The Select Alarms option displays the Alarm Selection dialog. The Reset to Defaults option resets the alarm selection criteria to their default settings and updates the Alarm Manager dialog to reflect the new selection criteria.
Options Menu	Contains the single selection, Sound On. The “Sound” feature plays sound clips associated with alarms whenever a new alarm is generated. This option is a toggle switch which can be turned on and off by clicking the option.

Alarm List

The Alarm List region of the dialog displays a list of all the alarms that are currently in the system, and that meet the viewing criteria specified by the user, which may include closed alarms. The default is to display all Open and Acknowledged alarms, of any type, regardless of the time they were generated.

Each entry in the Alarm List contains the following information about a single alarm:

- Alarm type.
- The job for which the alarm was generated.
- Date and time at which the alarm was generated.
- The alarm's current state.
- Any comment associated with the alarm at the time it was generated.

Alarms are displayed in reverse order of occurrence; the newest alarms appear at the top of the list and older ones appear farther down. An alarm is made the currently selected alarm by clicking anywhere on the line on which it is displayed.

Selecting multiple alarms allows you to perform actions on multiple alarms at once.

To select multiple alarms:

Do one of the following:

- Press and hold the mouse button and drag to select a group of alarms.
- Hold the Control key and click each alarm that you want to select. Holding the Control key and clicking a selected alarm will deselect the alarm.

Clicking anywhere in the alarm list will deselect all currently selected alarms.

Note: You can configure the widths of the columns in the Alarm List using the X resource file, described in Alarm List Column Width in this chapter.

Currently Selected Alarm

The Currently Selected Alarm region of the dialog displays the currently selected alarm, and lets you enter a response in the Response edit box.

The Response edit box accepts multiple lines of text. The entered text is automatically word wrapped, with lines breaking at appropriate spaces. You can use the mouse to edit text. In addition, you can use the arrow and backspace keys as well as the Tab and Enter keys.

The User field, beneath the Response edit box, shows the user who invoked the Alarm Manager. This read-only field shows which user responded to the alarm field.

The Alarm State region lets you change the alarm state to Acknowledged or Closed. Once an alarm is changed from the Open state, you cannot put it back in the Open state.

Control

The third region of the Alarm Manager dialog is the Control region, at the bottom of the dialog. In this region, there are several buttons used to control the Alarm Manager. They are:

- Freeze Frame
- Select Job
- New Alarm

Freeze Frame Button

As in the Job Activity Console, the Alarm Manager is automatically updated at regular intervals. The Freeze Frame button suspends this automatic refreshing of the screen, keeping the Alarm List static while you are working on it. The currently selected alarm will remain displayed until another alarm is selected, even when the dialog is being updated; as a result, responses can be entered uninterrupted. The Freeze Frame feature is particularly useful when scrolling through the Alarm List, since newly arriving alarms are added at the top of the list, and the list scrolls back to the top each time the display is refreshed.

Select Job Button

The Select Job button causes the job associated with the currently selected alarm (if there is one) to become the currently selected job on the Job Activity Console. This is useful when you want to review the details of the job for which the alarm was generated. In the example in Alarm Manager Dialog, the currently selected alarm is associated with the job “AlarmClock.” Therefore, pressing the Select Job button in the Alarm Manager will cause “AlarmClock” to become the currently selected job in the Job Activity Console. This operation will also update the Job Path (History) dialog (discussed in Job Path (History) Dialog in this chapter) in the process.

New Alarm Button

The New Alarm button serves the same purpose as the Alarm button on the Job Activity Console. This button turns red when a new alarm arrives, which is particularly useful when the Alarm Manager is not refreshing (when the Freeze Frame feature is in effect).

When you press either the New Alarm button on this dialog, or the Alarm button on the Job Activity Console, the Alarm List is updated and both of these buttons are reset to green, even if the Freeze Frame feature is on.

Even when this dialog is refreshing regularly, the New Alarm button can serve as an indicator that a new alarm has arrived.

When the alarm selection criteria is restrictive enough to filter out the new alarm, a warning dialog displays when the New Alarm button is pressed. This dialog offers to reset the selection criteria to the defaults; this will ensure that any new alarms that are generated will be displayed. Regardless of whether or not you accept this option, the New Alarm button’s color will be reset to green.

Registering Responses and Changing Alarm States

To register a response or change the state of an alarm in the database, you must explicitly save the alarm.

To save an alarm to the database:

Do one of the following:

- Click OK. This action dismisses the Alarm Manager.
or:
■ Click Apply. This action does not dismiss the Alarm Manager.

To dismiss the Alarm Manager without saving any changes

- Click Cancel.

Typically, you would use Apply to register changes, since the Alarm Manager would probably be running on a continual basis.

Alarm Selection Dialog

The Alarm Selection dialog lets you dynamically control which alarms are displayed. Alarms can be selected by type of alarm, state of alarm (Open, Acknowledged, or Closed), and by the date and time of the alarm's occurrence.

Note: Alarms that have been archived cannot be displayed.

To display the Alarm Selection dialog:

Select View, Select Alarm at the top of the Alarm Manager dialog. The Alarm Selection dialog appears:



The Alarm Selection dialog is divided into three regions, described in the next sections:

- Select by Type
- Select by State
- Select by Time

Select by Type

In the Select by Type region of the dialog, a list of all possible alarm types is displayed. From this list, you can select one, several, or all types of alarms. The default is All alarm types.

To choose a single alarm from the list:

Click the alarm's name.

To choose a range of alarms:

Click and hold the mouse button on the first alarm name, drag the cursor to the last alarm in the range, and release the mouse button.

To choose noncontiguous alarms:

Hold down the Control key and click the desired alarms. Hold down the Control key and click a selected alarm to deselect the alarm.

To choose all alarm types:

Select the All Types option, which overrides any more-specific settings.

Select by State

You can also select alarms by the state of the alarm. You can select any or all of the states by toggling on the appropriate buttons, or the All States toggle. The default is to display all Open and Acknowledged alarms.

Select by Time

By default, alarms are shown regardless of the time they were generated. You can choose to display only alarms that were generated during a specific date and time window. Fields are provided to specify a From Date, From Time, To Date, and To Time. You can specify dates without times. However, you cannot specify times without dates.

The current system date and time are automatically filled in for your convenience. You use a 24-hour format when specifying times.

To make the alarm selection take effect:

Do one of the following:

- Click OK. This sets your selections and dismisses the Alarm Selection dialog.
or:
■ Click Apply. This sets your selections without dismissing the dialog.

To dismiss the dialog without applying the selections:

- Click Cancel.

Customizing the Operator Console

You can use a set of X resources to customize the appearance and behavior of the Operator Console. In particular, these resources allow you to control:

- The interval between database reads to refresh the displays.
- The fonts and colors that are used in the Job Activity window as well as the Operator Console dialogs.
- How many characters of certain fields should be displayed.
- Whether or not the Operator Console starts up in Freeze Frame mode.
- The Operator Console GUI icon text and the Operator Console title bar text.
- User-configurable action buttons.

Descriptions of each of the resources which can be customized are given following. All of these can be set by modifying the X resource file Autocons. The X resources files reside in the local app-defaults directory, which varies across platforms. It is usually in /usr/lib/X11/app-defaults or /usr/openwin/lib/app-defaults. If you are not sure which directory these files are in, ask your system administrator.

Individual users may have their own copy of the X resources files in their \$HOME directory, which will take precedence over the app-defaults files.

For most operating systems, if you are exporting the display to another machine you must edit the appropriate files in the app-defaults directory on the local machine.

For Solaris, you must edit the files in both the /usr/lib/X11/app-defaults and /usr/openwin/lib/app-defaults directories. The files in /usr/lib/X11/app-defaults control the resources when you export the display.

We have listed the various resource names here as a reference only. For the default values, see the Autocons file. The provided resource file values work well for a majority of platforms.

Refresh Time Interval

The following resource controls the time interval after which the Operator Console will be refreshed (specified in milliseconds).

`*refreshTime:`

Alarm Poll Time Interval

The following resource controls the time interval after which the database will be searched for the arrival of new alarms, and the Alarm Manager will be refreshed (specified in milliseconds).

`*pollTime:`

Changing Fonts

The fonts used in the Operator Console fall into the following three categories:

- Those you can choose, independent of other fonts.
- Bold fonts, which must correlate with normal fonts.
- Normal fonts, which must correlate with, bold fonts.

Several of the lists in the Operator Console present information in columns, such as the Job List, which displays the job name, command, and so forth. In order to maintain the appropriate spacing for these columns, all characters in each field must be the same width. Therefore, fixed-format fonts must be specified for all column-oriented lists.

In order for the labels at the top of the columns to align with the columns themselves, those labels must also use fixed-format fonts, in the same size and style. We recommend that a bold-faced font be used for the labels, so that they are consistent with the other, non-list field labels (unless, of course, you have chosen a non-bold font for everything).

The font chosen for the column data should be a non-bold version of the same font used in the labels.

Note: The model X resources file provided specifies values that work well for each of the SunOS and AIX platforms.

Freeze Frame at Start Up

The following resource controls whether or not the Operator Console starts up in Freeze Frame mode, where “1” is true (the default) and “0” is false:

```
*FreezeAtStartup: 1
```

Font Selection Resources

The following resource specifications control font selection. They are completely independent of any other font settings:

```
*.fontList:  
*XmTextField.fontList:  
*XmText.fontList:
```

Label Font Resources

The following label resources should be the same, for consistency across the application. They should also be fixed and be of the same type as the list fonts given following, except that those probably shouldn’t be bold.

```
*jobListForm*jobListLabel.fontList:  
*dependHeadLabel*fontList:  
*atomicsForm*atomicsLabel.fontList:  
*alarmListForm*alarmListLabel*fontList:
```

List Font Resources

The following list resources should be the same, for consistency across the application. They should also be fixed and be of the same type as the label fonts shown previously, except that these probably shouldn’t be bold (typically called “medium”).

```
*jobListForm*fontList:  
*dependList*fontList:  
*atomicsForm*fontList:  
*alarmListForm*fontList:
```

The following resources do not necessarily relate to other fonts, but should match the non-bold font of the list fonts shown previously, for consistency across the application.

```
*pathList*fontList:  
*alarmCurrentText*fontList:
```

Object Color

The following resources affect the colors of the various objects within the Operator Console. We recommend that the default colors be used, since they match those used in the main AutoSys graphical user interface.

Currently Selected Job Name Field

The following resource controls the “currently selected” job name field, and should be a different color from the rest of the interface, for emphasis.

```
*workAreaForm*XmForm*currJobName.background:white
```

Background Color of Variable Fields

The following resources control the background color of the variable fields found in the interface, such as lists and text values, which change based on user interactions.

```
*workAreaForm*XmForm*XmText.background:  
*workAreaForm*XmForm*XmTextField.background:  
*workAreaForm*XmForm*XmList.background:  
*workAreaForm*XmForm*XmScrolledList.background:  
*workAreaForm*XmForm*XmScrolledText.background:  
*workAreaForm*XmPanedWindow*XmForm*XmScrolledWindow.background:  
*XmDialogShell*XmTextField.background:  
*XmDialogShell*XmText.background:  
*XmDialogShell*XmList.background:  
*selection_dialog*XmTextField.background:  
*selection_dialog*XmList.background:  
*depend_dialog*XmList.background:  
*path_dialog*XmList.background:  
*alarm_dialog*XmTextField.background:  
*alarm_dialog*XmText.background:  
*alarm_dialog*XmList.background:
```

Border Colors

The following resources set the decorative dark blue borders of the interface:

```
*workAreaForm.background:  
*workAreaForm*XmPanedWindow.background:  
*workAreaForm*controlForm*XmSeparator.background:
```

Primary Interface Color

The following resources set the primary interface color, medium gray:

```
*workAreaForm*XmForm*background:  
*menuBar*background:  
*exitDialog*background:  
*XmDialogShell*background:  
*selection_dialog*background:  
*depend_dialog*background:  
*path_dialog*background:  
*alarm_dialog*background:
```

Toggle Button Color

The following resource sets the toggle button color used to indicate that the toggle button is “on”:

```
*selectColor:
```

Job List Column Widths

The following resources set the lengths of the various fields (columns) in the Job List of the Job Activity Console. Specify a number of characters. (The lines beginning with # are comments.)

```
#job name:  
*nameLength:  
#job description:  
*descLength:  
#job status:  
*statusLength:  
#command to be executed, according to job  
#definition:  
*commandLength:  
#machine job is (or was) associated with, when it  
#runs (or ran):  
*machineLength:  
#spacing between columns in all lists:  
*fieldSpacing:
```

Atomic Condition Fields

The following resources apply to the atomic conditions in the Job Activity Console—the fields that describe a starting condition at its most basic level:

```
#length of condition (such as "SUCCESS(JOBA)":  
*atomicCondLength:  
#length of current state (eg, "SUCCESS"):  
*atomicStateLength:  
#length of true/false flag - whether the current  
#state satisfies the condition:  
*atomicTFLength:
```

Operator Console Size

The following two resources force the Console size to be a specified width and height. Keep in mind that this will override the resizing that occurs as a function of the chosen font.

```
*maxAppWidth:  
*maxAppHeight:
```

Default Report Type

The following resource specifies the default report type to be run for the currently selected job, at each Console refresh. Options are: None, Summary, and Detail.

```
*reportType:
```

Alarm List Column Width

The following resources determine the widths of the columns in the Alarm List. Specify a number of characters.

```
#Length of type of alarm (eg. JOBFAILURE):  
*alarmTypeLength:  
#Length of name of the job which generated the #alarm:  
*alarmJobNameLength:  
#Length of mm/dd/[yy]yy hh:mm time stamp of alarm  
#occurrence:  
*alarmTimeLength:  
#Length of alarm state - open, acknowledged, or  
#closed:  
*alarmStateLength:  
#Length of system-generated comment:  
*alarmCommentLength:
```

Operator Console Title Bar Text and Icon Text

The following resources specify the title bar text and the icon text. By default the title bar text is “Job Activity Console” and the icon text is “JAC.”

```
Autocons.shellTitle:  
Autocons.iconTitle:
```

Note: When changing icon text, be sure the length of the new text string does not exceed the recommended maximum length for icon title text for your windowing system. Some window managers can display long icon text strings, while others will truncate them. Ensure the text string you specify for your icons displays appropriately. Also, some window managers allow you to change the size of icons and icon text font.

User-Configurable Action Buttons

The last column of buttons in the Control area of the Job Activity Console are user configurable. You may associate any command with these buttons, and specify your own button labels. The following resource specifications specify labels and commands for the three configurable buttons.

```
Autocons*userButton1Label:  
Autocons*userButton1Command:  
Autocons*userButton2Label:  
Autocons*userButton2Command:  
Autocons*userButton3Label:  
Autocons*userButton3Command:
```

By default, these resources are commented out. Be sure to remove the preceding exclamation point (!) when you edit these specifications.

Button labels can be up to 20 characters, including spaces.

You must specify the full path to the command (environment variables are not resolved). After the path to the command, environment variables are allowed. \$JOB will take the value of the currently active job. We recommend that you run the command in the background by placing an ampersand (&) at the end of the command. Otherwise, the Operator Console will pause until the command has completed.

The following example shows how to use this resource to customize button1 to delete the job that is currently selected in the Job Activity Console, and button2 to run a command called printScreen:

```
Autocons*userButton1Label: Delete Job  
Autocons*userButton1Command: /usr/local/bin/deleteJob $JOB&  
Autocons*userButton2Label: SnapShot  
Autocons*userButton2Command: /usr/local/bin/printScreen&
```

Console Clock Perspective

The following resource specification sets the default console clock perspective.

```
! clock perspective (Server|Machine|Job)  
Autocons*clockPerspective: Job
```

Server is the server machine time. Machine is the time zone of the machine running the Operator Console. Job is the time zone specified in the job definition (using the timezone attribute). If no time zone is set for the job, then the server machine time is used.

This chapter describes how to define monitors and reports using essential and optional monitor/report attributes. It also explains how to define monitors and reports using both the GUI and JIL.

About Monitors and Browsers

Monitors provide you with a real time view of the system. Reports (or “Browsers”) provide you the ability to examine historical information about job executions using a variety of reporting views.

Monitors and reports are simply applications that run against the database. Since all information is in the database, monitors and reports that retrieve information from the database provide a complete picture of the state of the entire system. Monitors and reports can run with any database, and Dual Event servers can be in use. Also, monitors and reports can be run on any AutoSys client machine.

Monitors and reports enable you to filter and screen only the information you are interested in from a vast collection of data. That is, they are tools that can give you information meaningful to you.

Both monitors and reports filter events by the following:

- Type of event
- Job or groups of jobs

In addition, reports also filter by time. Monitors do not filter by time because they provide real time information.

Note: Monitors can provide a picture of the system's state in real-time. If the Event processor is down, monitors will not provide any information. On the other hand, reports provide a picture of the system's state from a "historical" view, not in real time.

Monitors

Monitors provide a real time view of the system. These are the two steps necessary to use a monitor:

- Defining which events to monitor.
- Actually running the monitor.

A running monitor is an application that polls the database for new events that meet the selection criteria. Monitors are strictly informational. They provide an up-to-the-minute window to AutoSys events as they occur. For box jobs, all job levels can be observed, if desired.

Reports

A report (or browser) is a query run against the database, based on the selection criteria defined for that report. Its primary function is to enable you to quickly get very specific information, such as the finish time of the database backup for the last two weeks or all jobs that have an alarm associated with them. In addition, all job levels in box jobs can be reported if desired.

Like monitors, a report definition is stored in the database, enabling reports to be run at any time, without redefining the criteria. Reports can only display events that are still in the database. Archived events are inaccessible and cannot be displayed.

Defining Monitors and Reports

There are a number of attributes that are used to define and describe monitors and reports. Using these attributes, you can specify everything from the name to be assigned to the new monitor or report; to the events you are interested in tracking.

You can specify monitors and reports using either of the following:

- Through the Graphical User Interface (GUI).
- By passing Job Information Language (JIL) statements to the jil command.

In either case, the monitor or report specification is stored in the database, and the attributes you specify are virtually the same. You define a new monitor or report by assigning it a name and specifying any number of attributes that further define its behavior.

Using the GUI

To define a monitor or report using the GUI:

1. Issue the autosc command, which displays the GUI Control Panel.
2. In the GUI Control Panel, click Monitor/Browser button, which displays the Monitor/Browser dialog (as shown in Defining Monitors and Reports using the GUI.)
3. Set the various attributes and their values using the fields and buttons in the Monitor/Browser dialog.

Using JIL

To define a monitor or report using JIL:

Issue the jil command, and pass it the insert_monbro subcommand followed by a set of attribute: value pairs.

Chapter Organization

In this chapter, monitor and report attributes fit into two categories (sections): essential and optional. Essential attributes must be specified in order for a definition to be valid, and optional attributes are not required.

For each attribute described in this chapter, the following is indicated:

- Its name
- Its JIL attribute keyword
- Its corresponding GUI object or “GUI Field Name”
- A description of its use

Essential Monitor/Report Attributes

Common Essential Attributes—General

The following attributes are required for both monitors and reports. Although defaults might be available, the attributes are still essential in that every monitor or report must include them, whether by default or by explicit specification.

Monitor/Report Name

JIL Keyword	insert_monbro: monbro_name
GUI Field Name	Name
Description	The monitor or report name is used to identify the monitor or report, and must be unique. A monitor and report cannot have the same name, but a monitor or report can have the same name as a job. A monitor or report name can be from 1-30 alphanumeric characters; embedded blanks and tabs are illegal.

Mode

JIL Keyword	mode
GUI Field Name	Mode
Description	The mode attribute indicates whether a monitor or report is being specified.

Common Essential Attributes—Events

The events specification determines which events are to be monitored or reported. Events are any changes in the state of jobs. They can also be generated occurrences, such as alarms. Or, they can be manually generated events, such as starting a job, placing a job on hold, or killing a job.

As described in the next section, monitors and reports can track events by filtering at several levels. Monitors and reports can also track events based on selected jobs. The events to be tracked are determined by the combination of the various event filters and the job filter, which are specified by using any of the essential attributes.

All Events

Keyword	JIL all_events
GUI Field Name	ALL EVENTS
Description	The all_events attribute specifies whether any event filtering is in effect. If it is set to “yes,” the other event filtering attributes are ignored, and all events, regardless of source, will be reported for the selected jobs. These events include job status events and alarms.

Note: If you wish to monitor all the events for all jobs, you should not run a monitor. Instead, you should display the event processor log time in real time, using the following command:

```
autosyslog -e
```

Running a monitor adds another connection to the database, and establishes another process that is continually polling the database. This process will have a significant impact on system performance. Moreover, the information logged by the event processor contains much more diagnostic information than that monitor does.

Alarms

JIL Keyword	alarm
GUI Field Name	Alarms
Description	This attribute specifies whether generated alarms should be tracked. Alarms can be tracked in addition to job status events (described following).

All Job Status Events

JIL Keyword	all_status
GUI Field Name	ALL Job Status Events
Description	This attribute specifies whether all job status events should be tracked. Job status events occur whenever a job's status changes. If this attribute is set to "yes," the individual job status events shown below and a few internal job status events, will be tracked.
Alarms can also be tracked in addition to job status events.	

Individual Job Status Events

The following table contains individual job status events:

JIL Keyword	Field Name
running	Running
success	Success
failure	Failure
terminated	Terminated
starting	Starting
restart	ReStarting

Job Filter

JIL Keyword	job_filter
GUI Field Name	Job Filter
Description	The job filter attribute determines which jobs are to be monitored or reported. Monitors and reports can track events based on selected jobs. The events to be tracked are determined by the combination of the various event filters and the job filter. The job filter can be set to one of three settings: track all jobs (no job filtering), track a single box with the jobs it contains, or track a single job. If either of the latter two is selected, the name of the job is required.

Essential Report Attributes

One of the attributes described following is required when defining reports, in addition to those essential attributes listed previously. Reports allow you to peruse all the events in the database. The time criteria listed here allow you to select events that occurred within a particular span of time. They are mutually exclusive.

Current Run Only

JIL Keyword	currun
GUI Field Name	Current Run Only
Description	This attribute specifies that only events in the current or most recent execution of the specified jobs will be reported. This feature is useful for getting a sense of what is happening right now. For example, you could select the job status event “restarting,” turn off job filtering, and set this attribute to “yes” to see all the jobs that have been automatically restarted in their current or latest run.

Events After a Certain Date/Time

JIL Keyword	after_time
GUI Field Name	Events After Date/Time
Description	This attribute specifies that only events occurring after a certain date and time for the specified jobs will be reported. This attribute cannot be used in a monitor definition because monitors only show events as they occur.

Optional Monitor/Report Attributes

Optional Monitor Attributes

The following monitor attributes are optional.

Sound

JIL Keyword sound

GUI Field Name Sound

Description This attribute specifies whether the sound facility should be used. If the workstation running the monitor has sound capabilities, Unicenter AutoSys JM will use them to announce the events as they occur. The announced message is pieced together from pre-recorded sound clips.

Note: We strongly recommend that you use the sound attribute for monitoring, especially alarms. It frees you from needing to look through output files to see if there are any problems.

For details on recording sound and a list of machines for which Unicenter AutoSys JM supports sound, see the record_sounds command in Chapter 1, Commands, in the *Unicenter AutoSys Job Management Reference Guide*.

Verification Required for Alarms

JIL Keyword	alarm_verif
GUI Field Name	Verification Required for Alarms
Description	This attribute specifies whether alarms need to be responded to by personnel before they will be turned off. This verification feature prompts the user, for their initials and a comment, in the running window. This information is timestamped and recorded in the database, along with the alarm event. This approach provides an account of the alarms that were responded to, and when they were responded to.
An important feature of this attribute is that if the response is not given within 20 seconds, the message is repeated. Therefore, if one momentarily steps out of the room and there is an alarm, it keeps writing to the window, and playing the sound clip (if specified) until someone responds.	

Optional Report Attributes

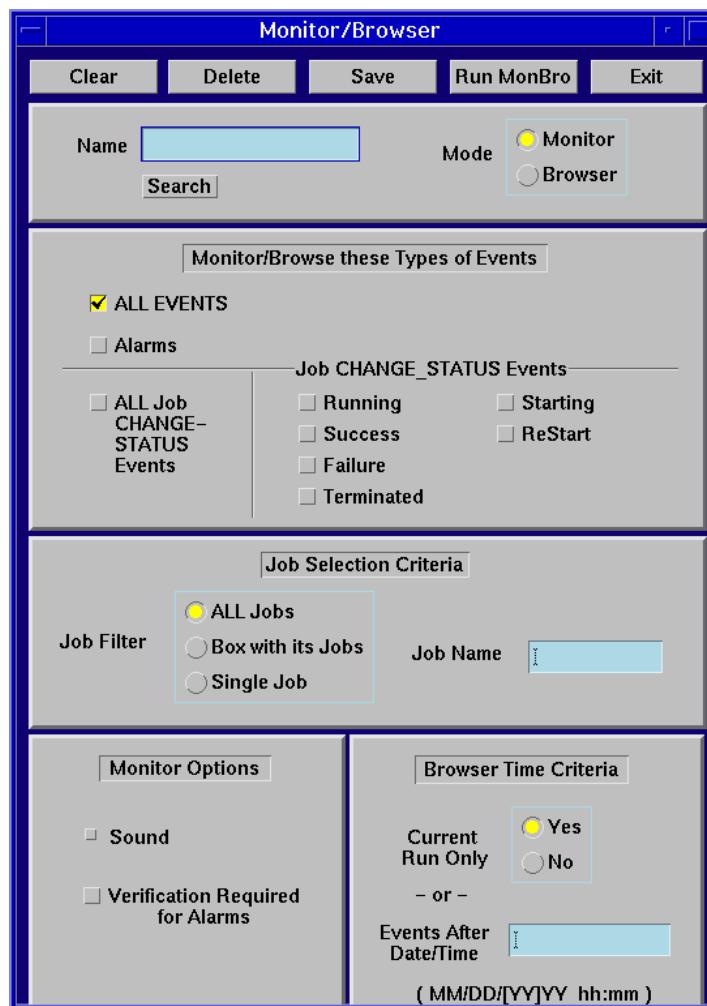
There are no optional report attributes.

Defining Monitors and Reports Using the GUI

This section describes the general use of the GUI Monitor/Browser dialog, which is used for defining monitors and reports.

The Monitor/Browser Dialog

The Monitor/Browser dialog contains fields representing all the information you need to define a monitor or report. When you click the Monitor/Browser button in the GUI Control Panel, the Monitor/Browser dialog appears:



The buttons at the top of the dialog are the dialog's control buttons. They perform the following actions:

Dialog Control Buttons	Actions
Clear	Clears the dialog without affecting the database. Use this button to clear all fields (in the dialog and memory), before you begin defining a new monitor or report.
Delete	Deletes the currently displayed monitor or report from the database.
Save	Stores the currently displayed monitor or report in the database, either modifying a pre-existing object, or creating a new one. It also clears the dialog in preparation for another monitor or report definition.
Run MonBro	Runs the monitor or report and displays output in an xterm window.
Exit	Closes the Monitor/Browser dialog and displays the GUI Control Panel. If Exit is pressed without pressing Save first, the recent changes are not saved. Exit only exits the dialog.

As indicated above, monitors and reports can be run from the Monitor/Browser dialog. A monitor must be saved in the database before it can be run, but reports do not need to be saved before they can be run. When a monitor or report is run from the dialog, a new terminal window appears to display the output. To close this window, press Control+C.

Defining an Example Monitor and Report

In this section, you'll define one monitor and one report. In the next section, describing the use of JIL, the examples from this section will be used to demonstrate the JIL statements you need to create the same definitions.

Defining a Monitor

First, you will define a monitor with the name "Regular." This monitor will monitor all alarms, plus job status events when a job changes state to running, success, failure, or terminated for the current job run.

To open the Monitor / Browser dialog:

In the GUI Control Panel, click the Monitor / Browser button, the Monitor / Browser dialog appears.

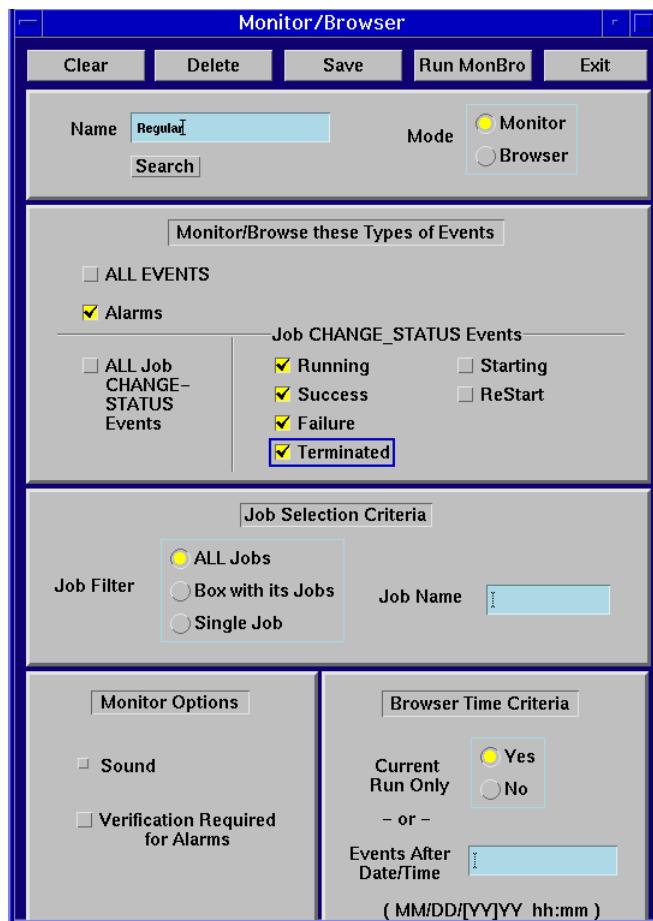
To define the example monitor:

1. In the Name field, enter the monitor's name:

Regular

2. In the Mode field, click the Monitor button.
3. In the Monitor / Browse these Types of Events region of the dialog, click Alarms.
4. In the Job CHANGE_STATUS Events region, click the Running, Success, Failure, and Terminated buttons.
5. In the Job Selection Criteria region of the dialog, click ALL Jobs.

Your entries in the Monitor/Browser dialog should appear as follows:



To save the monitor definition in the database:

At the top of the Monitor/Browser dialog, click Save.

To dismiss the Monitor/Browser dialog:

You can either dismiss the dialog using Exit, or you can leave it open to do the next exercise.

Note: If you want to run the monitor immediately, you must save it first, then click the Run MonBro button. When running a monitor or a report from the GUI, an xterm window is created to display the monitor or report output.

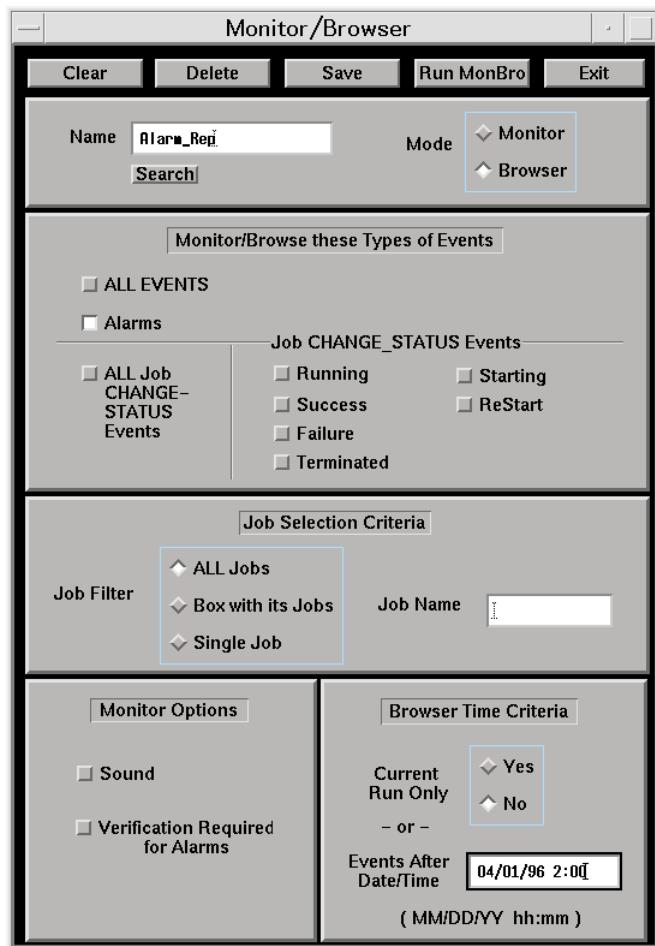
Defining a Report

Next, you will define a report with the name “Alarm_Report.” This report will report all alarms on any job, from December 22, 1997, at 12:00 to the present.

To define the example report:

1. Click Clear to clear the dialog and begin a new report.
2. In the Name field, enter the report’s name:
 Alarm_Report.
3. In the Mode field, click Browser.
4. In the Monitor/Browse these Types of Events region of the dialog, click the Alarms button only.
5. In the Job Selection Criteria region of the dialog, single-click ALL Jobs.
6. In the Browser Time Criteria region of the dialog, click the No button next to the words “Current Run Only,” and enter the date and time in the Events After Date/Time text field as follows (or you can enter a different, appropriate time):
 12/22/1997 12:00

Your entries in the Monitor/ Browser dialog should appear as follows:



To save the report definition in the database:

At the top of the Monitor/ Browser dialog, click Save.

To dismiss the Monitor/ Browser dialog:

You can either dismiss the dialog using the Exit button, or you can leave it open to create other definitions.

Defining Monitors and Reports using JIL

To define a monitor or report using JIL, use the same syntax as that used to specify a job. This is the syntax:

```
subcommand:monbro_name  
attribute_keyword:value
```

where

`monbro_name`

Is the monitor or report's name. It can be from 1-30 alphanumeric characters and is terminated with white space; embedded blanks and tabs are illegal.

The only difference between defining monitor/ reports and jobs is that different subcommands are used. For defining monitor or reports, these are the JIL subcommands:

- `insert_monbro`
- `update_monbro`
- `delete_monbro`

The following JIL script creates a monitor that will sound an alarm whenever a job finishes successfully:

```
insert_monbro: Job_Success  
mode: monitor /* "m" may be specified instead */  
sound: yes  
success: yes  
job_filter: all
```

Note: The JIL examples in the following sections include the monitor and report definitions presented in the GUI section previously shown.

For a list of machines for which Unicenter AutoSys JM supports sound, see the `record_sounds` command in the chapter “Commands” in the *Unicenter AutoSys Job Management Reference Guide*.

Defining Monitors

First, you will define a monitor with the name “Regular.” This monitor will monitor all alarms, plus job status events when a job changes state to running, success, failure, and terminated. It sounds an audible alarm whenever any of the events occur (for a list of machines which supports sound, see the record_sounds command in the chapter “Commands” in the *Unicenter AutoSys Job Management Reference Guide*.)

These are the JIL statements for this monitor definition:

```
/* Monitor for all ALARMS, and
 * Job EVENTS: RUNNING,SUCCESS,FAILURE & TERMINATED
 *
 * Sound is ON!
 */
insert_monbro: Regular
mode: m
sound: y
alarm: y
running: y
success: y
failure: y
terminated: y
```

The \$AUTOSYS/install/data/monbro.jil file contains the example JIL statements shown previously. It also contains the following JIL statements, which define a sample monitor:

```
/* Monitor for JUST ALARMS!
 * Verification Required is ON so someone must type in a response.
 */
insert_monbro: Alarm
mode: m
sound: y
alarm: y
alarm_verif: y
```

This set of statements defines a monitor that catches alarms, generates an audible alarm, and continually repeats the alarm until someone responds.

Defining a Report

Next, you will define a report with the name “Alarm_Rep.” This report will report all alarms on any job, from June 1, 1997, at 2:00 a.m., to the present.

```
insert_monbro: Alarm_Rep  
mode: b  
alarm: Y  
after_time: "06/01/1997 2:00"
```

Notice that quotes are required in the previous example, because the time contains a special character, the colon.

Note: Reports can only display events that are still in the database. Archived events are inaccessible and cannot be displayed.

Running a Monitor

To run a monitor:

Do one of the following:

- Enter the name in the Name field of the Monitor/Browser dialog, and click the Run MonBro button.

or:

- Run the monitor by executing the following command at the UNIX command line:

```
monbro -N monitor_name
```

Customizing the Monitor/Browser

There is a set of X resources you can use to customize the appearance and behavior of the Monitor/Browser GUI. In particular, these resources allow you to control:

- The time interval after which the Monitor/Browser GUI will drop the connection to the database.
- The Monitor/Browse GUI icon text and the Monitor/Browse title bar text.

Descriptions of the resources which can be customized are given following. All of these can be set by modifying the X resource file Autosc. The X resources files reside in the local app-defaults directory, which varies across platforms. It is usually in /usr/lib/X11/app-defaults or /usr/openwin/lib/app-defaults. If you are not sure which directory these files are in, ask your system administrator.

Individual users may have their own copy of the X resources files in their \$HOME directory, which will take precedence over the app-defaults files.

For most operating systems, if you are exporting the display to another machine you must edit the appropriate files in the app-defaults directory on the local machine.

For Solaris, you must edit the files in both the /usr/lib/X11/app-defaults and /usr/openwin/lib/app-defaults directories. The files in /usr/lib/X11/app-defaults control the resources when you export the display.

Database Connection Time-Out Interval

The following resource controls the time interval after which the Monitor/Browser GUI will drop the connection to the database:

```
! TimeOut to drop DB connections (in minutes)
*DDBDropTime: 400
```

If DDBDropTime is set to zero, the connection is dropped immediately after the database query has completed. A value greater than or equal to five means that the GUI will automatically drop all database connections if the database has not been accessed in the last DDBDropTime minutes. (Values of one to four are invalid). A new database connection will subsequently be established when required. If DDBDropTime is greater than 360, the connection to the database is maintained until you exit the GUI screens.

Monitor/Browser Title Bar Text and Icon Text

The following resources specify the title bar text and the icon text. By default the title bar text is “Monitor/Browser” and the icon text is “MonBro.”

```
Autosc.shellTitleMonBro:
Autosc.iconTitleMonBro:
```

Note: When changing icon text, be sure the length of the new text string does not exceed the recommended maximum length for icon title text for your windowing system. Some window managers can display long icon text strings, while others will truncate them. Ensure the text string you specify for your icons displays appropriately. Also, some window managers allow you to change the size of icons and icon text font.

This chapter describes the procedures for maintaining Unicenter AutoSys JM and the database.

Maintaining the Event Processor

The event processor is the engine of Unicenter AutoSys JM. When the event processor is not running, you cannot initiate new actions. You can stop the event processor, however, and the actions that have already started will run to completion.

Starting the Event Processor

To start the event processor:

1. On the server machine, enter the following command:

```
eventor
```

This command performs a number of consistency checks, and then starts the `event_demon` program.

You can start the shadow event processor at the same time as the primary event processor by specifying the `-M` option followed by the name of the machine on which you want the shadow event processor to run, like this:

```
eventor -M machine_name
```

The `eventor` script runs in the background automatically.

WARNING! *Do not try to start the event processor by invoking the `event_demon` binary at the command line. The `eventor` script is required to properly check and configure the environment for the event processor.*

eventor Command

The eventor command performs these tasks:

- Verifies that no other event processor is running on that machine.
- Invokes a restart procedure that looks for any events that are hung in the “processing” state. Because events are processed one at a time, there can only be one event hung in the processing state at a time. If there is an event in this state, it is queued again for processing. Normally, events will only be in this state if the event processor was stopped while it was processing an event.

Note: If the event being processed was a STARTJOB event, and the job it started is still alive, it will not be started again. Also, if an event is in the “processing” state for a long time, this does not imply that the job associated with that event is in some unusual state. It could be that the command is lengthy, or is waiting for machine resources.

- Invokes the chase command. Chase inspects the database to see what should be running, then checks each machine to verify that the appropriate jobs are running. If chase sees any anomalies, it sends an alarm, and, if the job definitions permit, it restarts any missing jobs.

The purpose of the chase feature is to verify the state and detect any problems upon startup.

- Starts the event_demon. Its output is redirected to the following file:

```
$AUTOUSER/out/event_demon.$AUTOSERV
```

When eventor starts, it automatically invokes the following command:

```
tail -f $AUTOUSER/out/event_demon.$AUTOSERV
```

This command displays the output of event_demon to the screen. To break out of the display, press Control+C. This terminates the tail command only, making the window available—the event_demon continues to run in the background.

Note: If you do not want eventor to run the tail command, you can use the -q option when you start the event processor. Several command line options allow you to start the event processor without some of the checks being performed; however, an experienced administrator should use these options only for special circumstances.

For more information about the eventor command, see its entry in the chapter “Commands” in the *Unicenter AutoSys Job Management Reference Guide*.

Starting in Global Auto Hold Mode

If you are restarting the event processor after a period of downtime, you can specify the -G option to start the event processor in Global Auto Hold mode. This prevents the system from being overloaded with job starts for the numerous jobs that were scheduled to run during the downtime.

When the event processor is in Global Auto Hold mode, it evaluates all jobs whose starting conditions have passed and are eligible to run. Instead of starting the jobs, however, the event processor puts the jobs ON_HOLD. It does this for all types of jobs (box, command, and file watcher). This allows you to decide which jobs should run and to start them selectively with the Force Start Job button in the Operator Console, or with the following command:

```
sendevent -E FORCE_STARTJOB
```

This FORCE_STARTJOB event is the only way to start a job put ON_HOLD with Global Auto Hold; it overrides the Global Auto Hold.

To turn off Global Auto Hold, you must shut down the event processor, and then start it again without the -G option.

You can start both the primary and shadow event processors with the -G option.

Monitoring the Event Processor

The output of the event processor (event_demon executable) is written to the following log file:

```
$AUTOUSER/out/event_demon.$AUTOSERV
```

This log file contains a record of all the actions taken by the event processor, including startup and shutdown information.

If the \$AUTOUSER directory is NFS mounted, you can view this output from any machine on the network.

To view the log file:

Do one of the following:

1. Type the following UNIX command:

```
tail -f $AUTOUSER/out/event_demon.$AUTOSERV
```

or:

2. Execute the autosyslog utility like:

```
autosyslog -e
```

When you execute this command, the last ten lines of the log file are displayed, and then all additions to the log are automatically displayed as they occur.

To terminate the autosyslog process:

Press Ctrl+C.

Note: We recommend that you use the autosyslog -e command to follow the behavior of the event processor. It displays the log file, which generates information on all event processor activity.

Event Processor Log File Size

At startup, the event processor checks the size of its log file, and if the file is 250 KB or more, the event processor deletes the file.

The event processor log has a file system threshold setting. The event processor shuts down if there is less than 8 KB of disk space available. However, if the amount of available disk space falls below that specified by the FileSystemThreshold parameter in the configuration file, the event processor issues warnings in the event processor log file.

For information on the FileSystemThreshold parameter, see Event Processor Log Disk Space in the chapter “Configuring.”

Stopping the Event Processor

It is safe to stop the event processor at any time, if it is stopped properly. Only the exec superuser can stop the event processor.

Note: Stopping the event processor does not affect jobs that are already running. They continue to run to completion, at which time the exit events are sent directly to the database. The effect of stopping the event processor is that actions triggered by incoming events sent, are not initiated until you start the event processor again.

To stop the event processor properly:

1. Log onto any configured machine as the exec superuser.
2. Issue the following command:

```
sendevent -E STOP_DEMON
```

This method allows the event processor to complete gracefully any processing it is performing. You can assign a high priority to the sendevent -E STOP_DEMON command by including the -P 1 argument.

When you issue the sendevent command, the STOP_DEMON event is sent to the database. The event processor then reads the STOP_DEMON event, goes into an orderly shutdown cycle, and exits. There might be a delay between when you send the STOP_DEMON event and when the event processor reads it and shuts down. If the event processor does not shut down immediately, do not send another STOP_DEMON event, because the event processor will process that event the next time it starts, and it will promptly shut down.

For more information about the sendevent command, see sendevent in the chapter “Commands” in the *Unicenter AutoSys Job Management Reference Guide*.

WARNING! *Do not attempt to stop the event processor (the event_demon process) by using the UNIX kill command. This method stops the event processor no matter what it is doing; it might be in the middle of processing an event. Also, if you are using dual-event servers and use these methods, the databases can lose synchronization.*

Shadow Event Processor Rollover

You can configure a shadow event processor to act as a backup event processor. The shadow event processor is normally idle; it listens for the periodic signals (every 90 seconds) from the primary event processor that indicates the primary processor is still functioning.

If the shadow event processor does not receive the signal, it does the following:

1. Checks the third machine for the .dibs file.
2. Does one of the following:
 - If it cannot connect to the third machine, the shadow event processor shuts down.
 - If it can connect but cannot locate the .dibs file, the shadow event processor creates the file, attempts to signal the primary event processor to stop, and takes over processing the events.
 - If it can connect and the .dibs file already exists, the shadow event processor shuts down.

Similarly, if the primary event processor cannot locate and signal the shadow event processor, the primary processor checks the third machine for the .dibs file, and follows the same procedure as the shadow event processor (as described previously).

If the primary event processor and an event server are on the same machine, the event processor failure could also mean an event server failure. In this situation, if dual event servers are configured, Unicenter AutoSys JM will roll over to the shadow event processor and to single-server mode.

Unicenter AutoSys JM uses the third machine and the existence of the .dibs file to resolve contentions and to eliminate the case where one processor takes over because its own network is down. However, the shadow event processor is not guaranteed to take over in 100% of the cases. For example, in the case of network problems, Unicenter AutoSys JM might not be able to determine which event processor is the “healthy” one, and it will shut down both processors.

Note: You can specify the shadow event processor and the third machine by modifying the tunable parameters in the configuration file. For information, see the chapter “Configuring,” in this guide.

Restoring the Primary Event Processor

To restore the Primary and the Shadow event processor

1. Stop the shadow event processor by logging on as the exec superuser, and issuing the following command:

```
sendevent -E STOP_DEMON
```

2. Issue the following command on the primary event processor machine:

```
eventor -M shadow_machine
```

This command starts the primary and shadow event processors at the same time.

Note: If you attempt to start the primary and shadow event processors without having a third machine specified in the configuration file, the shadow event processor will not start.

Running in Test Mode

If you want to check your configuration, you can run in test mode. This process is helpful for troubleshooting problems. Running the event processor in test mode allows you to test the set up as well as the execution of logic by the `jil` command, without having to run the defined jobs. A simple job is run in place of the defined job.

When running in test mode, you can determine the following:

- If the event processor and the remote agents are installed and configured properly. Running in test mode uses the same mechanisms of starting jobs and sending events in its normal mode.
- If the conditional logic for jobs, including nested boxes, is functioning correctly.

You can run the event processor at two levels of test mode. You do this by setting the `$AUTOTESTMODE` environment variable before starting the event processor. The levels of test mode are determined by the value of the `$AUTOTESTMODE` variable. These are the values, which are discussed in the following sections:

- `$AUTOTESTMODE = 1`
- `$AUTOTESTMODE = 2`

WARNING: *The event processor cannot run partially in test mode; Unicenter AutoSys JM does not provide a test mode for the database. Therefore, you should exercise extreme caution when you run in test mode on a live production system.*

\$AUTOTESTMODE = 1

At the first level of test mode, each job that you specify runs with the following test mode variations:

- The command /bin/date is executed on the remote machine instead of the command specified in the job definition.
- Standard output and standard errors for the command are redirected to the /tmp/autotest.\$AUTO_JOB_NAME file, where \$AUTO_JOB_NAME is the job name as defined to AutoSys.
- If the job being run in test mode is a file watcher job, it is not disabled; it runs as it would in real mode.

The following functions are disabled:

- Minimum and Maximum Run Alarms
- Sourcing a user-specified .profile file
- All resource checks

\$AUTOTESTMODE = 2

The second level of test mode runs with the same behaviors as the first level with the addition of the following procedures:

- Resource checks are performed.
- A user defined .profile file is sourced.
- Output from the /bin/date command goes to the user defined standard output and standard error files, if they are defined; otherwise, output goes to the file named /tmp/autotest.\$AUTO_JOB_NAME.

Maintenance Commands

The maintenance commands described following are located in the \$AUTOSYS/bin directory. You can execute these commands from the UNIX operating system prompt or as an job.

chase

The chase command verifies that the expected jobs are running. It goes to every machine that should be running a job and verifies that the following are true:

- The remote agent is running.
- The job is running.

Errors detected by chase are sent to standard output. The options used with chase further determine what actions are taken when error conditions are detected. chase can send alarms to alert you to the problems it finds (by using the -A option). In addition, it can automatically restart jobs that are “missing in action” and that are defined for restart (by using the -E option). For more information about the chase command, see the chapter “Commands” in the *Unicenter AutoSys Job Management Reference Guide*.

Note: There is no way for chase to tell if a machine is down; therefore, it cannot tell if jobs on that machine are running, or if the network connection to the machine is down. If you run chase with the -E option, jobs that have already run, or are running on the machine with the failed network connection might be restarted if the network connection is established again.

clean_files

The clean_files command deletes old remote agent log files. It performs this task by searching the database for all machines that have had jobs started on them, and then sending a command on that machine to purge all remaining log files from the machine's Remote Agent Log directory (specified by AutoRemoteDir in the configuration file). To remove only the log files older than a specific number of days, use the following command:

```
clean_files -d days
```

where:

days

Specifies that files older than this number of days should be deleted.

For more information about the clean_files command, see the chapter "Commands" in the *Unicenter AutoSys Job Management Reference Guide*.

Backing Up Definitions

We recommend that you back up the following definitions periodically to ensure that you have files to restore from in case of a system failure:

- calendar definitions
- job definitions
- machine definitions
- monitor and browser definitions
- global variables

For information about restoring the backed up definitions, see Restoring Definitions in this chapter.

We recommend that you keep a copy of your license keys in case you need to reinstall them.

To back up definitions:

1. To save your calendar definitions:
 - a. Open a Calendar Definition window.
 - b. Choose File, Export.

The Export File Name dialog is displayed.

- c. In the Export File Name dialog, select a directory that is outside of the directory structure and select or enter a file name.
- d. Click OK.

Note: The calendar definitions are saved as text.

2. To save your job definitions, from a UNIX command prompt, execute the following command:

`autorep -J ALL -q > /directory/autosys.jil`

where:

directory

Is a directory outside of the directory structure. We recommend that you save to the same directory where you saved your calendar definitions.

This command saves your job definitions to a file named autosys.jil.

3. To append your machine definitions to the same file that contains your job definitions, from the UNIX command prompt, execute the following command:

```
autorep -M ALL -q >> /directory/autosys.jil
```

where:

directory

Is the same directory where you saved your job definitions, a directory outside of the directory structure.

Note: To append definitions to an existing file, you enter `>>` instead of `>`. We recommend that you append your job, machine, and monitor and browser definitions to the same file. Then you have only one file to restore following a system failure.

4. To append your monitor and browser definitions to the same file that contains your job and machine definitions, from the UNIX command prompt, execute the following command:

```
monbro -N ALL -q >> /directory/autosys.jil
```

where:

directory

Is the same directory where you saved your job definitions, a directory outside of the AutoSys directory structure.

5. To save your global variables to their own file, from the UNIX command prompt, execute the following command:

```
autorep -G ALL > /directory/globals.jil
```

where:

directory

Is a directory outside of the directory structure. We recommend that you save to the same directory where you saved your other definitions.

This command saves your global variables to a file named `globals.jil`. This file is simply a record of what you must redefine following a system failure.

Note: You can create a job that runs periodically to back up your definitions automatically.

6. To save your license keys, run the gatekeeper command to print your current license keys to a file.

Restoring Definitions

The procedure in this section assumes that you backed up your definitions by following the procedure in Backing Up Definitions in this chapter.

To restore definitions:

1. To restore your calendar definitions:
 - a. Open a Calendar Definition window.
 - b. Choose File, Import.

The Import File Name dialog is displayed.

 - c. In the Import File Name dialog, select the directory and file name of the text file that contains your calendar definitions.
 - d. Click OK.
2. To restore your job, machine, and monitor and browser definitions, from a UNIX command prompt, execute the following command:

`jil < /directory/autosys.jil`

where:

directory

Is the directory where you saved your definitions.

3. Restore your global variables, reference your backup file and redefine any global variables.

Database Overview

All information is stored in a relational database known as the event server or database. For this database, you can use either a Sybase or an Oracle database.

An event server contains all of the information about a particular instance. The event processor reads from the event server to determine what actions to take. The remote agents send starting, running, and completion information about jobs to the event server.

Due to the critical nature of the information stored in the database, you can configure to run with dual-event servers (two databases). Dual databases provide redundancy in the event of an event server crash.

Note: While Unicenter JM uses the database solely as an SQL engine, it does use Sybase Open Client C Library communications protocol and Oracle SQL*Net V2 to send events around the system.

Event Server Overview

An event server can be associated with only one instance and one running event processor.

An event server contains the following objects:

- Job definitions
- Events
- Monitor and report (browser) definitions
- Calendar information
- Machine definitions

For a list of the database tables and views as well as the event and alarm codes used in the database, see the chapter “Database Tables and Codes” in the *Unicenter AutoSys Job Management Reference Guide*.

Using Dual Event Server Mode

When you configure with dual-event servers, all of the data is duplicated on two event servers. In dual-server mode both servers are peers, and the event processor is responsible for keeping the databases synchronized. The event processor continually reads from both databases as it processes events.

For information about installing a second event server, see *Installing Dual-Event Servers* in the chapter “Advanced Configurations” in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

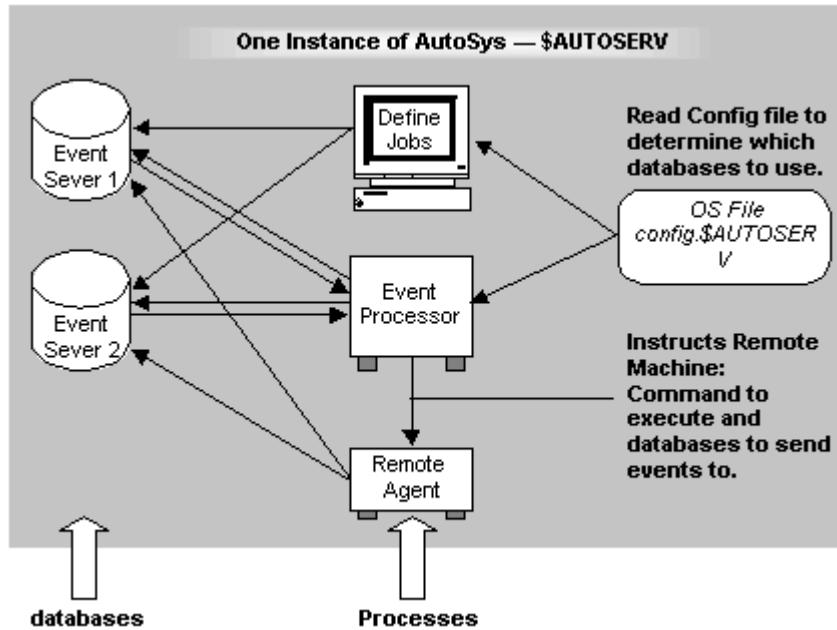
Database Storage Requirements

The standard sizes for databases are 64 MB (Sybase) and 100 MB (Oracle). The standard sizes for databases are the recommended sizes. If your job load is large, you should create a larger database. The size requirements for your database depend on the following:

- The number of jobs you define.
- How many of the jobs have dependencies.
- How often the jobs are run.
- How often the database is cleaned. (Every time a job runs, it generates at least three events and an entry in the job_runs table.)

Database Architecture

The following figure shows the layout of databases in an environment, and it will help you understand configuration options. It depicts how Unicenter AutoSys JM determines which database to use, and how the three primary components (the event processor, the database, and the remote agent) interact.



The previous figure shows one instance that is configured with dual-event servers, which are used only by this one instance. Both the event processor and the remote agent ensure that events are written to the appropriate databases.

The controlling variable in the architecture is the environment variable named AUTOSERV. This variable is the instance name that indicates, among other things, the name of the configuration file to be used. The configuration file name is determined by expanding the environment variable in the following way:

\$AUTouser/config.\$AUTOSERV

This configuration file contains information about which database to use, and in what capacity. All commands access this file, unless they are overridden at the command line with an argument that states an instance name.

For information on configuring instances, see the chapter “Configuring,” in this guide.

Note: Different instances can start jobs on the same machine. The remote agent receives instructions from the event processor at runtime, and the remote agent can send events to the necessary databases.

General Database Maintenance

You can configure Unicenter AutoSys JM to maintain itself as it runs. In fact, there are defaults in place that provide frequent, automatic maintenance. You can customize the configuration file. In addition to this regularly scheduled maintenance, you can issue general and database maintenance commands from the command line.

Periodic maintenance is essential to keeping Unicenter AutoSys JM working correctly. Several events are generated for each run of each job. If these events are not “pruned” from the database periodically, the database will eventually fill up, bringing Unicenter AutoSys JM and its jobs to a halt. Therefore, we recommend that you use the suggested periodic maintenance practices described in this section.

Daily Database Maintenance

Once a day, the event processor performs internal database maintenance. During this time, it does not process any events; it waits for the maintenance activities to complete before resuming normal operations. By default, this maintenance cycle starts at 3:30 a.m. If it is necessary to change the start time, reset it to a time of minimal activity.

The DBMaintTime parameter in the configuration file allows you to specify the time for daily database maintenance. Use a 24-hour format for the entry. For information on the database maintenance parameter, see the Configuration File Parameters section in the chapter “Configuring,” in this guide.

DBMaint Script

By default, Unicenter AutoSys JM executes the \$AUTOSYS/bin/DBMaint script during its daily maintenance cycle. This script runs the dbstatistics and archive_events commands.

DBMaint runs the dbstatistics command to perform the following tasks:

- Update statistics in the database for optimal performance. For Sybase database, it updates statistics for the event, job, job_status, and job_cond tables. For Oracle, it computes statistics for all of the tables.
- Run the dbspace command to check the available space in the database. If the amount of free space is insufficient, it issues warning messages and generates a DB_PROBLEM alarm.

Note: If you use an Oracle database, running DBMaint may report that your database is close to full when this is not the case. This can occur because DBMaint calculates how much space is not allocated for extents. The extents may be nearly empty, but DBMaint reports the whole extent as used space.

- Calculate and update the average job run statistics in the avg_job_run table. When dbstatistics is run, old data is overwritten with the new data.

DBMaint runs the archive_events command to remove old information from the various database tables. Specifically, archive_events removes the following:

- Events and any alarms associated with them from the event table
- Job run information from the job_runs table.
- autotrack log information from the audit_info and audit_msg tables
- ServerVision audit information from the svarchive_tbl table

The output from DBMaint, \$AUTOUSER/out/DBMaint.out, tells you how much space is left in your database, so that you can check (and monitor) if the event tables are filling up. This is a good way to calculate how many events in a single day can be maintained safely in the database before they should be archived.

For more information on the dbstatistics and archive_events commands, see their entries in the chapter “Commands” in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*. For a list of the database tables and views, as well as the event and alarm codes used in the tables, see the chapter “Database Tables and Codes” in the *Unicenter AutoSys Job Management for Windows and UNIX Reference Guide*.

WARNING! *If you are archiving large event tables, your SQL connection might time out, causing the DBMaint script to core dump. If this occurs, change the -t argument of the archive_events command to a higher value.*

Modifying the DBMaint Script

You can modify the \$AUTOSYS/bin/DBMaint script. For example, you might want to modify the script to perform database backups also. For information on backing up bundled Sybase, see Bundled Sybase Backup and Recovery in this chapter.

When you modify the script, copy it first, and then add your enhancements to the copied version. If you modify the script, you should keep a backup copy of it; then, when you upgrade, you will not lose your changes. You can use your enhanced script to modify the newly installed script.

The script name is specified by the DBMaintCmd parameter in the configuration file.

Data Locking

Depending on your environment, you may see many Sybase deadlock messages in the event_processor log. While this has no adverse effect on the integrity of your system, it can affect performance. To reduce the number of deadlocks, you can turn on row-level locking.

To turn on row-level locking for the entire database, type the following at an xql prompt:

```
sp_configure "lock scheme", 0, datarows;
```

Or, you can lock tables individually, with the following command from an xql prompt:

```
alter table <table> lock datarows;
```

Where *<table>* indicates the name of the table on which to configure row-level locking. Based on usage, we recommend the following tables:

Tables

event	proc_event	job
job2	job_cond	job_status
job_runs	last_Eoid_counter	next_oid

Event Server Rollover Recovery

When Unicenter AutoSys JM is running in dual-server mode and detects an unrecoverable error condition on one of the event servers, it will automatically rollover to single server mode.

An unrecoverable error is defined as one of the following:

- The connection to the database is lost, and, after the configured number of attempts to remedy this situation have transpired, the database still remains unconnected.
- A database has had an unrecoverable error (for example, corrupt database or media failure).

Upon event server rollover, the event processor edits the \$AUTOUSER/config.\$AUTOSERV configuration file on only the event processor machines. The event processor comments out the database that has been taken off-line and marks the remaining database as being in single server mode. The event processor makes these changes so that utilities attempting to access the database will write to or read from only the running event server.

Note: On an event server rollover, the configuration file is edited on the event processor machines only; configuration files on client machines are not modified.

Event Server Crash

If one event server crashes, Unicenter AutoSys JM will automatically roll over to the “healthy” server and continue running in single-server mode. After you recover the crashed server, you can reconfigure to run in dual-server mode again.

WARNING: *Do not, under any circumstances, restart the down event server and run in dual server mode. Before starting the down server, you must make sure that the two event servers are synchronized, following the instructions in Synchronizing the Event Servers.*

Synchronizing the Event Servers

If you have gone into single-server mode due to problems, and now want to go back to running dual-event servers, you must synchronize the databases.

Note: Before doing this, you must have already installed and configured your system for dual-event servers, as described in *Installing Dual Event servers* section in the chapter “Advanced Configurations” in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

To synchronize the databases:

1. Log on to a database machine as the exec superuser.
2. Make sure that no one is executing jil or using the GUI dialogs to change job definitions.
3. Stop the event processor by entering the following command:

```
sendevent -E STOP_DEMON
```

After the event processor is stopped, no additional jobs will be started.

4. Synchronize the databases using the autobcp script. This is described in detail in *Run the autobcp Script* in the chapter “Advanced Configurations” of the *Unicenter AutoSys Job Management for UNIX Installation Guide*.
5. Edit the \$AUTOUSER/config.\$AUTOSERV configuration file on the server machine. In the configuration file, remove the rollover comment from the EventServer line that defines the sever that went offline. The event processor commented out this line during the rollover to single-server mode.
6. Start the event processor, using the eventor command on the event processor machine, like:

```
eventor
```

Or, if you are running a shadow event processor, start the event processors like:

```
eventor -M shadow_machine
```

The event processor should print a message indicating that it is in dual-server mode.

Note: If Unicenter AutoSys JM is configured to run in dual-server mode, the event processor will not start unless both databases are available.

Improving Database Performance

This section contains information about improving your database performance.

Improving Sybase Database Performance

These are ways to improve your Sybase database performance:

- Increase the amount of memory that Sybase can use.

WARNING! Only the database administrator should increase Sybase memory.

- Upgrade your processor, memory, or hard disks.
- Install the Sybase server and the event processor on a dedicated machine or machines. Do not share machine resources with other processes.
- Use the Sybase server for Unicenter AutoSys JM only.
- Tune the kernel for optimal Sybase performance. For information on how to do this, see your Sybase and operating system documentation.
- For unbundled Sybase only, put your data on a raw partition. This improves access time.

WARNING! Only the database administrator should put your data on a raw partition.

Improving Oracle Database Performance

Tuning Oracle to improve its performance is complicated. The following are some suggestions for the database administrator:

- Tune the shared pool size. Make changes to the shared pool size by altering the init.ora value of SHARED_POOL_SIZE.

To determine if you need to increase the shared pool size, enter the following query in SQL*Plus:

```
select sum(pins) "Executions,"  
sum(reloads) "Cache Misses while Executing,"  
((sum(reloads)/sum(pins))*100) "Ratio of Misses"  
from v$librarycache;
```

The ratio of misses should be less than 1%. (The ratio of misses number is displayed as a percentage.) If it is higher than 1%, you should increase the value of SHARED_POOL_SIZE incrementally until the value of executions approaches zero.

- Tune the buffer cache. Make changes to the buffer cache by altering the init.ora value of DB_BLOCK_BUFFERS.

To determine if you need to allocate more memory, enter the following query as the "sys" user in SQL*Plus:

```
select name, value  
from v$sysstat  
where name in  
('db block gets', 'consistent gets', 'physical reads');
```

Monitor the statistics from the query while running. Calculate the hit ratio for the buffer cache by using this formula:

$$\text{hit ratio} = 1 - (\text{physical reads}/(\text{db block gets} + \text{consistent gets}))$$

The closer the hit ratio approaches 1.00, the better your system performs. If you have free memory and the hit ratio is below .95, increase the value of DB_BLOCK_BUFFERS. Make sure you have at least 5% free memory.

- Maximize disk I/O by separating the data files. If you have disk contention, place the autodata and autoindexes data files on separate disk drives, and if possible, different drive controllers.
- Tune the sort area. A sort area in memory sorts records before they are written to disk. Increasing the size of the sort area by increasing the init.ora value of SORT_AREA_SIZE improves sort efficiency.

To determine if sorting is affecting the performance of your system, monitor the sorting disk activity in your system by entering the following query in SQL*Plus:

```
select name, value from v$sysstat  
where name in ('sorts (memory)', 'sorts (disk)');
```

If disk sorts are greater than 1% of memory sorts, then increase the value of SORT_AREA_SIZE.

Maintaining Bundled Sybase SQL Servers

Because you can purchase Unicenter AutoSys JM with a bundled Sybase database, this section is specific to Sybase event server maintenance. It contains information to help you query the database as well as to perform basic maintenance procedures on Sybase SQL Server databases.

Note: The following sections are specifically for bundled Sybase users. If you are using an existing Sybase or Oracle database, consult your database administrator for details on starting, stopping, and configuring your database.

Sybase Architecture

The Sybase database is based on a client/server architecture, with the communications between clients and server built into the product. The server portion is called the Sybase SQL Server. This server is a multi-threaded, single process that runs on one machine. It listens on a specific port for a request from a client, fulfills that request, and then returns the information to the client.

The client communicates with the server using a C library known as Open Client, or the DB Library. This library handles the communications between the client application and the Sybase SQL Server as well as sending requests and parsing results for the use of the application.

This means that all commands and processes, including the event processor, the remote agent, and monitors, are DB Library applications that connect to the databases.

Because all commands are merely Sybase clients, you can execute those commands from any machine that has access to the event server and is a licensed client.

Note: The DB Library allows a client to maintain multiple connections to the same server or multiple servers. It is through this mechanism that the dual-event servers are maintained.

Sybase Environment

If you are using a Sybase, the following environment variables are used:

DSQUERY—Defines the name of the Sybase data server.

SYBASE—Specifies the complete path to the Sybase software directory.

The Sybase software directory contains the Sybase configuration file, which on UNIX is the interfaces file and on Windows is the SQL.INI file. AutoSys uses the Sybase configuration file to look up database information. It is the means by which the network is navigated to find the Sybase data server.

Default Sybase Users

To perform many administrative tasks on your bundled Sybase database, you use the xql utility, which is an supplied utility that can access the Sybase data server from any properly configured client machine. In most cases, you must log onto Sybase as “sa” using the “sysadmin” password, or the appropriate password. You can log onto Sybase within a command line, or you can log onto Sybase as “sa,” then operate interactively at the xql prompt.

You do most of the Sybase administration through system-stored procedures. All system-stored procedures start with the sp_ prefix.

For more information on using xql, see its entry in the chapter “Commands” in the *Unicenter AutoSys Job Management Reference Guide*.

Database Users

When using the bundled Sybase version, there are two users defined by default in the database: the system administrator and the user. Each of these users has a default user name and password.

User	User Name	Default Password
System Administrator	sa	sysadmin
User	autosys	autosys

For information on changing the “sa” password, see the next section, Changing the System Administrator Password. For information on changing the “autosys” password, see autosys_secure in the chapter “Commands” in the *Unicenter AutoSys Job Management Reference Guide*.

Changing the System Administrator Password

You can change the system administrator password from its initial value of “sysadmin” to a new value.

To change the system administrator password

1. Execute the following xql command:

```
xql -Usa -Psysadmin
```

The following prompt appears:

```
xql>>[AUTOSYSDB] [master] 1>
```

Note: If you are not using the default settings, the name of your data server is substituted for AUTOSYSDB.

2. Enter the following command sequence (replacing *newPassword* with the new password):

```
xql>>[AUTOSYSDB] [master] 1> sp_password  
xql>>[AUTOSYSDB] [master] 2> sysadmin, newPassword;
```

Notice the semi-colon at the end of the command line. It is an end-of-statement delimiter, replacing the Sybase “go.”

WARNING! After you change the “sa” password, the old password is unrecoverable. Make sure that the new password is recorded; otherwise, the entire database will have to be re-installed.

Starting Sybase

Starting Sybase involves executing the command that will bring up the database on the machine where Unicenter AutoSys JM has been installed. The following example is for Sybase-bundled versions. If there is a pre-existing data server at your site, consult your local database administrator about how to start the database.

To start the bundled Sybase:

1. As the user who installed Unicenter AutoSys JM, log onto the machine where the data server is to run. This is usually done on the machine where Unicenter AutoSys JM has been installed, but not always. If the machine has a windowing system running, use the console window. (You do this so all database messages will be displayed in that window.)
2. Source the \$AUTOUSER/autosys.env file. This sets the \$SYBASE environment variable and the alias start_autodbd.
3. Enter one of the following commands:
 - `start_autodbd`
 - `$SYBASE/install/RUN_AUTOSYSDB`

Both of these commands start the server in the background, then output startup messages.

Note: The most common problem encountered when starting up the database is the permission assignments on the user who owns the files used. If you experience some difficulty, ensure that you have the proper permissions to execute the operation. If you do have the necessary permissions and the database does not start, contact Computer Associates Technical Support.

4. To verify that the database is up and accessible, enter the following xql command:

```
xql -Uautosys -Pautosys -c "select getdate()"
```

If this command returns the date, then the database is up.

Stopping Sybase

You must shut down Sybase before shutting down or rebooting the machine. Before you shut down Sybase, however, you should shut down the event processor.

To stop Sybase:

1. If the event processor is running, stop it. To do this, log on as the exec superuser and enter the following command:

```
sendevent -E STOP_DEMON
```

Note: If you are not sure whether the event processor is running, do not send the STOP_DEMON event. If the event processor is not running and you send the event, it will be queued and sent when the event processor is started again. Before you attempt to stop the event processor, ensure that it is running by using the chk_auto_up command.

2. Stop the Sybase service by entering the following command (the sa_password is initially installed as "sysadmin"):

```
xql -Usa -Psa_password -c "shutdown"
```

This command allows any database processes to complete, and then shuts the database down. If you must shut down the database immediately, use this command:

```
xql -Usa -Psa_password -c "shutdown with no_wait"
```

For information on changing the “sa” password, see [Changing the System Administrator Password](#) in this chapter.

Accessing Sybase

Unicenter AutoSys JM comes with a utility named xql, which resides in the \$AUTOSYS/bin directory. Use this utility for interactive database queries and for shell-level database access. For a detailed description of how to use xql, see its entry in the chapter “Commands” in the *Unicenter AutoSys Job Management Reference Guide*.

Some examples of using xql in a shell script can be found in the following scripts:

- \$AUTOSYS/dbobj/create_table
- \$AUTOSYS/bin/chk_auto_up

Note: The xql utility functions only with the Sybase database. If you use an Oracle database, use sqlplus instead.

Identifying Processes Connected to the Database

These are reasons you might want to identify the processes that are connected to the database:

- Before you change the “autosys” user password or shut down Unicenter AutoSys JM , you should ensure that no processes are connected to the database. You can identify the active processes, then shut them down.
- Many GUI processes connected to the database can slow it down. You can see how many and what type of processes are connected to the database, and then ask users to shut down the GUIs they are not currently using.

To see what processes are connected to the database and their status:

1. At the xql prompt, enter the following:

```
xql>>[AUTOSYSDB] [master] 1> select program_name, hostname,  
xql>>[AUTOSYSDB] [master] 2> hostprocess,  
xql>>[AUTOSYSDB] [master] 3> status from sysprocesses;
```

The list of processes connected to the database is displayed, like:

program_name	hostname	hostprocess	status
xql	Joe	14050	running
			sleeping
event_demon	Michelle	13448	recv sleep
jil	Erik	12272	recv sleep

status 0, rows affected 8

In the example list of processes, xql is the process running your query to display the processes, and event_demon is the event processor. The processes without names are internal Sybase processes, which you can ignore.

The following are the most common processes you will see (there are others): autocal, event_demon, sendevent, autocons, hostscape, timescape, autosc, jil, xql, auto_remote, and jobscape.

Displaying the Database Date and Time

Jobs are scheduled and run based on the date and time for the machine on which the database is running. If your jobs do not run when you expect them to, you can check the database clock.

To display the database date and time, at the xql prompt, enter the following:

```
xql>>[AUTOSYSDB] [master] 1> select getdate();
```

Bundled Sybase Backup and Recovery

You must back up the database periodically, so that you can recover it in the event of catastrophic system or media failure. You should decide on a routine for backing up the database so that you will have no trouble recovering a lost database, using the backup database dump.

This section describes how to create a backup server and dump the database to a file, which you can then back up to tape.

If you have not yet defined a dump device to Sybase, you must define a disk or tape dump device, as described in Defining a Disk Dump Device in this chapter. Then follow the steps in Dumping the Database in this chapter.

Defining a Dump Device

A dump device can be either a disk or a tape.

Defining a Disk Dump Device	To define a disk as a dump device
	<ol style="list-style-type: none">1. Display the xql prompt, by entering the following command (assuming that the "sa" password is "sysadmin"): <pre>xql -Usa -Psysadmin</pre>2. At the xql prompt, enter the following: <pre>xql>sp_addumpdevice "disk," <i>dump_device</i>, "path name," 2;</pre>where: <i>dump_device</i> Is an arbitrary name; this name will be used for subsequent database dumps and loads. "<i>path name</i>" Is the full path name of the file where the database dump will be written. Ensure that this file can be created to the appropriate size for your database, and that it can be mounted on the machine that hosts the second event server.
	For example, the following command will define a disk dump device named "autodump," and the database will be dumped to a file named /backup/autodumpdata (for this example to work, the backup directory must already exist). <pre>xql>sp_addumpdevice "disk," autodump, "/backup/autodumpdata," 2;</pre>
Defining a Tape Dump Device	To define a tape device as a dump device:
	<ol style="list-style-type: none">1. Display the xql prompt, by entering the following command (assuming that the "sa" password is "sysadmin"): <pre>xql -Usa -Psysadmin</pre>

2. At the xql prompt, enter the following:

```
xql>sp_addumpdevice "tape," dump_device, "physical_device," 3, skip, size;  
where:
```

dump_device Is an arbitrary name; this name will be used for subsequent database dumps and loads.

"physical_device" Is the device name of the actual tape device on your machine.

size Is the capacity of the tape device (in MB).

For example, the following command will define a tape dump device named "autodumptape," and the database will be dumped to a tape with a 40 MB capacity loaded in the device named /dev/rmt8:

```
xql>sp_addumpdevice "tape," autodumptape, "/dev/rmt8," 3, skip, 40;
```

Sybase Backup Server

If you are using bundled Sybase, you must create and start a backup server.

Note: If you are running on Dynix, NCR, Pyramid, SCO UNIX, SCO UnixWare, or Solaris, skip the following section and see Creating a Backup Server Using autotli.

Creating a Backup Server

To create a backup server:

1. Edit the \$SYBASE/interfaces file
2. Copy the entire entry for AUTOSYSDB to a new entry.
3. In the new entry, change AUTOSYSDB to SYB_BACKUP.

When you are finished, you will have two entries in your interfaces file: one for AUTOSYSDB and another for SYB_BACKUP. Your SYB_BACKUP entry should look similar to the following:

```
SYB_BACKUP  
query tcp sun-ether fiji 5335  
master tcp sun-ether fiji 5335  
console tcp sun-ether fiji 5336
```

The previous example creates a backup server on a host named “fiji,” using port number 5335. The backup server port can be the same as the port used by the data server.

Note: The format of the interfaces file requires that a single tab (do not use spaces) precede the first word of every line that follows the SYB_BACKUP line (which should not have any spaces before it). A single space is used to delimit each element in an entry line. Incorrect formatting will prevent communication with the database.

Creating a Backup Server Using autotli

For this format, instead of editing the interfaces file directly, you run the autotli command to define the backup server and append the output to the interfaces file.

To create a backup server using autotli:

Enter the following command:

```
$AUTOSYS/install/autotli -s SYB_BACKUP -h host -p port >> $SYBASE/interfaces
```

where:

SYB_BACKUP

Specifies the name of the backup server.

host

Specifies the host name.

port

Specifies the backup server port. The backup server port can be the same as the port used by the data server, as long as the backup server and data server are on different machines.

>>

Appends the information to the \$SYBASE/interfaces file. (Be sure to use two redirect symbols (>>); a single redirect (>) symbol will overwrite the existing file.)

For example, the following command creates the backup server “SYB_BACKUP” on host “fiji,” using port 5335:

```
$AUTOSYS/install/autotli -s SYB_BACKUP -h fiji -p 5335 >> $SYBASE/interfaces
```

The previous command will create the following entry in the \$SYBASE/interfaces file:

```
# SYB_BACKUP on fiji (192.186.244.21) using tcp
#   services: query (5335) master (5335) console (5336)
#
SYB_BACKUP
query tli /dev/tcp x:00>214d7c0a8f41500000000000000000000
master tli /dev/tcp x:00>214d7c0a8f415000000000000000000000
console tli /dev/tcp x:00>214d8c0a8f4150000000000000000000
```

Starting the Backup Server

To start the backup server:

Enter the following command:

```
$SYBASE/install/RUN_SYB_BACKUP
```

If you already defined a dump device to Sybase, continue with the next section. Otherwise, you must define a disk or tape dump device to Sybase, as described in Defining a Dump Device in this chapter.

Dumping the Database

Verify that SYB_BACKUP (the backup server) is running before you dump the database. If it is not running, start the backup server by entering the following command:

```
$SYBASE/install/RUN_SYB_BACKUP
```

To dump a Sybase database:

Enter the following xql commands:

```
xql -Usa -Psysadmin
xql>dump database database_name to dump_device;
```

where:

database_name

Is the name of the database that contains your data. By convention, this is "autosys."

dump_device

Is the name of the Sybase dump device that you already defined to Sybase.

For example, the following writes to a dump device named "autodump":

```
xql>dump database autosys to autodump;
```

Loading the Database

To load the database:

1. Sign on to the server that will contain the second event server.
2. Enter the following xql command:

```
xql -Usa -Psysadmin
```

3. To place the database in single user mode and guarantee that no transactions can occur while the load is in progress, add the following database options:

```
xql>sp_dboption autosys, "no chkpt on recovery," TRUE;
xql>sp_dboption autosys, "dbo use only," TRUE;
xql>sp_dboption autosys, "read only," TRUE;
```

4. To issue a checkpoint, enter the following:

```
xql>checkpoint;
```

5. To load the database backup, enter the following:

```
xql>load database database_name2 from dump_device;
```

where:

database_name2 Is the name of the database that contains your data. By convention, this is “autosys.”

dump_device Is the name for the Sybase dump device that you already defined to Sybase.

For example, the following will load a disk dump named “autodump” of the database named “autosys”:

```
xql>load database autosys from autodump;
```

6. After the load has succeeded, enter the following to unset the single user options that you set before executing the load:

```
xql>sp_dboption autosys, "no chkpt on recovery," FALSE;
xql>sp_dboption autosys, "dbo use only," FALSE;
xql>sp_dboption autosys, "read only," FALSE;
```

7. To bring the database online, enter the following:

```
xql>online database autosys;
```

Recovering a Bundled Sybase Database

To recover a damaged database using the backup file:

1. Stop the event processor.
2. Drop the damaged database.
3. Re-create the database.
4. Reload the database.
5. Restart the event processor.

WARNING! You should drop the database and re-create it only in extreme situations. Before using this process to recover a damaged database, investigate all other options.

Stopping the Event Processor

To stop the event processor:

Log on as the exec superuser and issue the following command:

```
sendevent -E STOP_DEMON
```

Dropping the Damaged Database

You can drop the damaged database by using the drop database command. Before you drop the damaged database, however, you should ensure that you have a database dump file to use to restore the database.

To drop the damaged database:

1. Enter the following xql command (assuming that the "sa" password is "sysadmin"):

```
xql -Usa -Psysadmin
```

2. At the xql prompt, enter the following:

```
xql>>[AUTOSYSDB] [master] 1> drop database autosys;
```

If the database is so damaged that drop database does not work, contact Computer Associates Technical Support.

Note: The semicolon at the end of the command line is an end-of-statement delimiter in xql, replacing the Sybase "go."

Re-Creating the AutoSys Database

After dropping the damaged database, you must recreate a new database. This new database is used to load the database dump file (for example, the autodump file) that you are recovering. Use the create database command to create the new database.

To re-create the database:

1. To create the new database, at the xql prompt, enter the following:

```
xql>>[AUTOSYSDB] [master] 1> create database  
xql>>[AUTOSYSDB] [master] 2> autosys on default  
xql>>[AUTOSYSDB] [master] 3> = 50;
```

Note: The above size of 50 MB is only an example.

The output generated by this command will look similar to this:

```
Msg 1805, Level 0, State 1, Line 1 CREATE DATABASE:  
allocating 15360 pages on disk 'default'
```

2. To change the owner of the new database to "autosys," enter the following at the xql prompt:

```
xql>>[AUTOSYSDB] [master] 1> use autosys;  
xql>>[AUTOSYSDB] [autosys] 1> sp_changedbowner  
xql>>[AUTOSYSDB] [autosys] 2> autosys;
```

When the procedure has completed successfully, a message similar to this should be returned:

```
Database owner changed.
```

Reloading the Database

You are now ready to execute the load database command to restore the database you originally dumped to autodump.

To reload the database and put it online:

Enter the following at the xql prompt:

```
xql>>[AUTOSYSDB] [master] 1> load database autosys  
xql>>[AUTOSYSDB] [master] 2> from autodump;  
xql>>[AUTOSYSDB] [master] 1> online database autosys;
```

The database is now restored and online.

Note: For Sybase System 11, you must put the database online. Previous versions of Sybase did not require this.

To exit xql:

Enter the following at the xql prompt:

```
xql>>[AUTOSYSDB] [master] 1> exit
```

Restarting the event processor

When you complete this process, Unicenter AutoSys JM will be in the state it was when the database was dumped to the backup file.

The runtime behavior is controlled by the parameters in the configuration file and the environment variables set in the /etc/auto.profile file. This chapter describes these files.

Note: If you are running on Windows, the configuration parameters are set through the Unicenter AutoSys Administrator. For more information on the Unicenter AutoSys Administrator, see the *Unicenter AutoSys Job Management for UNIX User Guide*.

Configuration File

Upon startup, Unicenter AutoSys JM reads the configuration file to determine its behavior, including which databases to connect to and how to react to certain error conditions. In particular, the event processor bases much of its runtime behavior on the parameters found in this file. If you change the settings in the configuration file, you must stop and restart the event processor so it can read the new settings.

The configuration file has the following name:

\$AUTOUSER/config.\$AUTOSERV

The file is instance-specific, and the \$AUTOSERV value is the name of the instance of which the configuration file is associated. The \$AUTOSERV variable must be three uppercase alphabetic characters and must be unique to each instance.

Note: Events have a unique ID called eoid, which is prefixed by the first three letters of \$AUTOSERV. This ensures an event's uniqueness and traceability across multiple instances.

Sample Configuration File

A sample configuration file can be found in the file named \$AUTOSYS/install/data/config.ACE. You might want to reference this file as you read through this chapter. The configuration file has the following entries:

```
# Configuration file for AutoSys
#
# Specify the Databases
#
EventServer=AUTOSYSDB:autosys
# Sybase database example
#EventServer=SYB SERVER2:autosys
# Oracle database example
#EventServer=ORASERVER2
# # # # Tunable Parameters # # # #
#
# # # Database Connection Parameters # # #
#
# Database Connection Time Out (in seconds)
DBLibWaitTime=90
# # # Cross Instance DB connection Flag # # #
# If = 0 connect before every EVENT is sent to the
# other Instance and drop after it is sent.
# If > 0 connect the first time an EVENT is sent, and
# maintain the connection forever.
XInstanceDBDropTime=1
#
# Number of times for Event Processor to attempt to
# Reconnect to Event Servers
#
DBEventReconnect=50
#
# USE the following for Dual DataBases
#DBEventReconnect=50,5
#
# # # # Event Processor Parameters # # # #
#
# Event Processor ERRORS
# If there are more than 20 errors in 60 seconds it
# will exit. The MAX value of EDNumErrors is 100.
EDNumErrors=20
EDErrTimeInt=60
#
# Machines for chk_auto_up to inspect to see if an
# Event Processor is running there.
#EDMachines=host1,host2,host3
EDMachines=localhost
#
# For Shadow Event Processor, the Third Machine to
# resolve contention problems between Event
# Processors
#ThirdMachine=a_hostname
#
# Minimum kbytes of space that must be available for
# the EP log file (on the partition or disk). If free
# space falls below this, the EP will issue warnings
# and shut down.
#
FileSystemThreshold=32
# Event Processor's internal Daily Database
# Maintenance Cycle # #
#
# Daily start-time
```

```
DBMaintTime=03:30
# Command to Run to perform Maintenance
#
DBMaintCmd=$AUTOSYS/bin/DBMaint
#
# For Dual Server Mode - transfer events timeout
EvtTransferWaitTime=5
#
# Check Heartbeat every 2 minutes
#Check_Heartbeat=2
#
# Output Directory for the Remote Agent
#
# Note: Some OS's have problems with file locks in /tmp
# If so use some directory other than /tmp.
#
AutoRemoteDir=/tmp
#
# Clean Remote Agent files: 1=Remove files if No
# Problems!
CleanTmpFiles=1
#
# Create Remote Agent Output File for Sourcing the
# Environment
# In UNIX: capture std_out & std_err from sourcing
# /etc/auto.profile
# In NT: output the Environment to the file
#
RemoteProFiles=1
#
# Host machines to send SNMP traps to. (Specifying
# a machine ENABLES traps)
#SnmpManagerHosts=host1,host2
#
# Snmp community. This is almost always "public"
#SnmpCommunity=public
# Enable sending HP Operations Center messages (opcmsg)
# for AutoSys alarms.
# This defines the message group under which
# messages will be sent.
#OpcMessageGroup=Job
#
# This parameter sets the amount of time that the
# event_demon will wait for the OpCenter message
# to be sent.
#
#OpcWaitTime=4
#
# RESTART configuration stuff
#
# Max number of times to RESTART a job due to system
# errors
MaxRestartTrys=10
#
# Formula for computing the Wait time between
# restart attempts:
# WaitTime = RestartConstant + (Num_of_Trys *
#     RestartFactor)
#   if (WaitTime > MaxRestartWait) then
#     WaitTime = MaxRestartWait
#
RestartConstant=10
RestartFactor=5
MaxRestartWait=300
# Preferred method of Load Balancing
#   Can be: vmstat | rstatd (default is vmstat)
```

```
#MachineMethod=rstatd
#
# List of Signals to Send to a Job for the KILLJOB
# event
KillSignals=2,9
#
# Port number of auto_remote
AutoRemPort=5280
#
# Specify if standard error and standard output
# files should be appended to or overwritten.
# 0 overwrites the file.
# 1 appends the file.
AutoInstWideAppend=1
#
```

The following sections describe these configuration parameters.

Configuration File Parameters

You can modify the parameters in the configuration file to control the behavior and optimize your system.

The event processor reads the settings in the configuration file on startup only. Therefore, if you make changes to the file, you must stop and restart the event processor so it can read the new settings.

To stop and restart the event processor:

1. Stop the event processor, using the following command:

```
sendevent -E STOP_DEMON
```

2. Restart the event processor, enter the following:

```
eventor
```

Or, if you are running with a shadow event processor, you can start both the primary and the shadow event processors at the same time, like:

```
eventor -M shadow_machine
```

Database Time-Out Period

DBLibWaitTime (Sybase Only)

The DBLibWaitTime configuration parameter specifies the amount of time the event processor will wait before breaking the connection with an event server that is in an unknown state. The event processor continually maintains and checks its connections with the databases, and when an event server goes into an unknown state, the event processor will break the connection after the wait time specified by the DBLibWaitTime parameter.

The following line in the configuration file sets the timer for 90 seconds:

```
DBLibWaitTime=90
```

Typically, the database should never time out. However, if a database does time out, AutoSys will attempt to reconnect to the database the number of times specified by the DBEventReconnect parameter. If you see the database connections timing out often, it probably indicates some kind of machine or data server contention problem.

Note: If you set this value to DBLibWaitTime=0, it means that no time-out value is to be applied—the connection is continuous. Because it can cause the event processor to hang, this setting is not recommended.

Cross-Instance Database Connections

XInstanceDBDropTime

The XInstanceDBDropTime parameter specifies how an instance will connect with another instance if it has a job defined with a cross-instance job dependency.

If the value of this parameter is equal to zero, the event processor will connect to the other instance before every new event is sent. After the event has been sent, the connection will be dropped. Use this option for instances that are using cross-instance job dependencies infrequently.

If the value of this parameter is equal to one, the event processor will connect to the other instance before the first event is sent, and it will maintain the connection indefinitely. Use this option for instances that are using cross-instance job dependencies often.

The following line in the configuration file sets the cross-instance connection to 1, and thus, the event processor will maintain the connection after making the initial connection:

```
XInstanceDBDropTime=1
```

SNMP Connections

Unicenter AutoSys JM can be integrated with Hewlett-Packard's Node Manager software, versions 4.10 or higher. This enables OpenView users to do the following:

Monitor all alarms generated Unicenter AutoSys JM.

Monitor all UNIX signals received by the Event Processor.

Specify that certain commands be issued when an alarm or signal is received by OpenView.

Unicenter AutoSys JM sends alarms and signals to OpenView using the Simple Network Management Protocol (or SNMP), and posts alarms and signals using the SNMP trap mechanism.

Note: Whenever the Event Processor receives a UNIX signal, it posts an SNMP event to OpenView. This can be particularly useful if a signal has been sent that will shutdown the Event Processor. The signal will be posted to OpenView before the Event Processor shuts down.

To integrate with OpenView, both of the following parameters must be configured for alarms to be detected by OpenView.

SnmpManagerHosts

The SnmpManagerHosts parameter specifies the machines that will send SNMP traps. It contains a list of machines on the network that are running SNMP managers, such as HP's OpenView or IBM's NetView, and to which you want to send SNMP traps (for example post SNMP events). By entering the name of a machine with this parameter, you enable this functionality.

The entry in the configuration file would look like:

```
SnmpManagerHosts=host1,host2
```

SnmpCommunity

The SnmpCommunity parameter specifies the SNMP community associated with all SNMP traps sent. This parameter is effectively a password that can be used to filter SNMP traps by SNMP managers, such as HP's OpenView. This value is almost always public, and thus, the entry in the AutoSys configuration file usually looks like:

```
SnmpCommunity=public
```

Database Connections

Unicenter AutoSys JM can be configured to attempt connect, and reconnect, to databases a specified number of times. This behavior occurs when the first attempt to connect to the database is made, and when a database connection has been lost and there is a reconnect attempt made. This database connection behavior also sets the rollover criteria for dual-server mode.

DBEventReconnect

The DBEventReconnect parameter controls the number of times an event processor should attempt to connect (or reconnect) to an event server before shutting down, or before rolling over to single-server mode. This parameter is used on startup and when there is a connection problem during runtime.

In single-server mode, this parameter is set to a simple number, like:

```
DBEventReconnect=50
```

This setting specifies that the event processor should attempt a connection with the event server 50 times. If it cannot connect after 50 attempts, it shuts down.

In dual-server mode, this parameter contains two values describing the connection and rollover behaviors, like:

```
DBEventReconnect=50,5
```

This setting specifies that the event processor should attempt five connections with the event servers. If after five times it cannot connect, it should rollover to single-server mode, marking the other event server as “down.” Once in single-server mode, the event processor should attempt a connection 50 times, and if it is unsuccessful, the event processor shuts down.

Upon startup, the event processor attempts to connect to the event servers five times. If the event processor is unable to connect to both databases, it assumes there is a connection or configuration problem, and will gracefully shut down.

Event Processor Cascading Errors

To guard against cascading errors, you can set bounds that will force Unicenter AutoSys JM to automatically shut down the event processor if too many errors occur within a certain period of time. The EDNumErrors and EDErrTimeInt parameters work together to guard against these cascading event processor errors, which are caused by the event processor automatically reissuing failed directives.

The primary reason for setting these two parameters to shut the event processor down in this situation is to avoid starting any new processes while there are serious problems in the system.

EDNumErrors, EDErrTimeInt

The EDNumErrors parameter specifies the maximum number of errors that can happen within the time specified by the EDErrTimeInt parameter, in order to determine if Unicenter AutoSys JM should shut down the event processor. If the specified Number of Errors occurs within the Error Time Interval, shuts down the event processor as a safety measure.

If there are more than EDNumErrors errors within EDErrTimeInt seconds, Unicenter AutoSys JM shuts down the event processor. The default settings specify to shut the processor down if more than 20 errors occur within 60 seconds, and the entries in the configuration file look like this:

```
EDNumErrors=20  
EDErrTimeInt=60
```

Machines to Check for Running Event Processors

EDMachines

The EDMachines parameter specifies a list of valid server machines on the network. Unicenter AutoSys JM checks all of the machines on this list for running event processors, both when chk_auto_up is run and when a new event processor is started.

The list should contain all of the machines that could have a started and running event processor on them, so that a new event processor cannot be started on a different machine. Having a complete list of EDMachines is especially critical when a shadow event processor takeover has occurred, and your primary server is configured to automatically restart the event processor.

The entry in the configuration file looks like:

```
EDMachines=host1,host2,host3
```

Third Machine for Event Processor Contention

ThirdMachine

When running with a shadow event processor, Unicenter AutoSys JM uses a “third machine” to resolve possible contentions between the two event processors, and to ensure that only one of the processors is running at any given time.

If the shadow event processor does not receive a “ping” from the primary event processor, or if the primary event processor cannot signal the shadow event processor, the processors connect to this third machine and create a .dibs file that locks the other processor out. The .dibs file indicates that the one event processor has taken over and the other should shut down.

The third machine must have a remote agent installation on it. In addition, the third machine’s hostname must be in the configuration file on both the primary and shadow event processor machines. For example, to specify the machine named “ferrari” as the third machine, enter the following line in the configuration files for both event processor machines:

```
ThirdMachine=ferrari
```

Note: The third machine must have a remote agent installed on it, and it must have a valid client license. In addition, the third machine must be installed on the same type of machine as the primary and shadow event processors, either Windows or UNIX.

For more information on running with a shadow event processor, see the shadow event processor section in Chapter 1, Introduction, of the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

Event Processor Log Disk Space

The event processor will shut down if there is less than 8 KB of disk space available to write to the event processor log file (event_demon.\$AUTOSERV). If the amount of available disk space falls below that specified by the FileSystemThreshold parameter, the event processor will issue warnings in the event processor log file similar to the following:

```
WARNING: The disk partition containing the EP log file is too full!
WARNING: EP will shutdown if partition has less than 8192 bytes available!
```

The default FileSystemThreshold setting is 32 KB. Valid settings must be less than 10 MB and greater than 8192 bytes.

If the amount of disk space falls below 8 KB, the event processor will issue an EP_SHUTDOWN alarm and shut down, issuing messages similar to the following:

```
ERROR: No disk space left to write Event Processor log
EVENT: STOP_DEMON
The Event STOP_DEMON has just been received. We are going down!
```

The entry in the configuration file looks like:

```
FileSystemThreshold=32
```

Internal Database Maintenance

Once a day, the event processor goes into a database maintenance cycle. During this time, it does not process any events, and it waits for the maintenance activities to complete before resuming normal operations.

In the configuration file, you can specify the time of day for this maintenance cycle, and you can specify a maintenance command. The time you specify should be during a time of minimal activity.

Unicenter AutoSys JM provides a database maintenance script called DBMaint to run during this maintenance cycle. This script does the following:

- Updates statistics for the optimizer, checks the available space in the database, and sends a DB_PROBLEM alarm if the amount of free space is insufficient.
- Cleans out old information from the database tables using the archive_events command.

The DBMaint file is installed in the \$AUTOSYS/bin directory. We recommend that you configure your system to backup the database during this maintenance cycle.

For more information on using the DBMaint script and backing up a bundled Sybase database, see DBMaint Script and Bundled Sybase Backup and Recovery in this chapter.

DBMaintTime, DBMaintCmd

The following entries in the configuration file instruct the event processor to begin its maintenance cycle at 3:30 a.m. every day, and to execute the \$AUTOSYS/bin/DBMaint script at that time:

```
DBMaintTime=03:30  
DBMaintCmd=$AUTOSYS/bin/DBMaint
```

Event Transfer

When in dual-server mode, the event processor will copy a missing event from one event server to the other event server, after a time-out delay. This time-out delay is specified by the EvtTransferWaitTime parameter in the configuration file. The default setting of five does not usually need to be modified.

EvtTransferWaitTime

To set the default behavior of five seconds for the time-out, the configuration file contains the following line:

```
EvtTransferWaitTime=5
```

Sendevent Retries

The following two configuration-file parameters control how many times and how frequently the sendevent command will attempt to send an event to the event server database.

SendeventMaxRetries

Specifies the maximum number of times that the sendevent command will attempt to send an event to the event server database.

To set the number of retry attempts to 10, enter the following line in the configuration file:

```
SendeventMaxRetries=10
```

SendeventRetryInterval

Specifies the interval, in seconds, between attempts to send an event to the event server database. There is no default value.

To set the interval to 15 seconds, enter the following line in the configuration file:

```
SendeventRetryInterval=5
```

Heartbeats

Heartbeats offer a method by which the continued progress of an application can be automatically monitored. That is, you can program your user applications to send “heartbeats” that can be monitored. A heartbeat is a signal (SIGUSR2) sent from the application to the remote agent that started the application. That remote agent then sends a HEARTBEAT event to the event servers.

The event processor will check that the HEARTBEAT event has occurred within the heartbeat interval specified for the job.

Note: The event processor, and not the remote agent, checks if there is a HEARTBEAT between the remote agent and the event servers, and if the HEARTBEAT is absent, there is a problem, and an alarm is sent. Therefore, the HEARTBEAT option also provides a good indication of the stability of the network.

Check_Heartbeat

The Check_Heartbeat parameter specifies the interval value (in minutes) that you want the event processor to use when checking for heartbeats. If there are no applications sending heartbeats, you do not have to set this parameter. By default, this parameter is commented out in the configuration file.

For example, to instruct the event processor to check for missing heartbeats every two minutes, you would uncomment the following line in the configuration file:

```
Check_Heartbeat=2
```

Shadow Event Processor Pings

In high-availability mode, the shadow event processor pings the primary event processor periodically to ensure that it is still running. The following parameter determines the ping interval.

ShadowPingDelay

Specifies the interval, in seconds, after a successful ping of the shadow event processor before another ping is attempted. The default is 60 seconds.

To set the interval to 30 seconds, enter the following line in the configuration file:

```
ShadowPingDelay=30
```

Remote Agent Log Files Directory

During its operation, the remote agent writes output messages to files in the directory specified by the AutoRemoteDir parameter. For some operating systems, the locking of files located on the /tmp file system is not supported (for example, on SunOS platforms when /tmp is mounted on tmpfs). Since Unicenter AutoSys JM uses the locks to determine if the remote agent is running, another directory must be specified.

The AutoRemoteDir parameter is passed to the remote agent by the event processor when it starts. As a result, the remote agent log files directory must already exist, and be “writable” on every machine running.

In a cross-platform environment where a UNIX event processor starts a Windows remote agent (or vice versa), the path to the log files directory is translated into the format expected by the recipient platform. A UNIX remote agent will remove the drive letter and colon, if present, and replace \ characters with / characters. For example, C:\tmp becomes /tmp. A Windows remote agent will prepend the system drive letter and colon if they are not present, and replace all / characters with \ characters. For example, /tmp becomes C:\tmp.

AutoRemoteDir

The following entry in the configuration file specifies that the remote agents should use the /tmp directory for enterprise-wide logging:

```
AutoRemoteDir=/tmp
```

File Maintenance

CleanTmpFiles

For every job that Unicenter AutoSys JM runs, it creates a file in the remote agent Log directory on the machine where the job runs. If CleanTmpFiles is turned off, these files remain on each machine until they are removed with the clean_files command.

As an alternative to using the clean_files command, you can set the value for the CleanTmpFiles variable in the configuration file to be equal to 1, like:

```
CleanTmpFiles=1
```

Then, upon the successful completion of its tasks, the remote agent will remove the /tmp/auto_rem* file (assuming the default /tmp directory is specified by the AutoRemoteDir parameter). This is the format of the remote agent log filename, the auto_rem* file that is removed:

```
auto_rem.joid.run_number.ntry
```

If the remote agent cannot run the job successfully, the files will not be removed because they are useful to have when diagnosing the run problem.

Notes: To view the remote agent output file, use the autosys_log command on the client machine, like:

```
autosys_log -J job_name
```

Regardless of how you set the CleanTmpFiles parameter, you should run the clean_files command on a periodic basis to remove files from unsuccessful remote agent processes.

RemoteProFiles

When the RemoteProFiles parameter is turned on, it redirects to a file any stderr and stdout output generated when the /etc/auto.profile file is sourced. This parameter is “on” by default, and the entry in the configuration file looks like:

```
RemoteProFiles=1
```

The name of the file to which the profile output is written is based on the log file name. This is the form of the auto_rem_pro* filename:

```
auto_rem_pro.joid.run_number.ntry
```

This output file will contain entries if anything specified in the profile file failed (for example, environment variables or definitions were not set). For example, if the profile file attempts to set an environment variable using setenv, the Bourne shell that runs would not be able to process this C Shell syntax, and the output file would contain the following line:

```
setenv: not found
```

Non-fatal errors when a profile file is sourced are not recorded and will not appear in the output file.

To view the profile output file, use the autosys_log command on the client machine, like:

```
autosys_log -J job_name -p
```

This command will display the log file first, appending the profile output, if there is any. If no profile output file exists, the log file will display:

```
File: profile_output_file Does Not Exist.
```

Note: If CleanTmpFiles is turned on (set to 1), the output file will be removed when the job completes successfully, and the profile log information will not be available. If CleanTmpFiles is turned off, the file will remain until it is removed with the clean_files command.

Number of Restart Attempts

MaxRestartTries

Unicenter AutoSys JM attempts to restart a job if it has problems starting the job due to system problems, such as machine unavailability, the socket connect timed out, the remote agent was unable to start another process, or the file system space resource check failed. Unicenter AutoSys JM attempts to restart a job the number of times specified by the MaxRestartTries parameter.

For example, to set the number of job restarts to five, include the following line in the configuration file:

```
MaxRestartTries=5
```

Note: This parameter governs retries that result because of system or network problems. It is different from the n_retrys job definition attribute, which controls restarts when a job fails due to application failure (for example, Unicenter AutoSys JM is unable to find a file or a command, or permissions are not properly set).

Calculating the Wait Time Between Restarts

RestartConstant, RestartFactor, MaxRestartWait

The following formula is used to calculate the amount of time (in seconds) between each restart of a job:

```
WaitTime=RestartConstant+(Num_of_Trys*RestartFactor)
if WaitTime > MaxRestartWait,
then WaitTime = MaxRestartWait
```

where:

WaitTime Is the calculated interval in seconds to wait before attempting the next restart of a job.

RestartConstant Is a constant value you specify.

Num_of_Trys Is the counter tracking the number of times the job has already tried to start.

RestartFactor Is a factor you specify.

MaxRestartWait Is the maximum amount of time (in seconds) before attempting to restart a job.

If the calculated *WaitTime* is greater than the specified value for *MaxRestartWait*, then the resulting *WaitTime* is set to *MaxRestartWait*.

For example, the entry in the configuration file might look like:

```
RestartConstant=10
RestartFactor=5
MaxRestartWait=300
```

Method of Load Balancing

MachineMethod

This parameter specifies the method used to determine the percentage of CPU cycles available on a real machine belonging to a virtual machine. This method is used to achieve load balancing. The possible choices are *rstatd* and *vmstat*. If *MachineMethod* is not specified, *vmstat* is the default setting.

For example, to set the method to *rstatd*, enter this in the configuration file:

```
MachineMethod=rstatd
```

If rstatd is chosen, you must ensure that this demon is running on all client machines.

To ensure that the demon is running, do the following:

- Edit the internet demon configuration file (/etc/inetd.conf) on all client machines, and un-comment the rstatd entry.
- Send a SIGHUP signal (kill -1) to reset the running inetd process.

Sometimes, a kill -1 command is not sufficient to reset the inetd. If rstatd fails, you might have to issue a kill -9 command, then restart inetd. If necessary, check with your systems administrator.

Note: rstatd is not currently supported on NCR or Pyramid client machines.

KILLJOB Signals

KillSignals

The KillSignals parameter specifies a comma-separated list of signals to send to a job whenever the KILLJOB event is sent. If the job is running on a UNIX machine, the parameter specifies a single or comma-delimited list of UNIX signals to be sent to the job. If a list of signals is specified, the signals are sent in the order listed, with five-second sleeps between each call.

If the job to be killed is running on a Windows machine, this list is ignored and the job is simply terminated.

To preserve backward compatibility, the entry in the configuration file is:

```
KillSignals=2,9
```

We recommend that you set this parameter to 15,9. In most cases, this will lead Unicenter AutoSys JM to return a TERMINATED state for a job that was killed. If it does not, as might happen for some shells on some operating systems, set the KillSignals parameter to 9.

Note: The KillSignals listed in the configuration file are overridden when issuing the sendevent command with the -k option.

Port Number for Remote Agent

AutoRemPort

The event processor communicates to the remote agent by way of a TCP/IP socket connection, and the port number for this socket connection is specified by the AutoRemPort parameter. The internet services demon (inetd) on the client machine uses the port number to point to the name of the service (found in /etc/services). The service name is located in the inetd configuration file (/etc/inetd.conf), where it finds the path name to the remote agent binary.

It is possible to have different versions installed on the same hardware, where the versions are not cross compatible between the event processor and the remote agent. By setting up multiple services and using different port numbers, you can maintain multiple versions of the software.

The AutoRemPort number in the configuration file is set at installation time. If you alter it, you must change both the AutoRemPort number and the port numbers in all the /etc/services files on all AutoSys client and all server machines.

This is the entry in the configuration file, with the default setting of 5280:

```
AutoRemPort=5280
```

Note: If you are using NIS or NIS+, and wish to change AutoRemPort, you must modify /etc/services on your NIS or NIS+ master and push it to all client machines, and then do a kill -1 process on the inetd.

Cross-Platform Scheduling

AutoSysAgentSupport

The AutoSysAgentSupport parameter specifies whether the event processor should start the asbIII process for cross-platform scheduling. If set to 1, cross-platform scheduling is enabled.

If the value of this parameter is equal to zero, then the event processor will not send jobs to AutoSys Connect and AutoSys Agent managed machines. If the value of this parameter is equal to one, then the event processor will send jobs to these machines.

By default, Unicenter AutoSys JM Connect and Unicenter AutoSys Agent job support is off, and the entry in the configuration file looks like:

```
AutoSysAgentSupport=0
```

For more information, see the appendix “Integrating with the Mainframe and Unicenter AutoSys Agents for AS/400 and OpenVMS,” in this guide.

Instance Wide Append Parameter

AutoInstWideAppendx

The AutoInstWideAppend parameter specifies whether an instance will overwrite or append information to the standard output and standard error files.

If the value of this parameter is equal to zero, then the files will be overwritten. If the value of this parameter is equal to one, then new information will be appended to the files.

By default, new information is appended to the files, and the entry in the configuration file looks like:

```
AutoInstWideAppend=1
```

Each client machine can override the instance-wide setting by using the AutoMachWideAppend variable in the /etc/auto.profile file. If specified, this variable would appear as shown in the following example:

```
#AUTOENV#AutoMachWideAppend=TRUE
```

Note: If you are running jobs across platforms, the event processor of the issuing instance controls the default behavior. For Windows, the default is to overwrite this file.

An individual job definition can override either the instance-wide or the machine setting by placing the following notation as the first characters in the standard output and standard error files specifications:

```
>> Overwrite file  
>> Append file
```

Job Starting Interval

InetdSleepTime

The InetdSleepTime parameter controls how long the inetd waits between job starts on the same remote agent machine. By default, this is set to two seconds, and there is no parameter located in the configuration file. To reduce the time the inetd waits between job starts on a machine, you can add the InetdSleepTime parameter to the configuration file and lower this number, which is specified in seconds.

To lower the default setting, add an entry like this to the configuration file:

```
InetdSleepTime=1
```

Note: Setting InetdSleepTime too low for your hardware could adversely affect performance. In addition, you must ensure your machine has a processor fast enough to handle starting jobs at a faster interval. Otherwise, there will be frequent socket connection failures, which will cause numerous job restarts.

Unicenter Event Management Integration

UnicenterEvents

Before enabling Unicenter event integration, you must install the event manager agent on the event processor machine. The Event Agent can be installed alone, as part of Unicenter Framework, or as part of the entire Unicenter product.

When the UnicenterEvents parameter is non-zero, event_demon will start an additional demon, auto_wto, to transfer messages between the event processor and Unicenter. The auto_wto demon creates a FIFO (File In File Out,) /tmp/auto_wto_pipe, to communicate with the event processor.

By default, UnicenterEvents is off.

The example entry in the configuration file looks like:

```
UnicenterEvents=0
```

where:

0

Is one of the following:

- 0-To report no events to Unicenter (default setting)
- 1-To log all Alarms.
- 2-To only log Alarms and job completions.
- 3-To log all events that are generated to the Unicenter Console.

The auto.profile File

The remote agent is a process (called auto_remoteAgent") started by the event processor to perform a specific task on a remote (client) machine. The remote agent starts the command specified for a given job, sends running and completion information about a task to the event server, and then exits.

The remote agent starts on the client machine as "root." The remote agent environment is controlled through special descriptors located in the /etc/auto.profile file, located on the remote (client) machine.

The /etc/auto.profile file serves two purposes:

- It specifies a number of descriptors that set the environment for the remote agent on the client machine. These descriptors are environment variables that are preceded by the characters:

```
#AUTOENV#
```

- It specifies default settings for jobs that do not have a profile specified in the job definition. A job profile sets environment variables for a job immediately before the job is started.

You may want to view the /etc/auto.profile file in a text editor to familiarize yourself with the environment settings in this file.

For information about other settings that can be added to the auto.profile, see the following sections in this chapter:

- AutoMachWideAppend—See Instance Wide Append Parameter.
- DENY_USER—See Client-Side Security.

Sample Default auto.profile

The following example auto.profile file is for a Sybase installation.

```
# Set some environmental variables
#
# This must be a Bourne shell script,
# AND the variables exported for your command
# to see them.
# The following lines with $AUTOSYS and $AUTOUSER need to be
# uncommented if either:
# 1) running shadow EP, AND directories are in different locations
# 2) job's command is calling an AutoSys program.
#AUTOSYS=/dev/3.4/SYB ;export AUTOSYS
#AUTOUSER=/dev/3.4/SYB/autouser ;export AUTOUSER
# The following is for the windowing system
DISPLAY=:0.0
```

```
export DISPLAY
LD_LIBRARY_PATH=/usr/openwin/lib:/usr/lib/X11; export LD_LIBRARY_PATH
# set a PATH so executables can be found
PATH=.:$AUTOSYS/bin:$AUTOSYS/test/bin:/bin:/usr/bin:/usr/local/bin:/usr/openwin/b
in:/usr/bin/X11:/bin:/usr/ucb:/usr/etc"
export PATH
#####
# auto_remote environment variables
# DO NOT REMOVE
#AUTOENV#SYBASE=/usr/vendors/sadb
```

The remote agent looks for the #AUTOENV# descriptors and reads the variables that follow to set its environment. Do not remove the #AUTOENV# descriptors from the file. They are required to enable the remote agent to communicate with the database.

Note: The #AUTOENV# descriptor is not a comment. Do not remove the # character from the beginning of the line.

Remote Agent Database Connection Settings

Some of the environment variable descriptors in the auto.profile file define the event servers to which the remote agent should write events.

Sybase

For Sybase, the descriptor looks like:

```
#####
# auto_remote environment variables
# DO NOT REMOVE
#AUTOENV#SYBASE=/usr/vendors/sadb
```

The #AUTOENV#SYBASE descriptor defines where the remote agent looks for the Sybase interfaces file, which is specified by the SYBASE environment variable. If the interfaces file is not present in this location, the remote agent will not be able to write job statuses to the database.

Note: A common symptom that results from the remote agent being unable to write to the database is that jobs will remain in starting state.

Oracle

For Oracle, the descriptors look like:

```
#####
# auto_remote environment variables
# DO NOT REMOVE
```

```
#AUTOENV#TNS_ADMIN=/etc  
#AUTOENV#ORACLE_HOME=/var/opt/oracle
```

The #AUTOENV#TNS_ADMIN descriptor defines where the remote agent looks for the tnsnames.ora file, if the TNS_ADMIN environment variable is set. If the TNS_ADMIN variable is not set, the remote agent looks in the default directory, which is operating system specific.

The #AUTOENV#ORACLE_HOME descriptor defines the top-level Oracle directory, defined by the ORACLE_HOME environment variable.

Modifying Remote Agent Settings

To function correctly, the remote agents must have the correct TCP/IP socket connections, and they must have the appropriate environment set. The socket connection and environment values are set at installation time, but you can modify the settings.

Remote Agent Socket Connection

The event processor communicates with the remote agent by way of a TCP/IP socket connection. The port number for this socket connection is set in the following two files, and the number must be identical in both locations:

- \$AUTOUSER/config.\$AUTOSERV (AutoSys configuration file)
- /etc/services

In the configuration file, the socket connection is defined by the AutoRemPort variable, like:

```
# Port number of auto_remote  
AutoRemPort=5280
```

In the /etc/services file, the auto_remote entry defines the socket connection, like:

```
auto_remote 5280/tcp      # AutoSys Version 4.5
```

The internet services demon (inetd) on the client machine uses the port number defined in the configuration file to point to the name of the auto_remote service found in the /etc/services file. The service name is then located in the internet demon configuration file (usually /etc/inetd.conf), where it finds the path to the remote agent binary, like:

```
auto_remote stream tcp nowait root /usr/vendor/autotree/auto_remote auto_remote
```

Running Two Versions of Remote Agents

If you want to install two different versions of the remote agent on the same hardware, it is possible to set up multiple services by using different port numbers and service names. Each version would point to its own configuration file.

For example, assume you have one 3.3 remote agent using port number 3500. The entry in its AutoSys configuration file (for example, config.PRD) would look like:

```
AutoRemPort=3500
```

On the same machine, you could have a 3.4 remote agent using port number 4500. The entry in its configuration file (for example, config.DEV) would look like:

```
AutoRemPort=4500
```

If you do this, you must also modify the /etc/services file like:

```
auto_remote_33      3500/tcp    # AutoSys Version 3.3
auto_remote_34      4500/tcp# AutoSys Version 3.4
```

Then, you would modify the internet demon configuration file (/etc/inetd.conf) like:

```
auto_remote_33 stream tcp nowait root /usr/vendor/autotree_33/auto_remote
auto_remote
auto_remote_34 stream tcp nowait root /usr/vendor/autotree_34/auto_remote
auto_remote
```

Configuring Remote Authentication

You can configure to run using the following two methods of remote authentication:

- Unix ruserok() Authentication

When using this method, Unicenter AutoSys JM instructs a client's remote agent to make the UNIX system ruserok() call. This function checks the client machine's /etc/hosts.equiv and the user's .rhosts files to validate that the requesting user is registered in that environment.

- Unicenter AutoSys JM remote agent event processor Authentication

When using this method, a specific remote agent can be bound to one or more event processors. In other words, a remote agent will have to verify its permission to process an event processor's requests before starting each job. The remote agent does this by reading the /etc/.autostuff file on the machine where it is running.

Using the autosys_secure command, the edit superuser can enable (or disable) remote authentication. By default, remote authentication is initially disabled. If you enable remote agent, event processor authentication, you must configure Unicenter AutoSys JM to support it.

Configuring Event Processor Authentication

You can configure remote agents to only accept jobs from "trusted" event processors. To configure Unicenter AutoSys JM for remote agent event processor Authentication, you must do the following:

- Enable remote agent event processor Authentication.
- Create an ASCII file named.autostuff in the /etc directory of every client machine that will participate in this authentication method.

If both are present, the remote agent will only run jobs submitted by event processors listed in the autostuff file.

The /etc/.autostuff File

The /etc/.autostuff file should have read and write permissions for root only. Entries in this file must be in the following form:

AUTOSERV:hostname

where:

AUTOSERV

Is an instance name.

hostname

This is the name of the machine on which the event processor is running. This must be a real machine (if using DNS, this should be a fully-qualified name).

The file should contain an entry for each event processor you want authorized to run jobs on the remote agent machine. The entries cannot contain spaces within the event processor specification. You can use pound signs (#) for comments.

These are example file entries:

```
PRD:curly      #Production Instance
DEV:moe        #Development Instance
```

In this example, PRD is an instance name and the event processor for this instance is running on the machine curly. DEV is an instance name and the event processor for this instance is running on the machine moe. These event processors are authorized to issue jobs to the remote agent.

Client-Side Security

The AUTOENV environment variable DENY_ACCESS restricts access to the remote agent machine.

In the auto.profile file for the remote agent machine, you can specify a list of users whose jobs are prohibited from running on that machine. The list is a comma-delimited list of user names, with no spaces. The maximum number of characters is 512. For example:

```
#####
# auto_remote environment variables
# DO NOT REMOVE

#AUTOENV#DENY_ACCESS=root,demon,admin
```

In this example, jobs owned by root, demon, and admin will not be launched by the remote agent.

If a job owned by one of these users is submitted to run on the remote agent, the job fails as if the job's owner did not have a valid account on the machine. There will be no job restart attempts, regardless of MaxRestartTrys setting in the configuration file. When this occurs, the following error appears in the event processor log, as a STARTJOBFAIL alarm on the job:

```
Permission ERROR: Could not SET uid=uid on Host: host
```

User-Defined Alarm Callbacks

This notification mechanism provides a method for communicating problems to administrators in a manner that is external to the event system. That is, for certain types of alarms, you can configure Unicenter AutoSys JM to call user-defined routines that communicate the problem to specific members of your company. For example, by using electronic mail or a command line pager utility, your administrator can be notified as soon as certain events occur.

You can configure Unicenter AutoSys JM to call user-defined routines for the following types of system alarms:

- DB_ROLLOVER
Unicenter AutoSys JM has rolled over from dual-server to single-server mode.
- DB_PROBLEM
There is a problem with one of the databases.
- EP_ROLLOVER
The shadow event processor is taking over processing.
- EP_SHUTDOWN
The event processor is shutting down. This might be due to a normal shutdown, or due to an error condition.
- EP_HIGH_AVAIL
The third machine for resolving contentions between two event processors cannot be reached, one of the event processors is shutting down, or there are other event processor take-over problems.

To specify what executable should be invoked as a user-defined callback for one of the above alarms, a file named notify.\$AUTOSERV must be created in the \$AUTOUSER directory. An example of this file is provided in the \$AUTOSYS/install/data/notify.ACE file, which contains the following entries:

```
# Notify for certain CRITICAL ALARMS
#
# Form is: ALARM executable
# We pass in $1 = numeric code
#           $2 = Text Message
# Only have 1 space between the ALARM and the executable
#
# The environment is inherited from the Event Processor
# The following is executed: system( <executable>
# $1 $2 & );
#
#DB_ROLLOVER $AUTOUSER/notify_db
#DB_PROBLEM $AUTOUSER/notify_db
#EP_ROLLOVER $AUTOUSER/notify_ep
```

```
#EP_SHUTDOWN $AUTOUSER/notify_ep  
#EP_HIGH_AVAIL $AUTOUSER/notify_ep
```

Notification Example

To have Unicenter AutoSys JM call the program /usr/local/bin/pager when the event processor shuts down:

1. Copy the sample notification file to:

```
$AUTOUSER/notify.$AUTOSERV
```

2. Change the EP_SHUTDOWN line to:

```
EP_SHUTDOWN /usr/local/bin/pager $@
```

Then, Unicenter AutoSys JM will pass pager a numeric code and a text message. The pager itself must be coded to accept these parameters.

Problems with Unicenter AutoSys JM usually involve the interactions between the major components, rather than with the individual components themselves. This chapter presents a number of common problems, their symptoms, and how to resolve them. It provides very useful information about troubleshooting the primary components.

To troubleshoot more effectively, it is essential that you understand the stages in the life of a job, the order in which they occur, and the roles played by the three major components (that is, event server, event processor, and remote agent).

When a job is defined, its starting conditions are saved to the event server (database), and the following occurs:

- When its starting conditions are met, the event processor initiates a remote agent on the client machine to execute the job.
- The remote agent runs the job and sends the exit status of the job back to the event server.
- After the job completes, it is not run again until its starting conditions are met.

This is the basic cycle for all jobs.

For more information about job processing, see Basic Functionality in the chapter “Introduction,” in this guide. For information about troubleshooting CCI, see the appendix “Troubleshooting CCI” in this guide.

Event Server Troubleshooting

Event Server Is Down (Sybase)

Symptoms

1. When trying to start xql, you get a message like:

```
DB-Library error: dbproc NULL  
Error in SybInit: dbopen failed
```

2. When running programs like autorep or autosc, you get a messages like:

```
Client ERROR:
```

or:

```
Unable to connect: SQL Server is unavailable or does not exist.
```

3. When you run chk_auto_up, you receive a message similar to:

```
Couldn't connect with Server: AUTOSYSDB:autosys
```

4. You are unable to insert keys with gatekeeper.

Resolution

This indicates that either the data server is down, or the process in question is unable to access it. To confirm that the data server is down, log on to the server machine and run the chk_auto_up utility. You can also look at the process table (using the UNIX ps command) for the process name "data server."

If the database is indeed down, you must restart it. For directions on how to do this, see Starting Sybase in the chapter "Maintaining," in this guide.

If the database is running, the problem could be that you are pointing to the wrong data server. The DSQUERY environment variable points to the name of the data server (typically AUTOSYSDB). If it is not set properly and you are not specifying a data server name to xql (using the -S server option), then xql will fail.

The environment variables point to the configuration file \$AUTOUSER/config.\$AUTOSERV. This file contains the name of the event server. Check to make sure that the environment variables and the configuration file point to the proper location and event server.

Enter the following command for the instance ID:

```
echo $AUTOSERV
```

The results should look similar to:

```
ACE
```

Enter the following command for the event server and database name:

```
get_server $AUTOSERV
```

The results should look similar to:

```
E AUTOSYSDB autosys
```

where:

E Means that this is an event server.

AUTOSYSDB Is the name of the event server.

autosys Is the name of the database (for Sybase and Microsoft SQL Server).

If the database service is up, and the environment variable is set properly, then the Sybase interfaces file might not have the proper form, or be in the proper location. This typically occurs on servers if the environment has been changed, or, on clients, if the interfaces file has not been correctly installed.

For more information about the Sybase interfaces file and connecting to databases, see the chapter “Introduction” in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

Sybase Deadlock

Symptom

A message similar to the following appears in the event processor log when viewed with the autosyslog -e command or in the Sybase error log (\$SYBASE/install/errorlog_EventServer):

Your server command (process id #11) was deadlocked with another process and has been chosen as deadlock victim. Re-run your command.

Resolution

A deadlock is a Sybase condition that occurs when two users have a lock on separate objects, and they each want to acquire an additional lock on the other user's object. The first user is waiting for the second user to let go of the lock, but the second user will not let go until the lock on the first user's object is freed.

The data server detects the situation and chooses the user whose process has accumulated the least amount of CPU time as the "victim." The data server rolls back the victim's transaction, notifies the application with the above error message, and allows the other user's processes to move forward.

sendevent will try to rerun the command until it is successful or until it reaches the maximum number of tries specified by the -M option.

Not Enough User Connections (Bundled Sybase)

Symptoms

Users cannot make connections to the database; they cannot start the GUI or send events. When you check the Sybase error log (`($SYBASE/install/errorlog_EventServer)`) you see one or both of the following errors:

```
Not enough User Connections (Sybase error 1601)
No Pss structures available for new process
```

Resolution

These messages occur because there are more users who want to run jobs simultaneously than there are user connections; there are not enough connections available to the database. By default, the bundled Sybase installation of Unicenter AutoSys JM has a limit of 25 user connections. You can increase the number of user connections, but first you must determine the maximum number of user connections your system can support.

To determine the maximum number of user connections you can set for your system:

1. Log into the database as the "sa."
2. At the isql or xql prompt, enter:

```
1> select @@max_connections;
```

Results similar to the following are displayed:

```
-----
253
return status 0, rows affected 1
```

3. Enter the following:

```
1> select count(*) from master..sysdevices where cntrltype = 0;
```

Results similar to the following are displayed:

```
-----
3
return status 0, rows affected 1
```

4. Enter:

```
select count(*) from master..sysdevices where mirrorname is not NULL;
```

Results similar to the following are displayed:

```
-----  
0  
return status 0, rows affected 1
```

5. Enter:

```
1> select count(*) from master..sysservers where srvname != @@servername;
```

Results similar to the following are displayed:

```
-----  
1  
return status 0, rows affected 1
```

The maximum number of user connections that you can set is @@max_connections minus the sum of the results of the last three queries. In the example results to the above queries, the maximum number of user connections is 249.

To increase the number of user connections:

1. At the isql or xql prompt, enter the following command to specify the number of user connections you want:

```
1> sp_configure "user connections," number;
```

where:

number

Is the number of user connections you want.

WARNING! If you set the number of user connections too high, the database will be unusable. At this point, you might not be able to rerun sp_configure to lower the number of user connections. To return the database to working order, you must run buildmaster or recover the database from backups.

2. Stop and restart the event server. Changes will not take effect until you stop and restart the event server.

archive_events Fails (Bundled Sybase)

Symptom

When the transaction log is full, archive_events fails. This usually occurs because archive_events is not run regularly. We highly recommend that you run archive_events frequently, ideally as part of the daily database maintenance cycle by using DBMaint or as a regularly scheduled job.

You can usually avoid a full transaction log by having Sybase truncate the transaction log during regular checkpoints.

To have Sybase truncate the transaction log:

1. Log in to the database server as the “sa” by entering the following command (if you changed the “sa” password, use that one instead of sysadmin):

```
xql -Usa -Psysadmin
```

2. To check if the truncate option is set, enter the following commands:

```
1> sp_helpdb;
```

3. If trunc.log on chkpt is not set, enter one of the following commands depending on your version of Sybase:

```
1> sp_dboption autosys, "trunc log on chkpt," true;
```

or:

```
1> sp_dboption autosys, "trunc. log on chkpt.," true;
```

4. Enter the following commands:

```
1> checkpoint;  
1> exit
```

Resolution

To resolve the full transaction log problem:

1. Log into the database server as the “sa” by entering the following command (if you changed the “sa” password, use that one instead of sysadmin):

```
xql -Usa -Psysadmin
```

2. Use the database by entering the following command:

```
1> use autosys;
```

3. Dump the transaction log by entering the following command:

```
1> dump transaction autosys with no_log;
```

4. If this fails, enter the following commands:

```
1> set rowcount 1;
```

```
1> dump transaction autosys with no_log;
```

Repeat step 4 and each time increase the rowcount incrementally. To begin with, try doubling the number. If this fails at any time, decrease the rowcount and try again. When the rowcount is large, log out of the data server.

5. Log out of the data server, and then run archive_events. Begin with a large number of days and work down until you reach your target number of days. For example:

```
archive_events -n 30
```

```
archive_events -n 10
```

Event Server Will Not Start (Bundled Sybase)

Symptom

When starting a bundled Sybase event server, you get a message similar to the following:

```
00: 94/08/18 10:38:37:66 kernel: kcinit:  
couldn't open error log  
'/users/sybase/stdb/install/errorlog_AUTOSYSDB'  
(os error 13)
```

Resolution

You are not the same user who last started Sybase, and do not have permission to write to the Sybase error log file. Become that user, change the error log file (\$SYBASE/install/errorlog_\$DSQUERY) permissions, or delete the old error log file.

Event Processor Troubleshooting

Output from the event processor is redirected into the following log file:

```
$AUTOUSER/out/event_demon.$AUTOSERV
```

Everything that the event processor does, in the order it was done, is in this file. Network problems are usually reflected in this file as well. This file is very useful for reconstructing what happened when a problem occurs.

Event Processor Is Down

Symptoms

1. Jobs do not start.
2. When chk_auto_up is run, it returns a message similar to:

```
Could connect with Server: AUTOSYSDB:autosys
No Event Processor is RUNNING on machine: machine
```
3. The event processor log has not registered a date and timestamp for a period of time. The event processor log should register date and timestamps every minute.

Resolution

Confirm that the event processor is down by performing one of the following actions:

- Run the chk_auto_up utility.
- Perform a tail on the log file and check for date stamps.
- Look for the event_demon process using ps.

If the event processor is indeed down, log on as the exec superuser and run the eventor command to restart it.

Event Processor Will Not Start

Symptom

When running the eventor command, you see an error message similar to the following:

```
/usr/autotree/autosys/bin/eventor:/usr/autotree/autouser/out/event_demon.ACE:Cannot create
```

Resolution

You are not the same user who last started the event processor. Either become that user, or change permissions on the event processor output file, the \$AUTOUSER/out/event_demon.\$AUTOSERV file.

Remote Agent Troubleshooting

Remote Agent Verification

If you suspect problems with the remote agent, you can use autoping to verify your suspicions.

autoping

autoping is used to test the connections between the event processor and the remote agent. If you use the autoping -M -D client_hostname command, and it does not return an error, the remote agent should start properly.

The remote agent writes RUNNING and completion statuses directly to the event server.

Database Verification

Use autoping to verify the remote agent database connection. To check the database connections on machine, enter:

```
autoping -m machine -D
```

Instead of a single machine, you can type -m ALL to check all machines.

This command captures the output from the attempted database connection, displays it, and includes it in the alarm, if one is generated (use the -A argument to generate an alarm if problems are found).

```
autoping -m venice -D
AutoPinging Machine [venice] AND checking the
Remote Agent's DB Access.
AutoPing WAS SUCCESSFUL!
```

Remote Agent Will Not Start

The symptoms in this section are similar and result from network problems.

Symptom

The Event Processor's \$AUTOUSER/out/event_demon.\$AUTOSERV log file contains a message similar to the following:

```
Attempting to connect to AutoSys Remote Agent  
Service on socket=5280: Connection Refused  
Attempting to connect to 'inetd' on  
socket=5280: Interrupted system call  
The connection to machine: spartacus TIMED OUT.  
Either that machine, or the network to it, is  
having problems.  
ERROR trying to start job: test  
Error: Connect to socket FAILED.  
Command: sleep 1  
Machine: spartacus
```

Resolution

Either there is a network communication problem, or the client machine is down. The network or machine problems must be resolved before jobs can be run on that machine.

Symptom

The event processor's \$AUTOUSER/out/event_demon.\$AUTOSERV log file contains a message similar to:

```
Attempting to connect to AutoSys Remote Agent  
Service on socket=5280: Connection Refused  
Attempting to connect to 'inetd' on  
socket=5280: Interrupted system call  
Could NOT connect to machine: spartacus The  
internet demon may not be configured properly.  
ERROR trying to start job: test  
Error: Connect to socket FAILED.  
Command: sleep 1  
Machine: spartacus
```

Resolution

There is a problem with the /etc/services file. To locate this problem, check the following:

1. Check the /etc/services file to see if the following entry is there:

```
auto_remote 5280/tcp
```

2. Confirm that the remote agent port number, specified with AutoRemPort in the configuration file (\$AUTOUSER/config.\$AUTOSERV), is the same number as used in the /etc/services file.

3. If you are using NIS/NIS+, remake the services and push it out.

This refers to modifying the NIS/NIS+ master machine and propagating the information to all NIS/NIS+ clients. You should contact your NIS system administrator for instructions on how to do this.

Symptom

In the event processor's \$AUTOUSER/out/event_demon.\$AUTOSERV log file, you see a message similar to the following:

```
[spartacus connected]
*** Remote Agent Process not started. ***
socket read <>, rc=0
ERROR trying to start job: test
Error: Auto Remote process did not start.
Command: sleep 1
Machine: spartacus
```

Resolution

Check that the internet demon is properly configured on the client (remote agent) machine. There should be an entry in inetd.conf for auto_remote. If present, this line points to the remote agent's executable. Assuming that auto_remote is in the /usr/local/bin directory, and will be run by the user "root," the entry should look like this (on one line):

```
auto_remote stream tcp nowait root/usr/local/bin/auto_remote auto_remote
```

Note: The auto_remote command must be run as "root," because it does a setuid.

For more information about configuring the internet demon on the client machine, see the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

If the auto_remote entry exists in the inetd.conf file, follow these steps:

1. Make sure that the auto_remote binary (the remote agent executable) exists in the specified directory and has "execute" permission.
2. If the inetd.conf file needs to be modified, a SIGHUP (kill -1) needs to be sent to the inetd process to cause the configuration to be re-read. To do this, sign on as "root" and execute one of these commands:

```
$AUTOSYS/install/touch_inetd
```

Or execute:

```
kill -1 pid
```

where :

pid

Is the process ID of the internet demon (inetd).

This command forces the internet demon to re-read its configuration file, and to reset itself.

Note: On most AIX machines, the kill -1 command fails to reset the inetd. Instead, you must kill the inetd process using kill -9, and then restart the inetd manually.

3. Make sure the remote agent can write to the AutoRemoteDir directory (usually /tmp), and that this directory has available space.

Remote Agent Will Start - Command Will Not Run

Each time the remote agent is started on a machine, it creates the following log file:

AutoRemoteDir/auto_rem.pid

where:

AutoRemoteDir Is the remote agent log directory specified in the configuration file (usually specified as the /tmp directory).

auto_rem.pid Is the process ID of the remote agent.

When the remote agent receives its instructions from the event processor, it renames this file in order to give it a unique name. This is the form of the new filename:

AutoRemoteDir/auto_rem.joid.run_num.ntry

where:

joid Is the job object ID (the job's number in the database).

run_num Is the run job run number.

ntry Is the number of tries or restarts.

This file contains all the instructions passed to the remote agent by the event processor, the results of any resource checks, and a record of all actions it took. Any problems experienced by the remote agent are reported here, including the inability to send events to the databases, which is the most common problem.

To find the most recent instance of the remote agent log for a given job, you issue the following command on the machine where the job last ran:

```
autosyslog -J job_name
```

Note: If the configuration file specifies that the remote agent log files are to be cleaned up at the completion of a job, and the job completed normally, the file will have been removed. If the job failed for some reason, the file will not be deleted, regardless of the configuration file setting. To turn off automatic deletion of the remote agent log files, set the CleanTmpFiles parameter in the configuration file to 0.

For more information about the configuration file and the CleanTmpFiles parameter, see Configuration File Parameters in the chapter “Configuring,” in this guide.

Symptoms

1. The job is stuck in either the STARTING or RUNNING state as seen in either the event processor log or the output resulting from issuing the following command:

```
autorep -J job_name
```

2. The job has immediately returned as a FAILURE.

Resolution

If you have gotten to this point, the AutoRemoteDir/auto_rem* file should be present. By looking in this file, you will see how far Unicenter AutoSys JM was able to get. You should check and verify the following items:

1. The correct file is being sourced before running the job command. The default file is /etc/auto.profile. A job-specific file may be sourced instead of the default profile.
For more information, see the description of the profile attribute in the chapter “Job Attributes.”
2. The \$PATH variable is set up properly, and includes the proper location for all required executables. Variables, such as \$PATH, must be exported for the job to see them.
3. The file system that the job command is on is accessible from this machine.
4. The permissions are correct on the job command to be executed.
5. The permissions are correct on any standard input/output files specified for re-direction.
6. The profile is a Bourne shell script. (*Korn shell and C shell scripts will not work.*)

Note: A valuable debugging technique is to specify a file to be used for standard output and standard error for the job that will not run. If there are any shell level command problems, all error messages will be in that file.

Remote Agent Starts, Command Runs—No RUNNING Event Is Sent

Symptoms

1. Job is stuck in STARTING state.
2. This problem is detected either in the event processor window (or log) or in the results of running the autorep command on the job, when the following event appears, then nothing else, yet the job does run to completion on the client machine:

```
CHANGE_STATUS Status: STARTING Job: test_install
```

Resolution

This is a common problem and is nearly always the result of the remote agent being unable to contact the event server. First, ensure that network problems are not preventing communication between the remote agent and the event server machines. If this is not the problem, then check the following database-specific solutions.

With Sybase, this problem usually occurs because the interfaces file is not set up properly on the machine running the remote agent. With Oracle, this problem usually occurs because the SQL*Net V2 connections are not set up properly.

The remote agent must be able to connect to the event server in order to send the RUNNING, SUCCESS, FAILURE, or TERMINATED status events.

To verify the problem, look in the AutoRemoteDir/auto_rem* file for this job. You can accomplish this by issuing the following command on the machine where the job is supposed to have run:

```
autosyslog -J job_name
```

where:

job_name

Is the name of the job in question.

If the remote agent cannot send the event back to the database, it will write a message to that effect, plus some diagnostics, into this file. (The output from the autosyslog command could provide a helpful DBMS error number from the connect attempt.)

If you are using Sybase, check the following:

1. Check that the Sybase interfaces file exists and is readable by all users. The location of the interfaces file is pointed to by an entry in the /etc/auto.profile, which looks like this:

```
#AUTOENV#SYBASE=/usr/home/sybase
```

2. Check that the Sybase interfaces file has an entry for the data server that contains the AutoSys event server.

The format of the Sybase interfaces file requires that:

- Each data server name begins at the left margin with no preceding spaces or tabs.
- Each entry line has a single preceding tab.

Each element in an entry line is separated by a single space.

Incorrect formatting will cause the remote agent to be unable to communicate with the database.

If you are using Oracle, check for the following:

1. Check that the Oracle TNS names file, tnsnames.ora, exists, is readable, and contains the correct information for the event server. By default, the TNS names file is in one of the following locations:
 - On most systems, it is in /etc/tnsnames.ora
 - On some System V systems (for example, Solaris), it is in /var/opt/oracle/tnsnames.ora, or in /var/opt/tnsnames.ora

If the tnsnames.ora file is not in one of these locations, you must define it using the \$TNS_ADMIN environment variable. Set the \$TNS_ADMIN variable for the remote agent with an entry in /etc/auto.profile that looks like this:

```
#AUTOENV#TNS_ADMIN=/usr/home/oracle  
/etc/auto.profile must be readable by all users.
```

2. Check that the Oracle TNS names file has a SQL*Net V2 formatted entry for the event server.

Testing the Setup

To test that everything is set up properly, try to log onto the event server from the client machine, using the xql utility (for Sybase), or using sqlplus (for Oracle). When you log onto the event server, use the “autosys” user and password.

When testing Sybase using xql, be sure that your user environment is looking at the same interfaces file as the auto_remote (remote agent). Set SYBASE to the same value that is in /etc/auto.profile.

Note that the auto_remote only attempts to read the interfaces file once. After a bad interfaces file has been read, correcting it will not allow a running auto_remote to connect. After you correct the interfaces file, you will have to kill the auto_remote and restart the job.

For Sybase, try to log onto the event server from the remote machine using xql, like:

```
xql -U autosys -P autosys -S AUTOSYSDB
```

When testing Oracle using sqlplus, be sure that your user environment is looking at the same tnsnames.ora file as the auto_remote (remote agent). Set TNS_ADMIN to the same value that is in /etc/auto.profile.

Note that the auto_remote only attempts to read the tnsnames.ora file once. After a bad tnsnames.ora file has been read, correcting it will not allow a running auto_remote to connect. After you correct the tnsnames.ora file, you will have to kill the auto_remote and restart the job.

For Oracle, try to log onto the event server from the remote machine using sqlplus with a V2 connect descriptor, like:

```
sqlplus autotest/autotest@AUTOTESTDB
```

xql Will Not Start (Sybase Only)

Symptom

From the remote machine, xql returns the following message:

```
DB-Library error: dbproc NULL  
Error in SybInit: dbopen failed
```

Resolution

Check the following:

1. Determine if the data server is started and running, if not start the data server.
2. Verify that the DSQUERY environment is set to the proper data server.
or do the following:
Run xql with the -S and -D options to specify the correct data server and database.
3. If a fully-qualified xql statement still fails, then it is a problem with the interfaces file. For more information about dealing with this file, see the resolution in Remote Agent Starts, Command Runs—No RUNNING Event is Sent in this chapter.

Remote Agent Not Found

Symptom

When trying to start a job or trying to start the event processor with a shadow event processor, the following message appears in the event processor log when viewed with the autosyslog -e command:

```
Unknown Host Machine
```

Resolution

You may receive this message in the following situations:

1. There is a network problem and a connection cannot be made to the remote agent machine.
2. The remote agent machine is not in /etc/hosts or DNS.
3. The AutoSys configuration file lists the machine; however, there is a space after the machine name.

Check /etc/hosts or DNS for the machine name, and add it if necessary.

Check the configuration file, \$AUTOUSER/config.\$AUTOSERV (\$AUTOSERV is the name of the AutoSys instance). A space after the machine name is hard to see. Use an editor, such as vi (with the :set list option), to edit the configuration file and remove anything after the name of the machine and before the \$ that marks the end of the line.

Jobs Run Twice

If you receive failed to connect to socket errors during a job run, the job runs more than once. This is the expected behavior when this error occurs as explained by this scenario:

1. The server opens a connection to the client to run the job.
2. The remote agent on the client starts the job, and then tries to respond to the server.
3. The server issues a failed to connect to socket error because the remote agent took longer than 30 seconds (the timeout value) to start the job and respond.
4. The server checks if the job can be restarted, and then restarts the job. Meanwhile, the job is running and perhaps has completed on the client.
5. The server opens another connection to the client to run the job a second time.
6. The remote agent starts the job and responds to the server in time.
7. The job runs a second time.

Severe performance problems on the client are the main reason this occurs. For example, the following might affect performance:

- Running a full system backup on the client at the same time jobs are starting might slow down the system so that it cannot respond to the server.
- Network problems. If a job's home directory is on an NFS drive and there are bandwidth problems, the job might take so long to start that the socket times out.

Because socket time-out is not a customizable parameter, there is little you can do to avoid this situation from an Unicenter AutoSys JM perspective. However, you can analyze the performance of the client by asking these questions:

- Are there too many processes running on the client when you run jobs?
- Are you having network problems?
- Are you using NFS-mounted directories?
- Do you need more memory or processors on the client?

Job Failure Troubleshooting

Jobs Run Only From the Command Line

Symptom

Jobs run from the command line, but they fail when run.

Resolution

This problem is nearly always the shell environment where the job runs. The following are the possible reasons for the problem:

1. The profile in the job definition is not a Bourne shell (sh) type profile. If this is the case, the profile fails.
2. The default profile does not produce the proper environment for the job to run. The default profile for all jobs is /etc/auto.profile, not the job owner's logon profile \$HOME/.profile. If the job owner's profile is not specified in the job definition, it is never sourced.

To check the difference between the job definition and the user environment, do the following:

3. Write the current owner's environment to a file. Log in as the owner of the job on the machine where the job will run and enter the following command:
`env >user.env`
4. Write the remote agent environment to a file by entering the following JIL command:

```
insert_job: auto_env
machine: client_hostname
owner: owner
command: env
std_out_file: /tmp/auto.env
std_err_file: /tmp/auto.err
```

where:

client_hostname Is the hostname of the machine where the problem job runs.

owner Is the owner of the job that will not run.

5. Run the troubleshooting job by entering the following command:

```
sendevent -E STARTJOB -J auto_env
```

6. Check the two files for differences by entering the following command:

```
diff /tmp/auto.env user.env
```

This shows you where the environment and the user environment differ. Make the necessary changes in the job definition and the user profile.

Also, it is useful to define the std_err_file for the job that fails, because you can check the errors from the shell for a clue about what is missing.

Introducing CAICCI

CAICCI (Computer Associates International Common Communication Interface) is a transport layer that allows the asbIII process, which handles cross-platform events, to communicate with Agents on AS/400, OpenVMS, UNIX, Windows, and OS/390. On UNIX, CAICCI consists of several processes and a library. The processes are responsible for network transmission, creating and maintaining CAICCI resources. asbIII accesses the CAICCI API through the shared library.

On the UNIX platform, there are three demon processes:

- **caiccid**
This process is referred to as the main CAICCI demon because it is started first, builds the CAICCI resources and starts the other two CAICCI demons.
- **caiccilnd**
This process is referred to as the clean demon because its responsibility includes the maintenance of the CAICCI IPC resources.
- **caiccirmtd**
This is the remote demon process, which is responsible for the transmission of data across the network.

Installing CAICCI

The Unicenter AutoSys JM installer offers the option to install CAICCI with an AutoSys server. If you select Unicenter CCI, it installs CAICCI automatically. Later in the dialogs, the installer will ask for a list of remote hosts with which CAICCI is to communicate. Enter their names separated by spaces. You may change the list of remote hosts after installation by editing the file ccirmt.dprf as described below.

Configuring CAICCI

The asbIII process communicates to Agent machines and Connect using CAICCI. You need to configure CAICCI to communicate with a particular machine.

You must log in as root to edit the CAICCI configuration files:

- caiccid.prf
- ccirmtd.prf
- cciclnd.prf

Note: We do not recommend that you update the caiccid.prf configuration file unless the file hits the max_recvrs limit.

caiccid.prf

The caiccid.prf file tells the main CAICCI demons what to do and specifies the Max_Recvrs value.

Node-Specific Path Name

The caiccid.prf: file is found in the following location:

\$CAIGLBL0000/cci/config/nodename/caiccid.prf

where:

nodename Identifies the machine on which the Enterprise Management CAICCI demons are running.

The following is an example of the caiccid.prf file:

```
CLN_Demon = cciclnd startup
RMT_Demon = ccirmtd startup
Max_Recvrs = 48,32
```

CAICCI Demons

The following parameters specify what the CAICCI demons do and specifies the Max_Recvs parameter:

- CLN_Demon = ccilnd startup—This setting tells CAICCI to start the CAICCI clean demon when you start CAICCI.
- RMT_Demon = ccirmtd startup—This setting tells CAICCI to start the CAICCI remote demon when you start CAICCI.
- Max_Recvs = *nn,mm*—The value of *nn* defines the number of CAICCI receivers which also determines the size of the shared memory segment for RVT lists. The value of *mm* is the number of messages that CAICCI will queue up. These parameter values are explained in Shared Memory for RVTs, in this chapter.

Shared Memory for RVTs

Information about a receiver is kept in a structure called an RVT. Since all applications must have access to information about a receiver, the information is kept in shared memory as an RVT list. On UNIX, it may be viewed by using the CAICCI show command. This produces blocks of data. The first block contains global data and each subsequent block contains information specific to a particular receiver. This information includes the CAICCI address, the process ID of the application and whether the application is ready to accept data or has a message pending.

On UNIX, the process that creates shared memory is the main CAICCI demon, CAICCI. When CAICCI starts up it creates the shared memory segment; therefore, CAICCI must know beforehand how large a memory segment to create.

The value of *nn* determines the size of the shared memory segment because it is the number of RVTs CAICCID creates. This has the effect of limiting the number of application receivers. CAICCI is shipped with a default value *nn*=48. You may need to change the default value to match your installation.

Each application requires at least one RVT and each unprocessed CCI message requires another. On a busy server, you may need to increase the value of *nn*. An indication that you need to increase the value of *nn* is if you receive the CAICCI_E_FREERVTS error message. This message is displayed on the system log. On a busy server, you may need to increase the value of *nn* to 200 or 300.

After the value of *nn* is increased, asbIII may not start up because CAICCI has not started. This sometimes happens because CAICCI must protect access to this shared memory with the use of a semaphore group. CAICCI needs to create a semaphore group with a semaphore identifier for each RVT plus three extras. On most UNIX platforms, the number of semaphore identifiers in a semaphore group is governed by the SEMMSL kernel parameter. It is important to be aware of the following rule when increasing the first number of the Max_Recvrs parameter, *nn*:

```
SEMMSL >= nn + 3
```

You may need to increase the SEMMSL value.

CAICCI requires, at most, two distinct semaphore identifiers. If CAICCI gets ID=0, it will hold this group and request another. Unicenter® Network and Systems Management (Unicenter NSM) has requirements for XXXMNI, which should be added to requirements for other products using IPC (Inter-Process Communication) resources.

In addition to storing an application's CAICCI address, shared memory is also used to store the data that the application is sending. Sometimes messages arrive too quickly for the target application to dispose of them. In this case, an application may request that CAICCI queue up messages. The number of messages that CAICCI will queue up for each application is determined by the second value of the Max_Recv parameter—*mm*. The *mm* value may be set as high as 700; however, CAICCI is designed to shut down when the 700th buffer is filled to avoid problems with possible limited resources. For this reason, the recommended maximum value of *mm* is 699.

If the Max_Recv parameter is set to a value higher than allowed, it will default to the maximum. Sometimes an application will hang or will be too busy to pick up its messages. In a situation such as this, you will see the error message:

```
CAICCI_E_RECVBUSY Target [ ] queue is full, sender [ ]
```

The default behavior is for the sending application to sleep while waiting for room on the buffer. This may work for an application using CAICCI but not for the remote demon.

Customizing

The simplest way to enable CAICCI remote communications is to have the configuration requirements built for you during your initial installation.

If you have already installed CAICCI, you can use the \$CAIGLBL0000/cci/scripts/reinstall script to enable CAICCI remote communications.

1. During this procedure, the following prompt appears:

```
During the initial installation, several CCI parameter files were created.  
Do you want to recreate these files as part of this reinstallation?  
(y/n) (default: n)
```

Respond y to install the CAICCI remote demon.

2. You are then prompted for the names of the nodes with which the CAICCI remote demon is to establish communications, as follows:

```
Please enter the name of the remote host or RETURN to end:
```

Reply with a single node name.

3. The prompt is repeated in order for you to specify another node with which CAICCI will communicate. Respond with additional node names, one at a time, until all of the nodes have been specified.

When you are done, press Enter twice to continue.

ccirmtd.prf

The ccirmtd.prf file identifies the local CAICCI node name, the UNIX host name, and the block size for the local and remote machines.

Node-Specific Path Name

The ccirmtd.prf file is found in the following location:

\$CAIGLBL0000/cci/config/nodename/ccirmtd.prf

where:

nodename Identifies the machine on which the CAICCI demons are running.

Syntax

```
LOCAL = nodename cciname max_msg_size [startup | nostart] [port=1721 retry=n]
REMOTE = nodename cciname max_msg_size [startup | nostart] [port=1721 retry=n]
```

where:

nodename Indicates the hostname that will be passed to gethostbyname. This can be any name resolvable to the correct IP address and does not have to have logical connection to cciname.

cciname Indicates the logical name CAICCI will use to identify this host. This name is determined by the ca_uname function at install time and by ca_nodename during runtime. These functions are the equivalent of uname -n. This name can be as long as 64 characters but an alias must be used for names greater than eight characters.

max_message_size Specifies the maximum buffer that CAICCI will send or receive over the socket. It is a good idea not to adjust this. Each side of the connection may have this set to different values, up to 32KB. The lesser of the two values is used.

startup | nostart Tells ccirmtd whether or not to initiate a connection. Sometimes you may only want one side to initiate the connection. This is handy for a UNIX admin client when many people power down their PCs at night. You can eliminate many annoying messages when CAICCI is recycled during the night if the server does not start connections.

In addition, there are verbs of the form VERB=value:

- RETRY=*x*—Determines how ccirmtd will behave if the connection is dropped.

where *x* is one of the following:

0—ccirmtd will not retry the connection.

- 1—ccirmtd will start with a two second retry interval and double after each unsuccessful retry attempt.
- > 0—ccirmtd will wait n seconds between retry attempts.
- This is used in conjunction with the nostart option to allow the server to sit passively and wait for incoming connection requests. If a client host goes down the server will not attempt to reconnect, and we are again relieved of messages requesting that we check to see if CAICCI is active on the client.
- PORT=p—Allows us to specify an alternate port for this specific connection only. The default port number is 1721.

For example:

```
LOCAL = abcdef31 abcdef31 32768 startup
REMOTE = abcdef33 abcdef33 32768 startup
REMOTE = abcdef33 abcdef33 32768 startup port=7000
```

cciclnd.prf

The cciclnd.prf file defines the number of seconds to sleep between system scans for communications buffer and connections cleaning. The default time value for cciclnd is one second. The default value should not be changed unless instructed by Computer Associates Technical Support.

Node-Specific Path Name

The cciclnd.prf file is found in the following location:

```
$CAIGLBL0000/cci/config/nodename/cciclnd.prf
```

where:

nodename

Identifies the machine on which the CAICCI demons are running.

CAICCI Environment Variables

The following environment variables are available on the UNIX platform. The best place to set them is in the \$CAIGLBL0000/cci/scripts/rc file prior to the invocation of the main CAICCI demon process (caiccid), unless otherwise noted.

CAI_CCI_DEBUG

Definition:

Enables or disables CAICCI traces.

Use:

Set to enable traces, unset to disable traces.

Components Affected:

All CAICCI processes and applications using CAICCI.

When to Use:

Use for tracing.

CAI_CCI_LOG

Definition:

The directory to which CCI trace file is written.

Use:

CAI_CCI_LOG=path, where path is directory for traces.

Components Affected:

All CCI processes and applications using CCI.

When to Use:

Use if a larger volume is required for trace output.

CAI_CCI_CONFIG

Definition:

Sets path to CCI configuration directory.

Use:

CAI_CCI_CONFIG=path where configuration files reside:

\$path\ -n

Components Affected:

All CCI processes.

CAI_CCI_SHMMIN

Definition:

Minimum size of shared memory segment CAICCI requests.

Use:

CAI_CCI_SHMMIN=SHMMIN, where SHMMIN is a kernel parameter.

Components Affected:

All CAICCI processes and applications using CAICCI.

When to Use:

When the SHMMIN kernel parameter is greater than 1.

CAI_CCI_PORT1

Definition:

ccirmtd will bind to a specified port prior to connect calls.

Use:

`CAI_CCI_PORT1=n`

where:

n

$> 1024\text{ K}$

Components Affected:

Remote demon process.

When to Use:

For firewalls or certain multi NIC situations.

CCI_SELECT_TIME

Definition:

Determines the time out for select after connect.

Use:

`CCI_SELECT_TIME=n`

where:

n

> 1

Components Affected:

Remote demon process.

When to Use:

Sometimes network conditions will cause the TCP/IP handshake to take a long time to complete.

Starting and Stopping CAICCI

Note: You need to be logged in as root to start and stop CAICCI.

To start CAICCI, run the rc script:

```
$CAIGLBL0000/cci/scripts/rc
```

There should be three processes running for CAICCI:

```
$ ps -ef|grep cci
      root 17733 17731  0  Jan 18 ?          30:36 /uni/cci_kit/cci/bin/ccirmtd
      root 17732 17731  0  Jan 18 ?          1:20 /uni/cci_kit/cci/bin/cciclnd
      root 17731     1  0  Jan 18 ?          1:25 /uni/cci_kit/cci/bin/caiccid
```

To stop CAICCI, run the cshut script:

```
$CAIGLBL0000/cci/scripts/cshut
```

If CAICCI hangs and you cannot shut it down, do the following:

1. Kill the three CAICCI demon processes, enter:

```
kill -9 <3 pids of cci>
```

2. Search the shared memory for the caiccid process, enter:

```
ipcs -a|grep 0000d
```

3. Remove the message queue for the caiccid process returned from Step 2, enter:

```
ipcrm -q <message queue id>
```

4. Remove the shared memory for the caiccid process returned from Step 2, enter:

```
ipcrm -m <shared memory id>
```

5. Remove the semaphore for the caiccid process returned from Step 2, enter:

```
ipcrm -s <semaphore id>
```

Troubleshooting CCI

This appendix describes troubleshooting CCI for Unicenter AutoSys JM.

Troubleshooting Tools for Remote CCI Connections

The following are troubleshooting tools:

netstat

The netstat command allows you to check TCP/IP statistics:

- `netstat -a | grep caic`

Shows all connections to the local host involving a port, which can be resolved to caic(ci). The important connections are ESTABLISHED and LISTEN. If the latter is present, you know that the kernel accepts connections on behalf of the ccirmtd process. This means that a remote host attempting to connect to this host should get the TCP/IP connected state. Established connections are important because we know that CCI transactions may not transpire between the hosts in question if a TCP/IP Established connection does not exist.

It is important to understand that netstat output is of the form:

`ip-address:port`

where the local host is listed to the left of the remote host. One side will always have a port that resolves to caicci and the other side will have a numeric port. The latter side is that which initiated the connection.

Sometimes netstat –a does not return or may take a long time to return with very little information. This is usually indicative of name resolution problems. You can issue:

```
netstat -an | grep 1721
```

netstat skips the name resolution and displays information about connections.

- **netstat –i**

Shows information about the network interfaces on the local hosts. You can use the netstat –i command to determine if the host has more than one network card and determine the hostnames or IP addresses of these cards. The netstat –i command also provides valuable statistics about network collisions. A collision occurs when two hosts simultaneously attempt to send on an ethernet. The important thing to look for is a high ratio of outgoing or incoming packets to collisions.

ping

ping allows you to establish that a remote host can be reached. It is important to ping by IP address as well as host name. If you cannot ping a host, CCI cannot establish a connection to that host.

nslookup

nslookup allows you to be sure that the name of the host to which you wish to connect, as well as the IP address, is resolvable. If there is a question as to the integrity of the DNS environment, you can use nslookup to verify the IP address of the host to which you need to communicate. You then enter the IP address back into nslookup and verify that the same host name is returned. Verify the IP address and hostname for both hosts.

traceroute

The traceroute command on UNIX and the tracert command on Windows allow you to determine the route taken between two hosts. If a client cannot ping a host, this command may show where the network path is failing.

ccinet

ccinet may be used to pass commands to the ccirmtd demon on UNIX. On Windows, this is the rmtcntrl binary. This may be used as follows:

- ccinet ping

Can be used to send a special CCI test packet across the CCI connection. This does not use the native ping command nor does it operate in quite the same way.

- ccinet status

Allows you to determine the status of the CCI connections.

- ccir/ccis/ccic/ccii

Provides you a suite of test binaries to test CCI communication. (ccinet ping tests remote process to remote process communication)

CCI Command Line Controls

Commands may be passed to CCI processes. Executables, therefore, may gather diagnostic information from these processes.

On UNIX platforms the command binary for the main CCI demon is:

```
$CAIGLBL0000/cci/bin/cci
```

The following commands are available:

- cci show
- cci semashow and cci semaclear X
- cci shutdown
- cci debugon and cci debugoff

cci show

This allows you to view the shared memory segment where CCI stores the RVT list. This is useful to determine general CCI information, such as:

- The number of free and active RVTs
- The key used to create the CCI resources
- The identifiers for the CCI resources
- The process ID's of the CCI demons
- The time the shared memory was created

You can also use this command to display information about a specific receiver, such as:

- The existence of a specific receiver
- The number of pending messages for a specific receiver
- The PID of the process that created the receiver
- The PID of the process that holds the semaphore for this receiver
- The last send and receive time
- The number of sends and receives

cci semashow and cci semaclear X

You can use the semashow command to determine if any of the CCI semaphores are being held. This is useful information for conditions when Unicenter AutoSys JM is hanging. When a problem exists, the output of the command will be two or more lines:

```
CAICCI_I_0003 i[X] sema[YYYY] pid[Z]  
CAICCI_S_0046 Command completed successfully
```

When there is not a problem with the CCI semaphores, only the last line is returned.

where:

X The particular semaphore identifier in the CCI semaphore group.

YYYY The semaphore group.

Z The process ID of the process holding the semaphore.

To use this to troubleshoot a hanging condition, execute the command and note the process ID's of those processes holding a CCI semaphore. It is always a good idea to issue ps -ef in conjunction with this command. If the process holding the semaphore is defunct, the group responsible for support of this application should be contacted because CCI does not release resources held by defunct processes.

Next, you issue the following for each held semaphore in the semashow output:

```
cci semaclear X
```

The semaclear command releases the semaphore and can allow Unicenter AutoSys JM to continue normal operations.

cci shutdown

Tells the main demon to shut down. The use of this command is not advised if Unicenter AutoSys JM is still running.

cci debugon and cci debugoff

Will turn main demon tracing on and off.

The binary used to pass commands to the CCI remote demon is:

```
$CAIGLBL0000/cci/bin/rmt
```

The following commands available are:

- **ccinet show**

This command will output data concerning the hosts to which the remote demon is or should be connected. It will also output information about the receivers available on those remote platforms similar to the RVT information displayed by 'cci show.' The output from this command is written to the ccirmtd trace file. Therefore, to capture this output we prefer that traces have been enabled prior to its execution. This data is also output to the system console. The output from this command is usually important for solving all issues involving remote communication.

- **ccinet debugon and ccinet debugoff**

Used to enable or disable remote traces without recycling the remote demon.

Trace data is written to:

```
$CAIGLBL0000/cci/logs/ccirmtd_<pid>.log
```

- **ccinet status**

Will display information concerning the remote hosts to which the remote demon is connected or to which it should be connected. This data is displayed in tabular form on stdout. If you are receiving a "no receiver online" type error, check this output as it may show that you are not connected to the host in question.

- **ccinet release**

The release of the ccirmtd is displayed to stdout.

Note: The cci 666 command is no longer supported in NSM.

The command gives the release as follows:

xxxxyyzzzz

where:

xxx Is the source code version (for example, 1.137)

yy Is the genlevel of Unicenter (for example, 21 for TNG 2.1)

zzzz Is the release of Unicenter (for example, 9708)

- **ccinet disconnect *sysid***

Causes the local remote to issue a disconnect command to the specified sysid and close down the connection. This has the effect of severing the connection between these hosts. Neither side attempts to reestablish the connection.

- **ccinet reconnect *sysid***

If hosts are connected, this causes them to disconnect and then reconnect. If hosts are not connected, the local remote demon will attempt to connect to the remote host.

- **ccinet ping *sysid***

A useful diagnostic tool which causes the local remote demon to send a special CCI packet to which the other host responds. This command allows you to determine if the CCI connection is useful at the most basic level. Upon successful completion, the roundtrip time is displayed.

- **ccinet echo *sysid message***

If successful, the message is displayed on the target systems console.
Another useful tool for determining how well the CCI connection between two hosts is functioning.

- **ccinet retry *sysid N***

Will affect the retry time interval as follows:

- Set the retry time interval to *N*, if *N*>0
- Set the retry time interval to 2, then double on each successive failure, if *N*= -1
- Prevent the local host from attempting to reconnect if *N* = 0

The commands are passed to the CCI demon processes using the message queue facility. Therefore, if there is a problem with these facilities the commands may not function correctly.

Reinstalling CCI

If you need to reinstall CCI, for any reason, reinstall Unicenter AutoSys JM. When the installation asks you if you want to install CCI, answer yes.

Note: Before you reinstall CCI, unset CAIGLBL0000. Log in as root at a UNIX prompt and enter:

```
unset CAIGLBL0000
```

For more information about CCI installation, see the appendix “Introducing CCI” in the *Unicenter AutoSys Job Management for UNIX Installation Guide*.

General Debugging

To better monitor the behavior of the Unicenter AutoSys JM product and to facilitate in identifying problems while troubleshooting, a Unicenter AutoSys JM environment variable has been introduced called ISDBGACTIV. (For UNIX, ISDBGACTIV is an OS environment variable. For Windows it is a registry key). The environment variable must be set prior to initiating the product. Upon startup, the traceable applications will look for the set value of the ISDBGACTIV environment variable and will output certain trace messages given the value assigned. In UNIX, the ISDBGACTIV environment variable can be set as any other environment variable using either the setenv or export command depending on the UNIX operating system used. In Windows, the ISDBGACTIV environment variable must be set using the Administrator Tool through the System Environment Variable screen.

The following is a description of how the products interpret the ISDBGACTIV values:

ISDBGACTIV Value	ID	Description
1	DBG	Generic Debug Status Information
2	CDB	CCI Memory Output
4	MDB	Internal Communications Data Information (sockets, message queues)
8	OPX	Internal Communications Status Information
16	EDB	Event Processor Debug Messages
32	DDB	Database
64	RDB	Event Processor to Remote Agent Communications Information
128	SDB	Sendevent Status Information
256	JDB	Job Start Status Information

The individual values can be combined to control the number of traces generated. For example, to view Event Processor Debug Messages and Generic Debug Status, ISDBGACTIV should be set to $32 + 1 = 33$. To view all traces, ISDBGACTIV should be set to 63 ($32+16+8+4+2+1$). In general, DBG, MDB, and DDB generate large amounts of trace statements whereas OPX and EDB provide light traces.

The products that generate trace messages are the Event Processor, the AutoSys Broker, the AutoSys Broker CCI Send and Receive Objects, and the Remote Agent. Each of these products generate their own log files under normal circumstances located in the \$AUTOSERV/autouser/out directory. Any trace messages will be added to these log files at various places as the products encounter them.

Unicenter Integration

Before enabling Unicenter event integration, you must install the event manager agent on the event processor machine. The Event Agent can be installed alone, as part of Unicenter Framework, or as part of the entire Unicenter product.

To integrate Unicenter AutoSys JM for UNIX with Unicenter do the following:

1. Login as the Unicenter AutoSys JM user (requires privileges to update the configuration file).
2. The configuration file has the following name:

`$AUTOUSER/config.$AUTOSERV`

The example entry in the configuration file looks like:

`UnicenterEvents=0`

The valid values for the Unicenter Events are:

0—No events are sent.

1—Only the Alarms are sent.

2—Alarms and Job Completion status are sent.

3—All the events are sent.

3. Change the entry value with the integer value (0–3) that corresponds to the desired message level in the previous list.
4. You must stop and restart the Event Processor before these changes will take affect.

For more information about the configuration file, see the Chapter Configuration, in the *Unicenter AutoSys Job Management for UNIX User Guide*.

Index

\$

\$AUTORUN, 3-26
\$AUTOSYS/bin/chk_auto_up, 12-31
\$AUTOSYS/code/heartbeat.c, 4-24
\$AUTOSYS/code/heartbeat.sh, 4-24
\$AUTOSYS/dbobj/create_table, 12-31
\$AUTOTESTMODE, 12-9
\$AUTOUSER/autosys.env, 12-29
\$AUTOUSER/config.\$AUTOSERV, 13-1
\$SYBASE, 12-29
\$SYBASE/install/RUN_AUTOSYSDB, 12-29

/

/bin/date command, 12-9
/etc/.autostuff file, 13-29, 13-30
/etc/auto.profile file. See auto.profile file, 13-25
/etc/services file
 remote agent port number, 13-27
/tmp/autotest.\$JobName, 12-9

A

About asbIII, A-9
Adding Machines - JIL, 7-9
advanced configuration, 13-1
after_time report attribute, 11-8
alarm callbacks, 13-32
Alarm Manager
 about, 10-1
 acknowledging alarms, 10-23
 Alarm List, 10-23
 Alarm Selection dialog, 10-29
 Select by State region, 10-30
 Select by Time region, 10-31
 Select by Type region, 10-30
 changing alarm states, 10-28
 closing alarms, 10-23
 Control Region
 Freeze Frame button, 10-26
 New Alarm button, 10-27
 Select Job button, 10-27
 Currently Selected Alarm
 acknowledging, 10-26
 closing, 10-26
 Response edit box, 10-26
 menu bar, 10-24
 registering responses, 10-28
alarm monitor / report attribute, 11-6
alarm_if_fail job attribute, 4-15

alarm_verif monitor attribute, 11-10

alarms

- about, 1-10
- DB_PROBLEM, 13-32
- DB_ROLLOVER, 13-32
- EP_HIGH_AVAIL, 13-32
- EP_ROLLOVER, 13-32
- EP_SHUTDOWN, 13-32

all_events monitor / report attribute, 11-6

all_status monitor / report attribute, 11-7

asset level security, 2-21

atomic starting conditions, 10-8

attributes

- job
 - alarm_if_fail, 4-15
 - auto_delete, 4-17
 - auto_hold, 4-17
 - avg_runtime, 4-24
 - basic, 3-6
 - box_failure, 4-28
 - box_name, 4-13
 - box_success, 4-27
 - box_terminator, 4-15
 - chk_files, 4-25, 4-27
 - command, 4-4, 4-5
 - date_conditions, 4-9
 - days_of_week, 4-9
 - description, 4-12
 - essential
 - all jobs, 4-4
 - Box Jobs, 4-8
 - Command Jobs, 4-5
 - File Watcher Jobs, 4-8
 - heartbeat_interval, 4-24
 - job_load, 4-21
 - job_name, 4-4
 - job_terminator, 4-15
 - machine, 4-7, 4-8
 - max_exit_success, 4-23
 - max_run_alarm, 4-14
 - min_run_alarm, 4-13
 - n_retrys, 4-16

optional

- Box Jobs, 4-27
- Command Jobs, 4-19
- File Watcher Jobs, 4-26
- non-starting parameters, 4-12
- starting parameters, 4-9

- override_job, 4-22
- permission, 4-18
- priority, 4-22, 9-4
- run_calendar, 4-10
- std_err_file, 4-21
- std_in_file, 4-20
- term_run_time, 4-14
- timezone, 4-16
- watch_file, 4-8
- watch_file_min_size, 4-26
- watch_interval, 4-26

job dependencies, 3-16

machine

- factor attribute, 9-4
- max_load, 9-3

monitor

- alarm_verif, 11-10
- sound, 11-9

monitor / report

- alarm, 11-6
- all_events, 11-6
- all_status, 11-7
- essential, 11-5
- job_filter, 11-7
- mode, 11-5
- name, 11-5

report

- after_time, 11-8
- currun, 11-8

starting conditions, 3-16

authentication

- remote, 13-29

auto.profile file

- AutoMachWideAppend variable, 13-22
- DENY_ACCESS, 13-31
- remote agent settings, 13-25
- remoteProFiles, 13-16

auto_delete, 4-17

auto_remote. See, 13-25

autocal, 8-3

autocons, 10-2

autohold job attribute, 4-17

AutoInstWideAppend, 13-22

AutoMachWideAppend variable, 13-22

autoping, 14-12

AutoRemoteDir, 13-15

AutoRemPort, 13-20, 13-27

autosc, 11-4

AUTOSERV environment variable, 13-1

autostuff file, 13-29, 13-30

AutoSys

- components, 1-3
- database
 - defined, 1-3
- Graphical User Interface
 - see GUI, 1-2
- instances
 - defined, 1-8
- machines, 1-8
- security, 2-1

AutoSys Agent

- software requirements, A-4

AutoSys Agent support, A-1

AutoSys Connect and AutoSys Agent support, A-1

AutoSys/Xpert, 4-24

autosys_secure, 4-18

AutoSysAgentSupport, 13-21

- Cross-Platform Scheduling, 13-21

AutoSysAgentSupport

- Configure the AutoSys Machine, A-5

AutoSysAgentSupport Parameter, A-5

AutoSysAgentSupportReceiveSubmit

Configure the AutoSys Machine, A-6

AutoSysAgentSupportReceiveSubmit Parameter, A-6

autosyslog command, 12-4, 14-17

avg_runtime, 4-24

B

backups

- bundled Sybase, 12-33
- calendar definitions, 12-12
- global variables (using autorep), 12-13
- job definitions (using autorep), 12-12
- machine definitions (using autorep), 12-13
- monitor and browser definitions (using monbro), 12-13

batch files and exit codes, 3-24

Bi-Directional Scheduling, A-11

Box Jobs, 3-4, 5-1

- basic job definition, 3-7
- default behavior, 5-1
- diagram, 3-12
- examples, 5-9
- force starting jobs in a box, 5-7
- guidelines, 5-2
- non-default terminators, 5-5
- placing job in
 - GUI, 6-15
- placing job in
 - JIL, 7-10
- starting conditions, 3-4, 3-14
- status changes, 5-8

box_failure, 4-28

box_name, 4-13

box_success, 4-27

box_terminator, 4-15

browsers

- backing up definitions, 12-13
- defined, 11-1

C

- restoring definitions from backup file, 12-14
- Calendar Selection dialog, 8-13
- calendars, 8-1
- backing up definitions, 12-12
 - blocked dates, 8-9
 - blocking dates, 8-16
 - Calendar Definition window, 8-4
 - color key, 8-11
 - combining, 8-23
 - conflicting dates, 8-9, 8-11
 - creating
 - example, 8-12
 - custom, 3-16
 - customizing Calendar Facility, 8-28
 - date range, 8-16
 - date states, 8-9
 - Edit menu, 8-6
 - exporting, 8-25, 8-27
 - exporting definitions to file, 12-12
 - File menu, 8-5
 - importing, 8-25
 - importing definitions from file, 12-14
 - Job Definition Reference List, 8-7
 - merging, 8-23
 - Options menu, 8-8
 - printing, 8-24
 - rescheduling rules, 8-20
 - restoring definitions, 12-14
 - rule specification, 8-15
 - selecting, 8-13
 - setting dates, 8-16
 - Term Calendar Rule, 8-14
 - Term Calendar Viewer, 8-7, 8-22
 - Tools menu, 8-7
 - unsetting dates, 8-16
- CCI
- command line controls, B-4
 - reinstalling, B-8
 - troubleshooting tools, B-1
- ccinet, B-3
- charge back reporting, 9-15
- chase command, 12-10
- check file space, 4-25, 4-27
- Check_Heartbeat, 13-14
- chk_auto_up, 13-9
- chk_files, 4-25
- clean_files, 12-11, 13-15
- CleanTmpFiles, 13-15
- client machine, 1-8
- command job attribute, 4-4, 4-5
- Command Jobs, 3-3, 3-6
- command line controls
- CCI, B-4
- components
- Event Processor, 1-4
 - Event Server, 1-3
 - Remote Agent, 1-5
- conditions
- starting, 3-16
- config.EXTERNAL
- creating the file, A-6
- configuration file, 13-2
- AutoInstWideAppend, 13-22
 - AutoRemoteDir, 13-15
 - AutoRemPort, 13-20
 - Check_Heartbeat, 13-14
 - CleanTmpFiles, 13-15
 - DBEventReconnect, 13-8
 - DBLibWaitTime, 13-5
 - DBMaintCmd, 13-12
 - EDErrTimeInt, 13-9
 - EDMachines, 13-9
 - EDNumErrors, 13-9
 - EvtTransferWaitTime, 13-13
 - FileSystemThreshold, 13-11
 - InetdSleepTime, 13-23

KillSignals, 13-19
MachineMethod, 13-18
MaxRestartTrys, 13-17
MaxRestartWait, 13-18
RemoteProFiles, 13-16
RestartConstant, 13-18
RestartFactor, 13-18
sample, 13-2
ThirdMachine, 13-10
WaitTime, 13-18
XInstanceDBDropTime, 13-6

contacting technical support, 1-14

Control Panel
 GUI, 6-1

controlling the event processor, 2-21

cpu
 using available cycles to select machine to run on, 9-9

crash recovery
 database, 12-22, 12-39

cross-instance
 database connection, 13-6

Cross-Platform Scheduling, 13-21

Currently Selected Job region, 10-6

currun report attribute, 11-8

custom calendars
 overview, 3-16

customizing
 Calendar Facility, 8-28
 Job Definition, 6-25
 Operator Console, 10-32, 11-20

D

data locking, 12-21

database
 accessing interactively, 12-31
 administration, 12-27

 architecture, 12-17
 backup
 bundled Sybase, 12-33
 changing the, 12-28
 checking if up, 12-29
 connection to Job Definition GUI time-out interval, 6-25
 connection to Monitor/Browser GUI time-out interval, 11-21
 connections, 13-7
 crash recovery, 12-22
 data locking, 12-21
 defined, 12-15
 defining which to use, 12-17
 dumping, 12-33
 identifying connected processes, 12-32
 maintenance, 13-12
 maintenance script, 13-12
 maintenance time, 12-18
 passwords, 2-9
 recovery, 12-22, 12-39
 rollover, 12-22
 shutdown, 12-30
 starting, 12-29
 stopping, 12-30
 stopping service, 12-30
 storage requirements, 12-16
 time-out period, 13-5
 unrecoverable error, 12-22
 verifying connection, 14-12

 database field verification, 2-7

 dataserver
 defined, 1-3

 date dependency, 3-15

 date range in calendars, 8-16

 date/time job dependencies, 3-15

 Date/Time Options dialog
 example, 6-5

 date_conditions, 4-9

 days to run job
 setting

GUI, 6-19
JIL, 7-11

days_of_week, 4-9

DB Library, 12-26

DB_PROBLEM, 13-32

DB_ROLLOVER, 13-32

DBDropTime
 Job Definition GUI, 6-25
 Monitor/Browser GUI, 11-21

DBEventReconnect, 13-8

DBLibWaitTime, 13-5

DBMaint script, 12-19

DBMaintCmd, 13-12

dbstatistics script, 12-19

default owner of job, 2-10

delete job
 GUI, 6-21
 JIL, 7-13

DENY_ACCESS AUTOENV setting, 13-31

dependency
 job
 date/time, 3-15
 exit code, 3-23
 global variables, 3-25
 job status, 3-16

dependent jobs
 creating - GUI, 6-13
 creating -JIL, 7-7
 example, 3-20

description job attribute, 4-12

disable security, 2-20

DSQUERY variable, 12-27

dual event servers
 crash recovery, 12-22
 defined, 1-4
 mode, 12-15

synchronizing event servers, 12-23

dual Server Mode, 12-22

E

eAC, 2-17

EDErrTimeInt, 13-9

edit permissions, 2-11

Edit Superuser, 2-14

EDMachines, 13-9

EDNumErrors, 13-9

environment variables
 AUTOSERV, 13-1
 DSQUERY, 12-27
 See also profiles, 4-6
 SYBASE, 12-27
 user-defined, 4-6

Environment Variables for asbIII, A-10

EP_HIGH_AVAIL, 13-32

EP_ROLLOVER, 13-32

EP_SHUTDOWN, 13-32

errors
 event processor handling, 13-9
 event processor time interval, 13-9

essential job attributes, 4-4

eTrust
 Access Control, 2-17
 access modes, 2-23
 as-calendar class, 2-26
 as-control class, 2-30
 as-gvar class, 2-28
 as-job class, 2-24
 as-list class, 2-31
 as-machine class, 2-27
 as-owner class, 2-29
 as-view class, 2-30
 resource classes, 2-22

Security Administration, 2-18
security call logic, 2-35
security enabled applications, 2-33

event processor
allowable time between errors, 13-9
authentication on remote agent, 13-29
automatic shutdown, 13-8, 13-9
checking for running, 13-9
error handling, 13-9
heartbeat interval, 13-14
log
minimum disk space, 13-11
viewing, 12-3
maintaining, 12-1
monitoring, 12-2, 12-3
restoring, 12-7
running in test mode, 12-8
shutdown
automatic, 13-8
starting, 12-1, A-8
stopping, 12-4, A-5
tail command, 12-2
third machine, 13-10
troubleshooting, 14-10

Event Processor
defined, 1-4
See also event_demon, 1-4

Event Report
from Operator Console, 10-8

event server
dual, 12-15, 12-17
recovery, 12-22
synchronizing process, 12-23
transferring events, 13-12
troubleshooting, 14-2

Event Server
defined, 1-3
rollover, 12-22
See also database, 1-3
See also Dual Event Servers, 1-3

event_demon, 12-1
See also Event Processor, 1-4

eventor
script, 12-2

events
about, 1-9
cancelling, 10-12
copying between event servers, 13-13
EvtTransferWaitTime, 13-13

examples
calendar
creating, 8-12
calendar rescheduling rule, 8-21
individual queues, 9-19
JIL, 7-16
job dependencies, 3-20
load balancing, 9-10
monitor / report definition, 11-13
multiple machine queues, 9-20
queuing with priority, 9-18
real machine definition, 9-6
reports
defining in JIL, 11-19
system architecture, 1-6
virtual machine definition, 9-7
xql scripts, 12-31

exclusive condition, 3-21

exec superuser, 2-15

Exec Superuser, 2-15

execute permissions, 2-11

exit codes
batch files
with, 3-24
FALSE.EXE, 3-25
job dependencies, 3-23
maximum for success, 3-17

exitcode, 3-23

exporting calendars, 8-27, 12-12

F

factor attribute, 9-4
FALSE.EXE, 3-25
file locking, 13-15
file maintenance, 13-15
File Watcher Jobs, 3-5
 basic job definition, 3-6
 creating, 6-9
FileSystemThreshold, 13-11
filter
 monitor / report, 11-7
force starting jobs
 from Job Activity console, 10-9
 impact on load balancing, 9-11

G

gid, 2-11, 4-18
global variables
 backing up definitions, 12-13
 job dependencies, 3-25
 restoring definitions, 12-14
 setting
 in the Send Event dialog, 10-11
Graphical Calendar Facility, 8-1
 Calendar Definition window, 8-4
 screens, 8-3
 See also calendars, 8-3
 starting, 8-3
group ID, 2-11, 4-18
GUI

 Advanced Features dialog, 6-6
 Control Panel, 6-1
 Date/Time Options dialog, 6-5
 defined, 1-2
 defining monitor / reports, 11-11
 Job Definition dialog, 6-3

Monitor/Browser dialog, 6-2
one-time job overrides, 6-22
Operator Console, 6-2
starting, 6-1
time dependencies
 setting, 6-19
using to create a job definition, 4-2

H

heartbeat_interval, 4-24
heartbeats
 about, 13-13
 checking, 13-13
 code to include, 4-24
 time interval, 13-14
high availability
 dual event servers, 1-4
 shadow event processor, 1-5

I

Import/Export File Name (calendar) dialog, 8-25
importing calendars, 8-25, 12-14
inetd
 job starting interval, 13-23
 resetting, 13-19
InetdSleepTime, 13-23
inherit
 job's starting conditions, 3-14
insert_machine - JIL, 7-9
instances of AutoSys, 1-8

J

JIL
 creating a job definition, 4-2

defined, 1-2
defining jobs, 7-1
defining monitors/reports, 11-17
defining report, 11-19
example, 7-16
running, 7-4
sub-commands, 7-3
syntax rules, 7-1

Job Activity Console, 10-3
 Alarm button, 10-15
 Alarm Manager dialog, 10-23
 cancelling a sent event, 10-12
 Control Area, 10-9
 action buttons, 10-9
 control buttons, 10-13
 Dependent Jobs button, 10-13
 Freeze Frame button, 10-14
 Job Definition button, 10-13
 Report buttons, 10-14
 Currently Selected Job region, 10-6
 Dependent Jobs dialog, 10-14
 Job List, 10-4
 Job Path (History) dialog, 10-15
 Job Selection dialog, 10-17
 menu bar, 10-4
 reports, 10-8
 resizing regions, 10-16
 See also Operator Console, 10-3
 Send Event dialog, 10-10
 AUTOSERV instance, 10-11
 Cancel button, 10-11
 Change Status, 10-11
 Comment field, 10-11
 Execute button, 10-11
 Global Name/Value, 10-11
 Job Name, 10-10
 Queue Priority, 10-11
 Send Priority, 10-11
 Signal, 10-11
 time of event, 10-11
 starting conditions, 10-7

Job Definition dialog, 6-3
 customizing, 6-25
 database connection time-out, 6-25

icon text, 6-26
title bar text, 6-26

job definition encryption, 2-7

Job Definition Reference List, 8-7

Job Information Language, 1-10
 See also JIL, 1-10

job level security, 2-10

Job List region, 10-4

job overrides
 setting, 7-14, 7-15

job resource usage, 9-15

Job Selection dialog, 10-17
 Box Levels field, 10-18
 Job Name field, 10-18
 setting job selection criteria, 10-21
 sorting specified jobs, 10-21
 specifying jobs
 by machine, 10-19
 by name, 10-18
 by status, 10-19

job starts
 reducing start interval, 13-23

job status
 as job dependency, 3-16

job_filter monitor/report attribute, 11-7

job_load, 4-21

job_name, 4-4

job_terminator, 4-15

jobs
 attributes
 basic, 3-6
 backing up definitions, 12-12
 basic job information, 3-3
 Box Jobs, 5-1
 creating - GUI, 6-15
 creating - JIL, 7-8
 defined, 3-4
 changing - GUI, 6-17

changing - JIL, 7-10
Command Jobs
 creating - GUI, 6-7
 creating - JIL, 7-5
 defined, 3-3
creating
 Box Jobs - GUI, 6-15
 Box Jobs - JIL, 7-8
 Command Jobs - GUI, 6-7
 Command Jobs - JIL, 7-5
 File Watcher Jobs - GUI, 6-9
 File Watcher Jobs - JIL, 7-6
creating a job definition, 4-2
custom calendars, 3-16
cycles of processing, 14-1
date/time dependencies
 setting in GUI, 6-19
 setting in JIL, 7-11
days to run
 setting, 6-19
defined, 1-2
defining environment for, 4-6
defining in AutoSys, 3-27
definition
 Box, 3-7
 Command, 3-6
 File Watcher, 3-6
deleting
 GUI, 6-21
 JIL, 7-13
dependencies
 creating with GUI, 6-13
 creating with JIL, 7-7
 exit codes, 3-23
 global variables, 3-25
 job status, 3-16
 time
 GUI, 6-19
 JIL, 7-11
edit permissions, 2-11
execute permissions, 2-11
exit code, 3-23
File Watcher
 creating - GUI, 6-9
 creating - JIL, 7-6
 defined, 3-5
 force starting, 9-11, 10-9
 heartbeats from, 4-24
 number of restart attempts, 13-17
 overrides
 GUI, 6-22
 JIL, 7-14
 owner
 default, 2-10
 permissions on NT, 2-13
 queuing, 9-4, 9-15
 restart tries, 13-17
 restoring definitions from backup file, 12-14
 run number, 3-26
 saving definitions to backup file, 12-12
 starting conditions, 3-14
 states, 3-8
 status
 managing, 3-22
 status codes, 3-8
 time dependencies
 setting
 JIL, 7-11
 time/date dependencies, 3-15
 types, 3-2
 variables for, 4-6

K

KILLJOB signals, 13-19

KillSignals, 13-19

L

load balancing, 9-9
 example, 9-10
 force starting jobs, 9-11
 job attributes required for, 9-3
 maximum load on machine, 9-3
 method, 13-18
 user-defined, 9-21

locking

remote agent log file, 13-15

M

machine

attribute, 4-7, 4-8

backing up definitions, 12-13

client, 1-8

defining with JIL, 9-2

deleting

real machines, 9-6

virtual machines, 9-8

factor attribute, 9-4

job runs on, 4-7

max_load attribute, 9-3

permissions

edit and execute, 2-12

priority job attribute, 9-4

real, 9-1

restoring definitions from backup file, 12-14

saving definitions to backup file, 12-13

server, 1-8

virtual, 9-2

defining, 9-7

deleting, 9-8

MachineMethod, 13-18

maintenance

backing up AutoSys definitions, 12-12

calendar definitions, 12-12

global variables, 12-13

job definitions, 12-12

machine definitions, 12-13

monitor and browser definitions, 12-13

chase command, 12-10

clean_files, 12-11

commands, 12-10

database, 13-12

database time, 12-18

files, 13-15

restoring AutoSys definitions, 12-14

managing job status, 3-22

max_exit_success, 4-23

max_load machine attribute, 9-3

max_run_alarm, 4-14

maximum exit code for success, 3-17

maximum system load, 9-3

MaxRestartTrys, 13-17

MaxRestartWait, 13-18

method of load balancing, 13-18

min_run_alarm, 4-13

mode monitor/report attribute, 11-5

Monitor/Browser dialog, 6-2, 11-11

customizing, 11-20

database connection time-out, 11-21

icon text, 11-21

title bar text, 11-21

monitors, 11-1

about, 11-2

alarm_verif, 11-10

backing up definitions, 12-13

defined, 11-1

defining, 11-3, 11-13

GUI, 11-11

JIL, 11-17

filtering, 11-6

filtering events, 11-2

job_filter, 11-7

name, 11-5

restoring definitions from backup file, 12-14

sound, 11-9

status events, 11-7

multiple machine queues, 9-20

N

n_retrys, 4-16

name monitor/report attribute, 11-5

netstat, B-1

-
- NIS
troubleshooting, 14-14
- notification
user-defined notification routines, 13-32
- nslookup, B-2
- ntrys, 3-26
- SQL*Net V2, 12-15
- override_job, 4-22
- overrides
job
GUI, 6-22
JIL, 7-14
- owner
default, 2-10
-
- O**
- ON_HOLD vs. ON_ICE, 3-9
- one-time job overrides, 6-22
- Open Client C Library, 12-15
- Operator Console, 6-2
about, 10-1
customizing, 10-32
 Alarm List column width, 10-37
 alarm poll time interval, 10-33
 atomic conditions fields, 10-36
 background color of variable fields, 10-35
 border colors, 10-35
 changing fonts, 10-33
 currently selected Job Name field, 10-35
 Default Report type, 10-37
 font selection, 10-34
 freeze frame at start up, 10-34
 icon text, 10-38
 Job List column widths, 10-36
 label font, 10-34
 list font, 10-34
 object color, 10-35
 Operator Console size, 10-37
 primary interface colors, 10-36
 refresh time interval, 10-33, 10-36
 title bar text, 10-38
- Job Activity Console, 10-3
See also Job Activity Console, 10-1
starting, 10-2
- optional job attributes, 4-9
- Oracle
improving database performance, 12-25
-
- P**
- passwords
autosys user, 2-9
database, 2-9
database system administrator, 12-28
- permission attribute, 4-18
- permissions, 4-18
edit, 2-11, 4-18
execute, 2-11, 4-18
granting, 2-12
machine, 2-12
types, 2-11
user, 2-10
using umask, 2-10
Windows NT, 2-13
- ping, B-2
- policy manager, 2-19
- port number for remote agent, 13-20
- priority job attribute, 4-22, 9-4
- profiles, 4-6
-
- Q**
- queuing
and simple load limiting, 9-15
as subset of virtual machine, 9-19
jobs, 9-15

-
- multiple machine, 9-20
policies, 9-15
with priority, 9-17, 9-18
- R**
- real machines, 9-1
defining, 9-2
deleting, 9-6
example, 9-6
- reconnecting to database, 13-7
- recovery
database, 12-39
Sybase database, 12-39
- remote agent
auto_remote service, 13-28
database connectivity, 14-19
heartbeat interval, 13-14
log, 14-17
modifying settings for, 13-27
port number, 13-20
security
DENY_ACCESS, 13-31
settings, 13-27
socket connection, 13-27
troubleshooting, 14-12
- Remote Agent
defined, 1-5
Event Processor authentication, 2-9
security, 2-17
user authentication, 2-8
- remote agent event processor authentication, 13-29
- remote agent log
cleanup, 13-15
name assigned, 14-16
specifying directory for file, 13-15
- remote agent settings in auto.profile, 13-25
- remote authentication, 13-29
configuring, 13-29
Event Processor, 2-9
- user, 2-8
- RemoteProFiles, 13-16
- reports, 11-1
about, 11-3
all_events, 11-6
all_status, 11-7
defined, 11-1
defining, 11-3, 11-13
GUI, 11-11
JIL, 11-17
defining - GUI, 11-15
defining - JIL, 11-19
filtering, 11-6
filtering events, 11-2
job_filter, 11-7
name, 11-5
Operator Console, 10-8
status events, 11-7
- resource check
file space, 4-25, 4-27
- resources
Calendar Facility
date range, 8-30
Font Selection, 8-29
icon text, 8-30
Object Color, 8-29
Print Command, 8-30
title bar text, 8-30
Window Size, 8-30
- Job Definition, 6-25
database connection time-out, 6-25
icon text, 6-26
title bar text, 6-26
- Monitor/Browser
database connection time-out, 11-21
icon text, 11-21
title bar text, 11-21
- Operator Console, 10-32, 11-20
alarm list column width, 10-37
alarm poll time interval, 10-33
atomic condition fields, 10-36
border colors, 10-35
currently selected job name field color, 10-35

default report type, 10-37
font selection, 10-34
freeze frame at start up, 10-34
icon text, 10-38
job list column widths, 10-36
label font, 10-34
list font, 10-34
Operator Console size, 10-37
primary interface color, 10-36
refresh time interval, 10-33
title bar text, 10-38
toggle button color, 10-36
variable fields background color, 10-35

restart attempts
maximum number, 13-17

restart wait time
calculating, 13-18

RestartConstant, 13-18

RestartFactor, 13-18

restoring
calendar definitions, 12-14
global variables (using autorep), 12-14
job definitions (using autorep), 12-14
machine definitions (using autorep), 12-14
monitor and browser definitions, 12-14
primary event processor, 12-7

restricting access to jobs, 2-16

rollover
Event Server, 12-22
shadow event processor, 12-6

rstatd, 13-18

Rule Specification region, 8-15

run number, 3-26

RUN_AUTOSYSDB, 12-29

run_calendar, 4-10

run_num / ntry
defined, 3-26

ruserok, 2-8, 13-29

S

scripts
database maintenance, 13-12
DBMaint, 12-19
start_autodbd, 12-29

security, 2-1
database field verification, 2-7
DENY_ACCESS remote agent setting, 13-31
event processor authentication, 2-9
events sent by the event processor, 2-4
events sent by users, 2-2
granting permissions, 2-12
job definition encryption, 2-7
job level security, 2-10
job ownership, 2-10
job permissions and windows, 2-13
native, 2-2
overview, 2-1
permission types, 2-11
preventing unauthorized access, 2-7
Remote Agent, 2-17
remote agent authentication, 2-8
remote authentication, 13-29
restricting access to jobs, 2-16
security control, 2-13
superusers
AutoSys, 2-14
system level, 2-7
umask, 2-10
user and database administrator passwords, 2-9
user authentication, 2-8
user permissions, 2-10
user types, 2-11

security access, 2-20

security control, 2-13

select getdate, 12-29

Send Event dialog, 10-10
AUTOSERV instance, 10-11
Cancel button, 10-11
cancelling an event, 10-12
Change Status, 10-11

Comment field, 10-11
Execute button, 10-11
Global Name/Value, 10-11
Job Name, 10-10
Queue Priority, 10-11
Send Priority, 10-11
Signal, 10-11
time of event, 10-11

sendevent command, 2-15

server instance, 1-3

server machine, 1-8

ServerVision
charge back reporting, 9-15
job resource usage monitoring, 9-15
load balancing, 9-12

services file
Remote Agent port number, 13-27

shadow event processor
defined, 1-5
rollover, 12-6
starting, 12-7

SIGHUP signal, 13-19

signals
UNIX, 13-19

Single Server Mode, 12-22

SNMP connections, 13-17

socket connection for remote agent, 13-27

socket time-out, 13-17

sound monitor attribute, 11-9

SQL*Net V2, 12-15

standard output/error append parameter, 13-22

start interval between jobs, 13-23

start_autodb script, 12-29

starting conditions, 3-14, 3-16
in Job Activity Console, 10-7

STARTJOB, 7-4

states
job, 3-8
See also status, 3-8

status
job
as job dependency, 3-17
monitoring and reporting, 11-7
tracking changes, 11-7

monitor / report
failure, 11-7
restart, 11-7
running, 11-7
starting, 11-7
success, 11-7
terminated, 11-7

processed vs. unprocessed, 3-13
using in job dependencies, 3-16

std_err_file, 4-21

std_in_file, 4-20

STOP_DEMON, 12-5, 12-30

success
maximum exit code, 3-17

Summary Report
from Operator Console, 10-8

superusers, 2-14
Edit Superuser
defined, 2-14
exec superuser
defined, 2-15

svload command, 9-12

Sybase
accessing interactively, 12-31
architecture, 12-26
bundled, 12-26
defining a dump device, 12-34
dumping the database, 12-37
loading the database, 12-38
communication with, 12-26
database
displaying date and time, 12-33
identifying connected processes, 12-32

improving performance, 12-24
DB Library, 12-26
DSQUERY variable, 12-27
environment, 12-27
environment variable, 12-27
interfaces file, 12-27
server, 12-26
shutdown command, 12-30
SQL.INI file, 12-27
starting, 12-29
stopping, 12-30
System 10 backup server, 12-35
users, 12-27

SYBASE variable, 12-27

synchronizing event servers, 12-23

syntax rules
JIL, 7-1

system administrator
database, 12-28

system level security, 2-7

system load
maximum, 9-3

time dependencies
in job definition, 4-9
overview, 3-15
setting
GUI, 6-19
JIL, 7-11

timezone, 4-16

tmpfs
locking problems, 13-15

traceroute, B-2

transferring events between event servers, 13-12

troubleshooting, 14-1
event processor, 14-10
event servers, 14-2
remote agent, 14-12

troubleshooting tools
CCI, B-1
ccinet, B-3
netstat, B-1
nslookup, B-2
ping, B-2
traceroute, B-2

T

target machine, 4-7

technical support, 1-14

Term Calendar Rule dialog, 8-14
Control region, 8-21
Rescheduling Rule region, 8-20

Term Calendar Viewer, 8-7, 8-22
Calendar Display, 8-23
Navigation Controls, 8-23

term_run_time, 4-14

test mode
output file, 12-9
running in, 12-8

ThirdMachine, 13-10

U

uid, 2-11, 4-18

Unicenter Event Management Integration, 13-24
UnicenterEvents, 13-24

UnicenterEvents, 13-24

user ID, 2-11, 4-18

user types
group, 2-11
owner, 2-11
world, 2-11

user-defined
environment variables, 4-6

user-defined
load balancing, 9-21

user-defined
alarm callbacks, 13-32

users
bundled Sybase, 12-27

utilities
provided with AutoSys, 1-10

V

variables in a Command Job definition, 4-6

virtual machines, 9-2
defining, 9-2, 9-7
deleting, 9-8
example, 9-7
using delete_machine, 9-2
using insert_machine, 9-2

vmstat, 13-18

W

WaitTime, 13-18

watch_file, 4-8

watch_file_min_size, 4-26

watch_interval, 4-26

Windows NT
job permissions on, 2-13

X

X resources
customizing Calendar Facility, 8-28

XInstanceDBDropTime, 13-6

xql
example scripts, 12-31

