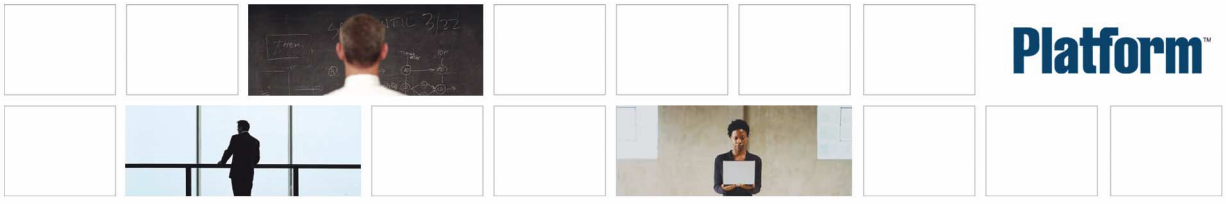
Administering Platform Process Manager™

Version 3.0
March 31 2005
Comments to: doc@platform.com



We'd like to hear from you

You can help us make this document better by telling us what you think of the content, organization, and usefulness of the information. If you find an error, or just want to make a suggestion for improving this document, please address your comments to doc@platform.com.

Your comments should pertain only to Platform documentation. For product support, contact support@platform.com.

Although the information in this document has been carefully reviewed, Platform Computing Corporation ("Platform") does not warrant it to be free of errors or omissions. Platform reserves the right to make corrections, updates, revisions or changes to the information in this document.

UNLESS OTHERWISE EXPRESSLY STATED BY PLATFORM, THE PROGRAM DESCRIBED IN THIS DOCUMENT IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL PLATFORM COMPUTING BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION ANY LOST PROFITS, DATA, OR SAVINGS, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS PROGRAM.

Last update March 31 2005

Contents

Welcome

Contents

About this Guide

This guide describes how to administer the Process Manager™ software ("Process Manager"). It provides an overview of Process Manager concepts, licensing information, how to configure Process Manager, commands, and some troubleshooting tips. This guide also describes how Process Manager interacts with Platform LSF ("LSF").

Who should use this guide

This guide is written for new Process Manager administrators who want to familiarize themselves with the fundamentals of managing a Process Manager system. For details regarding product use, see *Using* Platform Process Manager.

What you should already know

This guide assumes that you are familiar with basic LSF administration. You need to be familiar with the concepts of clusters, jobs, servers, and hosts, as described in *Platform LSF Administrator's Guide*.

How this guide is organized

Chapter 1 "About Process Manager" provides an overview of Process Manager, its data flow, failover, security, calendars, exception handling.

Chapter 2 "Maintaining Process Manager" describes how to add components to and maintain the Process Manager system.

Chapter 3 "Daemons" describes the Process Manager daemons and how to start and stop them.

Chapter 4 "Commands" describes each of the Process Manager commands, both end-user and administrative commands, and their syntax.

Chapter 5 "Files" describes the formats of each of the Process Manager files you use when maintaining Process Manager.

Typographical Conventions

Typeface	Meaning	Example
`Courier`	The names of on-screen computer output, commands, files, and directories	The `lsid` command
`Bold Courier`	What you type, exactly as shown	Type **`cd /bin`**
Italics	◆ Book titles, new words or terms, or words to be emphasized ◆ Command-line place holders—replace with a real name or value	The queue specified by *queue_name*
Bold Sans Serif	◆ Names of GUI elements that you manipulate	Click **OK**

Command Notation

Notation	Meaning	Example
Quotes " or '	Must be entered exactly as shown.	"*job_ID*[*index_list*]"
Commas ,	Must be entered exactly as shown.	-C *time0*,*time1*
Ellipsis …	The argument before the ellipsis can be repeated. Do not enter the ellipsis.	*job_ID* ...
lower case italics	The argument must be replaced with a real value you provide.	*job_ID*
OR bar |	You must enter one of the items separated by the bar. You cannot enter more than one item. Do not enter the bar.	[-h | -V]
Parenthesis ( )	Must be entered exactly as shown.	-X "*exception_cond*([*params*])::*action*] ...
Option or variable in square brackets [ ]	The argument within the brackets is optional. Do not enter the brackets.	lsid [-h]
Shell prompts	◆ C shell: `%` ◆ Bourne shell and Korn shell: `$` ◆ root account: `#` Unless otherwise noted, the C shell prompt is used in all command examples.	`% cd /bin`

Learning about Process Manager

World Wide Web and FTP

The latest information about all supported releases of Platform Process Manager is available on the Platform Web site at http://www.platform.com. Look in the Online Support area for current information.

If you have problems accessing the Platform web site, send email to info@platform.com.

Platform training

Platform's Professional Services training courses can help you gain the skills necessary to effectively install, configure and manage your Platform products. Courses are available for both new and experienced users and administrators at our corporate headquarters and Platform locations worldwide.

Customized on-site course delivery is also available.

Find out more about Platform Training at www.platform.com/training, or contact Training@platform.com for details.

Technical support

Contact Platform or your Process Manager vendor for technical support. Use one of the following to contact Platform technical support:

Email: support@platform.com

Toll-free phone:
- 1 877 444 4573

When contacting Platform, please include the full name of your company.

We'd like to hear from you

If you find an error in any Process Manager documentation, or you have a suggestion for improving it, please let us know. Contact doc@platform.com.

CHAPTER 1

About Process Manager

This chapter introduces Process Manager concepts and contains an overview of the Process Manager architecture. It also briefly describes the Process Manager Client components and their use.

Contents

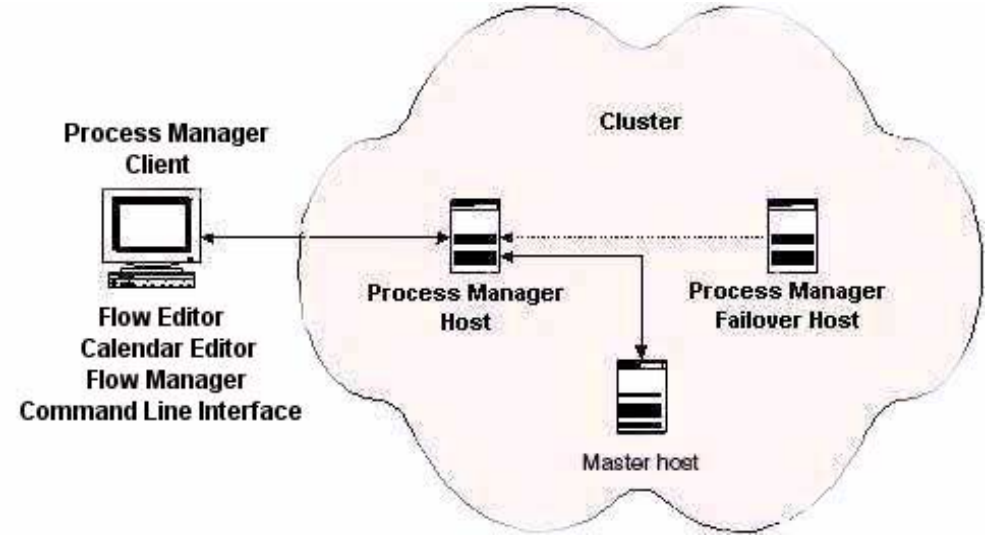
Overview

Process Manager is a workload management tool that allows users to automate their business processes in UNIX and Windows environments. Process Manager provides flexible scheduling capabilities and load balancing in an extensible, robust execution environment.

Using the Process Manager Client, users can create and submit complex flow definitions to Process Manager Server, which manages the dependencies within a flow and controls the submission of jobs to LSF. LSF provides resource management and load balancing, and runs the jobs and returns job status to the Process Manager Server. From Process Manager Client, users can also monitor and control their workflows within Process Manager.

An optional failover host provides Process Manager Server redundancy in the event that it experiences an outage.

Components



The system is made up of the following components:

- The Process Manager host
- The Process Manager failover host
- The Master host
- Process Manager Client, which consists of
  - The Flow Editor
  - The Calendar Editor
  - The Flow Manager

Process Manager Server

The Process Manager Server consists of a single daemon, `jfd`. The Process Manager Server controls the submission of jobs to LSF, managing any dependencies between work items.

The Process Manager Server failover host

An optional failover daemon (`fod`) is available for UNIX servers. The failover daemon starts the Process Manager Server and monitors its health. If required, the failover daemon starts the Process Manager Server on the failover machine.

LSF

LSF dispatches all jobs submitted to it by the Process Manager Server, and returns the status of each job to the Process Manager Server. It also manages any resource requirements and load balancing within the compute cluster.

Process Manager Client

The Process Manager Client contains the graphical client applications that work with Process Manager: Flow Editor, Calendar Editor, and Flow Manager.

The Flow Editor

Users use the Flow Editor to create flow definitions: the jobs and their relationships with other jobs in the flow, any dependencies they have on files, and any time dependencies they may have. Users also use the Flow Editor to submit their flow definitions, which places them under the control of Process Manager.

The Calendar Editor

Users use the Calendar Editor to define calendars, which Process Manager uses to calculate the days on which a job or flow should run. Calendars contain either specific dates or expressions that resolve to a series of dates. Process Manager calendars are independent of jobs, flow definitions and flows, so that they can be reused.

Users can create and modify their own calendars. These are referred to as *user* calendars. The Process Manager administrator can create calendars that can be used by any user of Process Manager. These are referred to as *system* calendars. Process Manager includes a number of built-in system calendars so you do not need to define some of the more commonly used expressions.

The Flow Manager

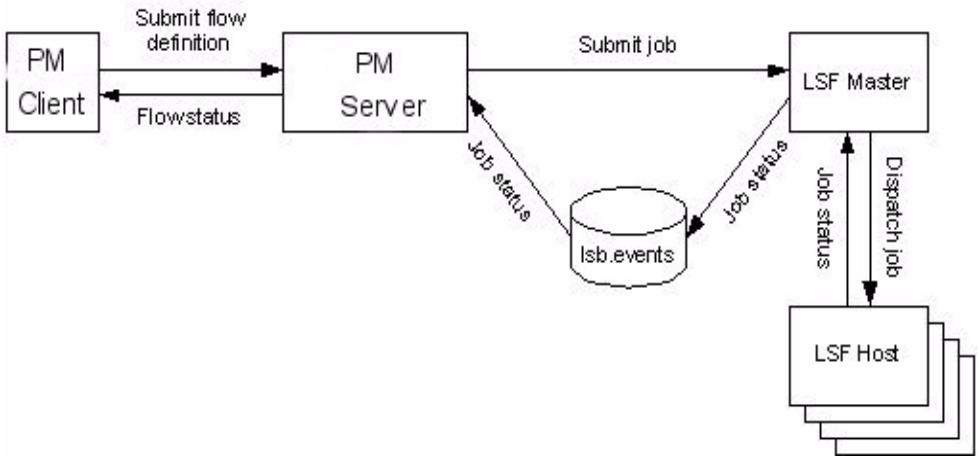
Users use the Flow Manager to trigger, monitor and control running flows, and to obtain history information about completed flows.

The command line interface

Users use the command line interface to submit predefined flows to the Process Manager Server, to trigger, monitor and control running flows, and to obtain history information about completed flows.

Data Flow

The following describes how Process Manager Server interacts with LSF to process flows:



1 The user uses the Flow Editor to create a flow definition and submits it to the Process Manager Server.

Your installation of Flow Editor allows you to create flow definitions but does not allow you to submit flow definitions. To enable full Flow Editor functionality, you must purchase an upgrade from Platform Computing. For more information, please contact Platform Computing at 1 877 444 4573.

2 The Process Manager Server stores the flow definition in its working directory.

3 When the flow is triggered, the Process Manager Server manages the dependencies within the flow. When a job is ready to be run, the Process Manager Server submits it to the LSF master host.

4 The LSF master host manages any resource dependencies the job may have, and dispatches the job to an appropriate LSF host.

5 When the job runs, the LSF host sends the status of the job to the LSF master host, which writes the job status to `lsb.events`.

6 The Process Manager Server reads `lsb.events` periodically to obtain the status of the jobs it submitted to LSF.

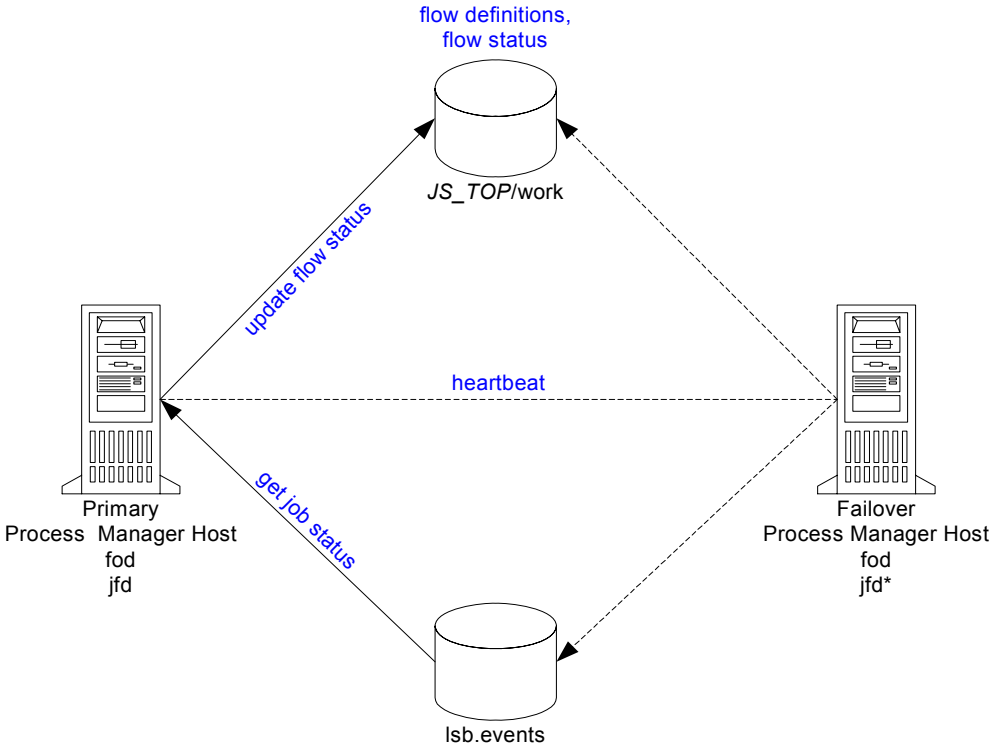
7 The Process Manager Server uses the status of the job to determine the next appropriate action in the flow.

8 On request from the user, the Process Manager Server presents flow status to the client.

About Failover

Process Manager supports an optional failover feature on UNIX, which provides redundancy for the Process Manager Server. The failover feature allows you to configure a second Process Manager Server host to take over the responsibilities of the primary Process Manager Server host if it should fail. The failover feature includes the failover daemon (`fod`), which starts the Process Manager Server on the primary Process Manager Server host. The failover daemon monitors the health of the primary Process Manager Server, starting Process Manager Server on the failover host if the primary fails to respond within a certain time period.

The failover feature relies on a shared file system for access to the working directory of the Process Manager Server.



1. Process Manager Server updates flow status in its working directory based on data it reads from `lsb.events`.
2. The `fod` on the failover host monitors the health of `fod` on the primary host. If it receives no response from the heartbeat, it assumes the primary host is down, and starts `jfd` on the failover host. Process Manager Server is now running on the failover host.
3. The `fod` on the failover host continues to monitor for a response from `fod` on the primary host. When it receives a response, it stops `jfd` on the failover host, returning control to the primary host.

   The failover host requires access to both the Process Manager working directory *`JS_TOP`*`/work`, and the events file `lsb.events`.

About Calendars

Process Manager uses calendars to define the dates in a time event, which is used to determine when a flow triggers or a job runs. Calendars are defined independently of flows and jobs so that they can be associated with multiple time events.

A time event consists of the date and time to trigger the event, and the duration in which the event is valid. The calendar provides the date specification for the time event.

Process Manager has two types of calendars:

- User calendars
- System calendars

You create both types of calendars using the Calendar Editor.

Users can only manipulate their own calendars, but they can use system calendars and calendars belonging to other users when combining calendars.

Note that the time used in the calendars is based on the time zone set in JS_TIME_ZONE. The time zone can be set as user, client, or Universal Time (UTC). For more information, see "JS_TIME_ZONE" on page 136.

About user calendars

User calendars are created by individual users. Users create a new calendar when they have a requirement for a unique time event, and no calendar in the current list of calendars resolves to the correct date or set of dates. Users can create simple calendars, or calendars that combine multiple calendars, both user and system, to create complex schedule criteria.

These calendars are owned by the user who created them and can be used by any user. Only the owner can modify or delete these calendars.

About system calendars

System calendars are built-in or created by a Process Manager administrator. These calendars are owned by the virtual user `Sys` and can be used by any user. Only the Process Manager administrator can modify system calendars.

Process Manager comes with a set of pre-defined system calendars that you can use as is or modify to suit the needs of your site. In addition to these built-in calendars, the Process Manager administrator may define other system calendars.

About changing or deleting calendars

Once created, calendars can be changed or deleted. However, you cannot change or delete a calendar when it is in use—when a flow definition is triggered by an event that uses the calendar, when a flow is running and contains a time event that uses that calendar, or when the calendar is referenced by another calendar.

Built-in system calendars

Types of Calendars	Calendar Names
Weekly calendars	Mondays@Sys Tuesdays@Sys Wednesdays@Sys Thursdays@Sys Fridays@Sys Saturdays@Sys Sundays@Sys Daily@Sys Weekdays@Sys Weekends@Sys Businessdays@Sys
Monthly calendars	First_monday_of_month@Sys First_tuesday_of_month@Sys First_wednesday_of_month@Sys First_thursday_of_month@Sys First_friday_of_month@Sys First_saturday_of_month@Sys First_sunday_of_month@Sys First_weekday_of_month@Sys Last_weekday_of_month@Sys First_businessday_of_month@Sys Last_businessday_of_month@Sys Biweekly_pay_days@Sys
Yearly calendars	Holidays@Sys First_day_of_year@Sys Last_day_of_year@Sys First_businessday_of_year@Sys Last_businessday_of_year@Sys First_weekday_of_year@Sys Last_weekday_of_year@Sys

The Holidays@Sys calendar

When you receive Process Manager, it comes with some predefined system calendars. Most of these calendars are ready to be used. The calendar Holidays@Sys can be a particularly important calendar for use in creating schedules, but it should be edited to reflect your company holidays, before users begin creating schedules. It should also be updated annually, to reflect the current year's statutory holidays, company-specific holidays, and so on.

Some of the other built-in calendars rely on the accuracy of Holidays@Sys, including any calendar that defines business days, since a business day is a weekday that is not a holiday. See "To update the Holidays@Sys calendar:" on page 55 for instructions.

The Biweekly_pay_days@Sys calendar

The Biweekly_pay_days@Sys calendar assumes a Friday pay day. If biweekly pay days are a different day of the week, edit this calendar to specify the correct day of the week for pay days.

About Exceptions

Process Manager provides flexible ways to handle certain job processing failures so that you can define what to do when these failures occur. A failure of a job to process is indicated by an exception. Process Manager provides some built-in exception handlers you can use to automate the recovery process, and an alarm facility you can use to notify people of particular failures.

Types of exceptions

Process Manager monitors for the following exceptions:

- Misschedule
- Overrun
- Underrun
- Start Failed
- Cannot Run

Misschedule

A *Misschedule* exception occurs when a work item depends on a time event, but is unable to start during the duration of that event. There are many reasons why your job can miss its schedule. For example, you may have specified a dependency that was not satisfied while the time event was active.

Overrun

An *Overrun* exception occurs when a work item exceeds its maximum allowable run time. You use this exception to detect run away or hung jobs.

Underrun

An *Underrun* exception occurs when a work item finishes sooner than its minimum expected run time. You use this exception to detect when a job finishes prematurely.

Start Failed

A *Start Failed* exception occurs when a job or job array is unable to run because its execution environment could not be set up properly. Typical reasons for this exception include lack of system resources such as a process table was full on the compute host, or a file system was not mounted properly.

Cannot Run

A *Cannot Run* exception occurs when a job or job array cannot proceed because of an error in submission. A typical reason for this exception might be an invalid job parameter.

Behavior when an exception occurs

The following describes the behavior when an exception occurs, and no automatic exception handling is specified:

When a...	Experiences this exception...	This happens...
Flow definition	Misschedule	The flow is not triggered.
Flow	Overrun	The flow continues to run after the exception occurs. The run time is calculated from when the flow is first triggered until its status changes from Running to Exit or Done, or until the Overrun time is reached, whichever comes first.
	Underrun	The time is calculated from when the flow is first triggered until its status changes from Running to Exit or Done.
Subflow	Misschedule	The subflow is not run.
	Overrun	The subflow continues to run after the exception occurs. The run time is calculated from when the subflow is first triggered until its status changes from Running to Exit or Done, or until the Overrun time is reached, whichever comes first.
	Underrun	The time is calculated from when the subflow first starts running until its status changes from running to Exit or Done.
Job	Misschedule	The job is not run.
	Cannot Run	The job is not run.
	Start Failed	The job is still waiting. Submission of the job is retried until the configured number of retry times. If the job still cannot run, a Cannot Run exception is raised. The default number of retry times is 20. To change this value, see "To change the job start retry value:" on page 47.
	Overrun	The job continues to run after the exception occurs. The run time is calculated from when the job is successfully submitted until it reaches Exit or Done state, or until the Overrun time is reached, whichever comes first.
	Underrun	The time is calculated from when the job is successfully submitted until it reaches Exit or Done state.
Job array	Misschedule	The job array is not run.
	Cannot Run	The job array is not run.
	Start Failed	The job array is still waiting. Submission of the job array is retried the configured number of retry times. If the job array still cannot be started, a Cannot Run exception is raised. The default number of retry times is 20. To change this value, see "To change the job start retry value:" on page 47.
	Overrun	The job array continues to run after the exception occurs. The run time is calculated from when the job array is successfully submitted until its status changes from Running to Exit or Done, or until the Overrun time is reached, whichever comes first.
	Underrun	The time is calculated from when the job array is successfully submitted until each element in the array reaches Exit or Done state.

User-specified conditions

In addition to the exceptions, you can specify and handle other conditions, depending on the type of work item you are defining. For example, when you are defining a job, you can monitor the job for a particular exit code, and automatically rerun the job if the exit code occurs. The behavior when one of these conditions occurs depends on what you specify in the flow definition.

You can monitor for the following conditions:

Work Item	Condition
Flow	An exit code of *n* (sum of all exit codes)
	n unsuccessful jobs
	A work item has exit code of *n*
Subflow	An exit code of *n*
	n unsuccessful jobs
	A work item has exit code of *n*
Job	An exit code of *n*
Job array	An exit code of *n*
	n unsuccessful jobs

About Exception Handling

Process Manager provides built-in exception handlers you can use to automatically take corrective action when certain exceptions occur, minimizing the human intervention required. You can also define your own exception handlers for certain conditions. See "User-defined exception handlers" on page 25 for more information.

Built-in exception handlers

The built-in exception handlers are:

- Rerun
- Kill
- Opening an alarm

Rerun

The *Rerun* exception handler reruns the entire work item. Use this exception handler in situations where rerunning the work item can fix the problem. The Rerun exception handler can be used with Underrun, Exit and Start Failed exceptions. Work items that have a dependency on a work item that is being rerun cannot have their dependency met until the work item has rerun the last time.

Kill

The *Kill* exception handler kills the work item. Use this exception handler when a work item has overrun its time limits. The Kill exception handler can be used with the Overrun exception, and when you are monitoring for the number of jobs done or exited in a flow or subflow.

Alarm

An *alarm* provides both a visual cue that an exception has occurred, and sends an email notification to one or more email addresses. You use an alarm to notify key personnel, such as database administrators, of problems that require attention. An alarm has no effect on the flow itself.

You can use an alarm as an automated exception handler for many types of exceptions.

An opened alarm appears in the list of open alarms in the Flow Manager until the history log file containing the alarm is deleted or archived.

Alarms are configured by the Process Manager administrator.

Behavior when built-in exception handlers are used

The following describes the behavior when an exception handler is used:

When a...	Experiences this Exception...	and the Handler Used is...	This Happens...
Flow	Overrun	Kill	The flow is killed. All incomplete jobs in the flow are killed. The flow status is 'Killed'.
		Alarm	The alarm is opened. The flow continues execution as designed.
	Underrun	Rerun	Flows that have a dependency on the success of this flow may not be triggered, depending on the type of dependency. The flow is recreated with the same flow ID. The flow is rerun from the first job, as many times as required until the execution time exceeds the underrun time specified.
		Alarm	The alarm is opened.
	Flow has exit code of *n*	Rerun	Flows that have a dependency on this flow may not be triggered, depending on the type of dependency. The flow is recreated with the same flow ID. The flow is rerun from the first job, as many times as required until an exit code other than *n* is reached.
		Alarm	The alarm is opened. Flows that have a dependency on this flow may not be triggered, depending on the type of dependency.
	n unsuccessful jobs	Kill	The flow is killed. All incomplete jobs in the flow are killed. The flow status is 'Killed'.
		Alarm	The alarm is opened. Flows that have a dependency on this flow may not be triggered, depending on the type of dependency. The flow continues execution as designed.
	Work item has exit code of *n*	Rerun	Flows that have a dependency on this flow may not be triggered, depending on the type of dependency. The flow is rerun from the first job, as many times as required until the work item has a different exit code.

When a...	Experiences this Exception...	and the Handler Used is...	This Happens...
Subflow	Overrun	Kill	The subflow is killed. The flow behaves as designed.
		Alarm	The alarm is opened. Both the flow and subflow continue execution as designed.
	Underrun	Rerun	Work items that have a dependency on this subflow may not be triggered, depending on the type of dependency. The subflow is rerun from the first job, as many times as required until the execution time exceeds the underrun time specified.
		Alarm	The alarm is opened. The flow continues execution as designed.
	Subflow has exit code of *n*	Rerun	Work items that have a dependency on this subflow may not be triggered, depending on the type of dependency. The subflow is rerun from the first job, as many times as required until an exit code other than *n* is reached.
		Alarm	The alarm is opened. The flow continues execution as designed.
	n unsuccessful jobs	Kill	The subflow is killed. The flow behaves as designed.
		Alarm	The alarm is opened. The flow and subflow continue execution as designed.
	A work item has exit code of *n*	Rerun	Work items that have a dependency on this flow may not be triggered, depending on the type of dependency. The flow is rerun from the first job, as many times as required until the work item has a different exit code.
Job or job array	Overrun	Kill	The job or job array is killed. The flow behaves as designed. The job or job array status is determined by its exit value.
		Alarm	The alarm is opened. Both the flow and job or job array continue to execute as designed.
	Underrun	Rerun	Objects that have a dependency on this job or job array may not be triggered, depending on the type of dependency. The job or job array is rerun as many times as required until the execution time exceeds the underrun time specified.
		Alarm	The alarm is opened. The flow continues execution as designed.
	An exit code of *n*	Rerun	The job or job array is rerun as many times as required until it ends successfully.
		Alarm	The alarm is opened. The flow behaves as designed.
	n unsuccessful jobs	Kill	The job array is killed. The flow behaves as designed. The job array status is determined by its exit value.
		Alarm	The alarm is opened. The flow continues execution as designed.

User-defined exception handlers

In addition to the built-in exception handlers, you can create your flow definitions to handle exceptions by:

- Running a recovery job
- Triggering another flow

Recovery job

You can use a job dependency in a flow definition to run a job that performs some recovery function when an exception occurs.

Recovery flow

You can create a flow that performs some recovery function for another flow. When you submit the recovery flow, specify the name of the flow and exception as an event to trigger the recovery flow.

CHAPTER C

2

Maintaining Process Manager

This chapter describes how to add components to the Process Manager system, how to maintain the system, how to obtain historical information and some troubleshooting techniques.

Contents

Adding a Client

To install a UNIX client, follow the steps in "Adding a UNIX client:". To install a Windows client, follow the steps in "Adding a Windows client:" on page 29.

Adding a UNIX client:

1. Copy the client tar file for the operating system Process Manager Client will run on to the UNIX host on which you want to install Process Manager. For example, `pm3.0_sas_linux2.4-glibc2.2-x86.tar`
2. Untar `pm3.0_sas_linux2.4-glibc2.2-x86.tar` as follows:

   `tar xvf pm3.0_sas_linux2.4-glibc2.2-x86.tar`

   This creates a directory called `pm3.0_sas_pinstall`.
3. In `pm3.0_sas_pinstall`, edit the file `install.config` to define your configuration. Complete sections 1 and 3. Remove the comment symbol (#) and set values for the following parameters:

Section 1

a. For JS_TOP, specify the full path to the top-level Process Manager installation directory. The installation script will create the directory you specify.

b. For JS_HOST, specify the fully qualified hostname of the host on which the Process Manager daemon will run. You can specify only one host, as each host requires its own configuration files.

c. For JS_PORT, specify the port number through which the clients will access the Process Manager Server. The default is 1966.

d. For JS_TARDIR, specify the full path to the directory containing the Process Manager distribution tar files. The default is the parent directory of the current working directory where `jsinstall` is running.

Section 3

Section 3 specifies whether or not you want to install LSF.

a. For LSF_INSTALL, specify **`true`** if you want to install LSF as well or **`false`** if you do not.

b. If you specified LSF_INSTALL=false, for LSF_ENVDIR specify the path of your LSF installation.

4. Save `install.config`.
5. As root, run `jsinstall` to start the installation:

   ```
   # ./jsinstall -f install.config

   Logging installation sequence in
   .../pm3.0_sas_pinstall/pm3.0_sas_install
   ```
6. If you selected to install LSF as well, read the End User License Agreement and enter **`Y`** to accept the terms.
7. Select the Process Manager Client. For example:

```
Searching for Platform Process Manager tar files in /usr/share/pm
please wait ...

  1) Solaris 2.7-32/Solaris 2.8-32 Server
  2) Solaris 2.7-32/Solaris 2.8-32 Client

List the numbers separated by spaces that you want to install.
(E.g. 1 3 7, or press Enter for all): 2
```

8 After the installation is complete, set the Process Manager environment:

- On `csh` or `tcsh`:

  `# source JS_TOP/conf/cshrc.js`

- On `sh`, `ksh` or `bash`:

  `# . JS_TOP/conf/profile.js`

Where *JS_TOP* is the top-level Process Manager installation directory, the value specified in the `install.config` file.

9 Run the three Process Manager Client applications: Flow Editor, Flow Manager, and Calendar Editor as follows:

a Run **`floweditor`**

b Run **`flowmanager`**

c Run **`calediter`**

Both the Flow Manager and the Calendar Editor require a connection to the Server to be able to start. If you are unable to start either of these applications, there is an error in the configuration, or the Server is not yet started.

Adding a Windows client:

1 Copy `pm3.0_pinstall_sas_win.exe` to the desktop or a shared file location from which you can run it.

2 Run `pm3.0_pinstall_sas_win.exe` to start the installation.

3 In the **Welcome** dialog, click **Next**

4 In the **Choose Destination Location** dialog, click **Next** to use to the default location; or click **Browse...** to select a different directory. Click **Next**.

In the **Select Components** dialog, select the Process Manager Client. Click **`Next`**.

1 In the **Client Configuration** dialog:

a In the **Host name** field, specify the name of the Process Manager host the desktop will connect to.

b In the **Port** field, specify the port number of the Process Manager host. If you used the default port number for the Server, leave the value at 1966.

c Click **Next**.

2 Verify that the settings are correct, and click **Next** to complete the installation.

3 Click **Finish**.

4 When the installation is complete, from the **Start** menu, select Platform Process Manager, and the appropriate application: Flow Editor, Flow Manager, or Calendar Editor.

Both the Flow Manager and the Calendar Editor require a connection to the Server to be able to start. If you are unable to start either of these applications, there is an error in the configuration, or the Server is not yet started.

Running the Process Manager Server on System Startup

On UNIX, the Process Manager Server can be configured to start and stop at system startup or shutdown. On Windows, the Process Manager Server runs as a service, and by default, starts and stops automatically with the system.

To run the Process Manager on system startup:

1. Ensure installation of the Process Manager daemon is complete, and that you have sourced the correct environment.
2. Log on as `root` to the host where the Process Manager daemon is installed.
3. Run the following script:

   ```
   #./bootsetup
   ```

   This script picks up your environment information and enables the daemon to start and stop at system boot time.

Dedicating the Process Manager Server Host

If you are running large flows or a large number of flows, it is recommended that you designate your Process Manager Server host as an LSF client host, rather than an LSF server host.

To dedicate the Process Manager Server host:

1. Edit the LSF cluster file `lsf.cluster.`*`cluster_name`*.
2. In the Host section of the file, locate the name of the host on which the Process Manager Server.
3. In the Server column for the primary Process Manager host, enter **0**, which specifies that this is a client host and does not run LSF jobs. For example:

```
Begin Host
HOSTNAME    model     type    server r1m pg tmp RESOURCES       RUNWINDOW
hostA       SparcIPC  Sparc   1      3.5 15   0 (sunos frame)   ()
hostD       Sparc10   Sparc   1      3.5 15   0 (sunos)         (5:18:30-1:8:30)
jshost      !         !       0      2.0 10   0 ()              ()
End Host
```

4. Save the file.
5. Run `lsadmin reconfig` and `badmin reconfig` to reconfigure the LSF cluster.

About User Variables

Process Manager provides substitution capabilities through the use of variables. When Process Manager encounters a variable, it substitutes the current value of that variable.

Process Manager users can use variables as part or all of a file name to make file names flexible, or use them to pass arguments to any job, or from scripts. They can export the value of a variable to one or more jobs in a flow, or to other flows that are currently running on the same Process Manager Server.

Process Manager users can set a value for a single variable within a script, or set values for a list of variables, and make all of the values available to the flow or to the Process Manager Server. They can use a single variable or a list of variables within a job, job array or file event definition.

Types of variables

There are two types of variables Process Manager users can set:

- Local variables—those whose values are available only to jobs, job arrays, subflows or events within the current flow. These variables are set in JS_FLOW_VARIABLE_LIST.
- Global variables—those whose values are available to all the flows within the Process Manager Server. These variables are set in JS_GLOBAL_VARIABLE_LIST.

Scope of variables

The variables set by the job have similar scope to variables in any programming language (C for example). If the job sets the variable in JS_FLOW_VARIABLE_LIST within a subflow, the scope of the variable is limited to the jobs and events within the subflow. If the same variable is overwritten by another job within the subflow, the new value is used for all subsequent jobs or events inside that subflow.

Local variable values override global variable values. Similarly, a value set within a subflow overrides any value set at the flow level, only within the subflow itself.

How variables are set

Process Manager uses a job starter as a wrapper to a job to export any user variables that are set within the job. The job starter actually runs the executable the job is defined to run. When the executable finishes, the job starter obtains any variables and values that were set by the job from JS_FLOW_VARIABLE_LIST and JS_GLOBAL_VARIABLE_LIST. The variables are written to the shared directory under *JS_TOP*/work/var_comm, where they are stored temporarily. The Process Manager Server retrieves the variables and their values and saves them in permanent storage under *JS_TOP*/work/variable.

Configuring to Support User Variables

If users in your Process Manager system will be setting and using user variables, you need to configure the system to support this.

- If the Process Manager Server runs on UNIX, and users will be setting variables in jobs that run on UNIX hosts, go to "To configure variables for UNIX hosts:".
- If the Process Manager Server runs on Windows, and users will be setting variables in jobs that run on Windows hosts, go to "To configure variables for Windows hosts:".
- If the Process Manager Server runs on UNIX and users will be setting variables from both UNIX and Windows hosts, go to you need to follow both sets of instructions.
- If your users will be using many variables in any job definition field, you may need to increase the number of variables that can be substituted at a time per field. Go to "To increase the number of variables that can be substituted:" for instructions.

To configure variables for UNIX hosts:

1. Configure one or more UNIX-specific queues to accept jobs that set variables. See "To configure a queue to support setting user variables:" for instructions.
2. Ensure that the korn shell (ksh) is available on the host, as the korn shell is required to export variables on UNIX.
3. Ensure that the *JS_TOP* directory is accessible by all LSF hosts that will run jobs that set variables—on a shared file system.

To configure variables for Windows hosts:

1. Configure one or more Windows-specific queues to accept jobs that set variables. See "To configure a queue to support setting user variables:" for instructions.
2. Ensure that the *JS_TOP* directory is accessible by all LSF hosts that will run jobs that set variables—on a shared file system.

To configure variables for both UNIX and Windows hosts:

1. Configure at least one Windows-specific queue and at least one Windows-specific queue to accept jobs that set variables. See "To configure a queue to support setting user variables:" for instructions.
2. On the UNIX LSF hosts, ensure that the korn shell (ksh) is available, as the korn shell is required to export variables on UNIX.
3. Log on to the Process Manager Server host as root or as the primary Process Manager administrator.
4. Configure the Server host as follows:
   a. Copy `platform_pm3.0_w2k_writevar.tar.Z` to the directory containing the Process Manager distribution files.
   b. Run `jsinstall` to start the installation:
      ```
      # ./jsinstall -f install.config
      ```
   c. Select **Windows 2000 Variables** from the list of components to install.
   d. Press **Enter** to complete the installation.

5 Edit `jsstarter.bat`
6 Set a value for *JS_TOP*. For example:
```
set JS_TOP=\\user\share\js
```
7 Save `jsstarter.bat`.
8 Ensure that the *JS_TOP* directory is accessible by all LSF hosts that will run jobs that set variables—on a shared file system.
9 Restart LSF.

To configure a queue to support setting user variables:

Note Any jobs submitted to the queues for setting variables must be wrapped in a script. It is recommended that you create these queues exclusively for setting variables to avoid confusion.

1 Create a new queue in the LSF queues file `lsb.queues`. If users will be setting variables in both UNIX and Windows jobs, you'll need a separate queue for each.
2 Add the variable JOB_STARTER in the queue configuration to point to the starter script shipped with Process Manager. Starter scripts are available in *JS_TOP*`/3.0/bin`.

   For example, for a UNIX queue:

   `JOB_STARTER=/`*`JS_TOP`*`/3.0/bin/jsstarter`

   For example, for a Windows queue:

   `JOB_STARTER=/`*`JS_TOP`*`/3.0/bin/jsstarter.bat`

   Ensure that the value you specify for *JS_TOP* is a fully-qualified UNC (Universal Naming Convention) name on a shared file system.
3 Run `badmin reconfig` to reconfigure LSF.

To increase the number of variables that can be substituted:

1 Follow the instructions in "Changing the Configuration" on page 37 to stop Process Manager Server and edit `js.conf`.
2 Add a line that specifies the maximum number of variable substitutions that can be performed in a single job definition field by specifying a value for JS_MAX_VAR_SUBSTITUTIONS For example:

   `JS_MAX_VAR_SUBSTITUTIONS=20`

   The default is 10 substitutions.
3 Complete the instructions for changing your configuration, saving `js.conf`, and starting Process Manager Server.

Controlling the Process Manager Server

Starting and stopping the Server on UNIX

On UNIX, the Process Manager Server has a single daemon, jfd. You control jfd with the jadmin command.

To start the Process Manager daemon

1. Log on to the Process Manager Server host as `root`.
2. Run **`jadmin start`**. This command starts jfd.

To stop the Process Manager daemon

1. Log on to the Process Manager Server host as `root` or as the primary Process Manager administrator.
2. Run **`jadmin stop`**. This command stops jfd.

Starting and stopping the Server on Windows

On Windows, the Process Manager Server runs as a service. By default, it is configured to start and stop automatically when the host is started and stopped.

To start the Process Manager service

1. Click **Start,** select **Settings**, and select **Control Panel.**
2. Double-click **Administrative Tools.**
3. Double-click **Services.**
4. Right-click on the service **Platform Process Manager** and select **Start**.

To stop the Process Manager service

1. Click **Start,** select **Settings**, and select **Control Panel.**
2. Double-click **Administrative Tools.**
3. Double-click **Services.**
4. Right-click on the service **Platform Process Manager** and select **Stop**.

Changing the Configuration

After you have installed the basic Process Manager configuration, you may need to change a configuration value, such as adding administrators.

To change a configuration value on UNIX:

1. Log on to the Process Manager Server host as `root` or as the primary Process Manager administrator.
2. Run **`jadmin stop`**.
3. Edit *JS_TOP*`/conf/js.conf`.
4. Make your changes.
5. Save `js.conf`.
6. Run **`jadmin start`** to start the Process Manager Server and make your changes take effect.

To change a configuration value on Windows:

1. Stop the Process Manager Server service. See "Starting and stopping the Server on Windows" on page 36 for instructions.
2. Edit *JS_TOP*`/conf/js.conf`.
3. Make your changes.
4. Save `js.conf`.
5. Start the Process Manager Server service to make your changes take effect.

Adding an Administrator

Process Manager uses role-based access control to secure certain types of objects. Special permissions are required to install and configure Process Manager, or to modify Process Manager items on behalf of another user.

Process Manager recognizes the following kinds of administrators:

- Primary Process Manager administrator—required to install a Process Manager Server and change permissions. It is also the user under which the Process Manager Server runs, and is the minimum authority required to stop the Process Manager Server. This is the first administrator defined in the list of administrators for the JS_ADMINS parameter in `js.conf`—there can be only one.
- Process Manager administrator—can create, delete, modify flows on behalf of another user. You can specify as many of these as required. You can also specify UNIX user group names as administrators. These are the administrators specified after the primary administrator for the JS_ADMINS parameter in `js.conf`.
- Process Manager control administrator—can control existing Process Manager items on behalf of another user. This user cannot submit or remove flows belonging to another user. You can specify as many of these as required. You can also specify UNIX user group names as control administrators. These are the administrators specified in the JS_CONTROL_ADMINS parameter in `js.conf`.

To add an administrator:

1. Follow the instructions in "Changing the Configuration" on page 37 to stop the Process Manager Server and edit `js.conf`.
2. To add a Process Manager administrator, for the JS_ADMINS parameter, specify one or more user IDs after the primary administrator name. Separate the names with a comma.
3. For JS_CONTROL_ADMINS, specify one or more user IDs or UNIX user group names. To specify a list, separate the names with a comma.
4. Complete the instructions for changing your configuration, saving `js.conf` and starting the Process Manager Server.

Maintaining User Passwords

Every job has a user ID associated with it. That user ID must always have a current password in the LSF password file, or the job is unable to run.

If user passwords at your site never expire, you simply need to ensure that all user IDs under which jobs might run initially have a password entered for them in the LSF password file. After that, maintenance is only required to add passwords for new users.

If user passwords at your site expire on a regular basis, you and your users need to be aware that a user's jobs cannot run if their passwords change and the LSF password file is not updated.

Updating the LSF password file

There are two ways that a user's password can be updated:

- Automatically
- By running the `lspasswd` command

Automatic updates

Every time a user logs into either the Flow Manager or the Calendar Editor, the user's password is updated in the LSF password file.

Running lspasswd

A user can update their own password without logging into the Flow Manager or Calendar Editor by running the `lspasswd` command. Simply run `lspasswd` and enter the current password when prompted.

Running a job as another user

If you, as the administrator, define a flow that runs a job on behalf of another user, you need to ensure that user's password is in the LSF password file. If the user logs on to either the Flow Manager or Calendar Editor regularly, the password is probably up to date. If not, either you or the user needs to run `lspasswd` to update the user's password so the job can run. Obviously, if you run `lspasswd` on behalf of the user, you need to know the user's password.

Specifying the Mail Host

The mail host parameter in `js.conf` defines the type of email server used and the name of the email host. This information is important for receiving email notifications from the Process Manager Server.

To specify the mail host:

1. Follow the instructions in "Changing the Configuration" on page 37 to stop the Process Manager Server and edit `js.conf`.
2. If the parameter JS_MAILHOST is already defined, change the value to specify the new email host. Otherwise, add a line that specifies the type of mail host and the name of the mail server host. For an SMTP mail host, specify SMTP:*hostname* as shown:

   `JS_MAILHOST=SMTP:barney`

   For an Exchange mail host, specify Exchange:*hostname*, as shown:

   `JS_MAILHOST=Exchange:fred`

   The default is SMTP on the local host.
3. Complete the instructions for changing your configuration, saving `js.conf` and starting the Process Manager Server.

Changing the Job Start Retry Value

The job start retry value controls the number of times that the Process Manager Server tries to start a job or job array before it raises a Start Failed exception.

To change the job start retry value:

1. Follow the instructions in "Changing the Configuration" on page 37 to stop the Process Manager Server and edit `js.conf`.
2. If the parameter JS_START_RETRY is already defined, change the value to specify the new number of retry times. Otherwise, add a line like the following to the file:

   `JS_START_RETRY=`*n*

   where *n* is the number of times to retry starting a job or job array before raising a Start Failed exception.
3. Complete the instructions for changing your configuration, saving js.conf and starting the Process Manager Server.

Changing the History Setting

History information is stored in a history log file. Data is added to this file for either a set period of time after a flow has completed, or when the history log file reaches a certain size. By default, these values are set to 24 hours or 500 KB, whichever occurs first. You can change these values after installation. After the set amount of time has elapsed, or the file reaches the specified size, a new history log file is created. The previous file remains in the log directory until you archive it or delete it.

To change the history setting:

1. Follow the instructions in "Changing the Configuration" on page 37 to stop the Process Manager Server and edit `js.conf`.
2. Locate the following parameters in the file:

   ```
   # JS_HISTORY_LIFETIME=24
   # JS_HISTORY_SIZE=500000
   ```

   and change them as follows:

   a. Delete the comment symbol (#) from the lines you want to change.
   b. Change the JS_HISTORY_LIFETIME value to the maximum number of hours of data you want to keep in each file.
   c. Change the JS_HISTORY_SIZE value to the maximum number of bytes of data you want to keep before creating a new file.

   Historical data will be kept in the current log file until either the size limit or the time limit is reached, whichever is reached first.
3. Complete the instructions for changing your configuration, saving js.conf and starting the Process Manager Server.

Creating System Calendars

Process Manager uses system calendars to share scheduling expressions that are commonly used. System calendars are created by the Process Manager administrator, and are owned by the virtual user `Sys`. They can be viewed and referenced by everyone.

Each system calendar is stored as an individual file in *`JS_TOP`*`/work/calendars`—one calendar per file.

You create a calendar using the Calendar Editor, and change the owner name to **`Sys`**.

Calendar names

When you create a calendar, you need to save it with a unique name. Some rules apply:

- Calendar names can contain the digits 0 to 9, the characters a to z and A to Z, and underscore (_)
- Calendar names cannot begin with a number
- System calendars are named as follows:

  calendar_name`@Sys`

To create a system calendar:

1. Using the Calendar Editor, create the calendar and save it. The calendar will be saved with your own user ID as the owner. For instructions on using the Calendar Editor, see *Using Platform* Process Manager, or the Calendar Editor online help.
2. In *`JS_TOP`*`/work/calendars`, locate the calendar you created. Change the owner of the calendar by editing the file and changing the owner from your user ID to **`Sys`**. Refer to the following example, where the owner is highlighted:

```
random
(2002/5/25,2002/6/16,2002/6/2,2002/6/3)
bhorner
random
1022181937
```

3. Rename the file or save the file with a new name. Ensure the suffix of the calendar is `Sys`.
4. If applicable, delete the original calendar you created.

To update the Holidays@Sys calendar:

1. Open the `Holidays@Sys` calendar.
2. Save the calendar with a new name.
3. Edit the list of dates to include all those dates that are company-wide holidays.
4. In *`JS_TOP`*`/work/calendars`, locate the calendar you created. Change the owner of the calendar by editing the file and changing the owner from your user ID to **`Sys`**. Refer to the following example, where the owner is highlighted:

```
random
(2002/5/25,2002/6/16,2002/6/2,2002/6/3)
bhorner
random
1022181937
```

5 Delete the original Holidays@Sys calendar.

6 Rename the file to Holidays@Sys. Ensure the suffix of the calendar is `Sys`.

Deleting a calendar

Periodically, you or a user may need to delete a calendar. This can be done from the Calendar Editor, or by using the `jcdel` command.

When a flow references a deleted calendar

A flow definition that references a deleted calendar can still be triggered, either automatically by an event, or manually. The flow definition can even be submitted, provided that the Process Manager daemon has not been restarted since the calendar was deleted. Flows that are running when the calendar is deleted are still able to run, provided that the daemon has not been restarted since the calendar was deleted. However, when the Process Manager daemon is restarted, it reloads its calendar information. At that time, any flow definitions or flows that reference deleted calendars can no longer be recognized by Process Manager. These flows appear in the error log. The flows themselves cannot run, and the flow definitions need to be changed to use an existing calendar, and resubmitted.

Configuring an Alarm

An alarm is used to send a notification when an exception occurs. The alarm definition specifies how a notification should be sent if an exception occurs. When a user defines a flow to schedule work, they can select an alarm to open if an exception occurs. They select an alarm from a configured list of alarms.

Alarms are configured by the Process Manager administrator. Each alarm must have a name and an email address.

Alarm definition

Alarms are stored in *JS_TOP*/work/alarms. Each alarm is in a separate file named *alarm_name*.alarm. The file name and its contents are case-sensitive. Each alarm can notify one or more email addresses.

The contents of an alarm file are as shown:

DESCRIPTION=<*description*>
NOTIFICATION=Email[*user1,user2,user3***]**

To configure an alarm:

1. As the Process Manager administrator, create a new file in *JS_TOP*/work/alarms. Specify a name for the file that is a meaningful name for the alarm, with a file suffix of alarm. For example:

   DBError.alarm

   The name you specify will appear in the Flow Editor in the list of available alarms.

2. Optional. Specify a meaningful description for the alarm. For example:

   DESCRIPTION=Send DBA a message indicating DBMS failure

3. Required. Specify one or more email addresses to notify regarding the exception. Separate the addresses with a comma. Specify the complete email address, or just specify the user name, if JS_MAILHOST was defined in js.conf. For example:

   NOTIFICATION=Email[bsmith,ajones]

   You must specify a valid notification statement with at least one email address, or the alarm is not valid.

4. To enable the alarm, reload the alarm list using the following command:

   jreconfigalarm

Viewing History

You can see the history of a work item, which shows details about when and how the item was run.

When you view history using the Flow Manager, or when you use the `jhist` command with no time interval specified, you see data for the past seven days.

To view the history of a flow definition:

For a flow definition, you can see the following information:

- If and when it was submitted
- If and when it was submitted to run immediately
- If and when it was removed from Process Manager
- If and when it was placed on hold or released
- If and when it was triggered by an event
- If and when a flow was created, and any IDs of those flows
- Time zone information for Process Manager Server

From the command line:

From the command line, run:

%**jhist -C flowdef -f** *flow_definition_name*

where *flow_name* is the name of the flow definition whose history you want to display.

To view the history of a flow:

For a flow, you can see the following information:

- When it started
- If and when it was killed
- If and when it was suspended
- If and when it was resumed
- When it completed
- Time zone information for Process Manager Server

From the command line

From the command line, run:

%**jhist -C flow -i** *flow_id*

where *flow_id* is the unique ID of the flow whose history you want to display.

To view the history of a job or job array:

For a job or job array, you can see the following information:

- The user name
- The ID of the flow in which it ran
- The job name
- The job ID
- The state of the job

- The status of the job
- When the job started
- When the job completed
- The CPU usage of the job
- The memory usage of the job
- Time zone information for Process Manager Server

From the command line

From the command line, run:

%**jhist -C job -j** *job_name*

where *job_name* is the name of the job or job array.

Troubleshooting

Process Manager daemon cannot restart—port is in use

The problem: If LSF is down, and the Process Manager daemon is killed or goes down before LSF comes back up, it is possible that one or more jobs were in the process of being submitted before the Process Manager Server went down. The processes for these jobs may be using the port the Process Manager daemon used before it went down.

The solution: Search for the bsub process of any job that Process Manager was trying to submit and kill it. The job will be resubmitted when the Process Manager Server restarts.

Overrun exception triggers at incorrect time

The problem An overrun exception is to trigger if a job runs longer than a specified number of minutes, for example 10 minutes. The overrun exception is flagged when the job runs for 9 minutes.

The solution The clock on the machine used to determine the start time of the job, and the clock on the machine on which the job is running are out of synchronization. Either adjust the overrun time to account for clock discrepancies, or synchronize the clocks on all machines.

After deleting a calendar, user cannot find flow

The problem The user deleted a calendar that was used, either to trigger a flow or to trigger a job within a flow. Then the Process Manager Server was restarted. After the Server restarts, the user cannot find the flow in the Flow Manager.

The solution Upon restart of the Process Manager Server, the flow is no longer associated with its flow definition in the Flow Manager. This is because the flow definition has an error. The flow now resides in the *JS_TOP*`/work/storage/error` directory.

Unable to run GUI on linux 2.2 through XTERM

The problem This problem is related to JRE defect #4466587. If the stack size is less than a certain limit on some linux platforms, a segmentation fault occurs.

The solution Increase the stack size to at least 2048. For tcsh or csh:

```
limit stacksize 2048
```

For bash:

```
ulimit -s 2048
```

User is unable to trigger their own flow

The problem

On Windows, if a user submits a flow under a user ID that is specified in one case, but logs in to Flow Manager with the same user ID typed in a different case, the Process Manager Server does not recognize the two user IDs as the same. The user cannot trigger the flow.

For example, when John creates a flow, he is logged in as jdoe. When he logs into Flow Manager to trigger the flow, he logs in as JDOE. To the Process Manager Server, he is not authorized to trigger this flow because it is not his.

The solution

A Windows user must always log in using the same case. The following are seen as different users:

- jdoe
- Jdoe
- JDOE

CHAPTER

3

Daemons

Contents

jfd

Process Manager Server daemon.

SYNOPSIS

jfd [**-2**|**-3**]

jfd [**-V**]

DESCRIPTION

`jfd` is responsible for managing flow definitions and flows. When a flow definition is submitted to Process Manager Server, `jfd` ensures that it is run according to its schedule or based on any triggering events, and manages any dependency conditions for each job in the flow before submitting the job to LSF for processing.

OPTIONS

-2

Specifies to run `jfd` as not daemonized, and log debug information to the log file specified in JS_LOGDIR. This option is used by failover. You cannot use it manually.

-3

Specifies to run `jfd` as not daemonized, and log debug information to `stderr` (normally the terminal). This option may be used for debugging purposes. Use only under the direction of Platform Technical Support.

-V

Prints the Process Manager release version to `stderr` and exits.

SEE ALSO

`fod`, `jadmin`

fod

Process Manager Server failover daemon. Only available with a full upgrade. Contact Platform Computing for more information.

SYNOPSIS

fod [**-***debug_level*] [**-d** *env_dir*] [**-C**]

fod [**-h**]|[**-V]**

DESCRIPTION

When used, `fod` is responsible for starting the Process Manager Server daemon `jfd`, and ensuring that it continues to run. `fod` monitors `jfd` and restarts it on the failover host if `jfd` fails.

SEE ALSO

`jfd`, `jadmin`

CHAPTER

5

Commands

Process Manager provides some administrative commands to help you maintain the system. Those administrative commands are listed in this chapter.

Contents

caleditor

starts the Calendar Editor.

SYNOPSIS

caleditor

You use the `caleditor` command to start the Calendar Editor, where you can create new calendars, edit or delete existing calendars.

EXAMPLES

`%` **caleditor**

opens the Calendar Editor.

floweditor

starts the Flow Editor.

SYNOPSIS

floweditor [*file_name*[*file_name* ...]]

DESCRIPTION

You use the `floweditor` command to start the Flow Editor. You can specify one or more flow definition file names to open automatically when the Flow Editor starts. You can use this as a shortcut to quickly open a flow definition for editing.

Your installation of Flow Editor allows you to create flow definitions but does not allow you to submit flow definitions. To enable full Flow Editor functionality, you must purchase an upgrade from Platform Computing. For more information, please contact Platform Computing at 1 877 444 4573.

OPTIONS

file_name

Specifies the name of the file to be opened when the Flow Editor starts. If you do not specify a file name, the Flow Editor starts with no files opened. You can specify a list of files by separating the file names with a space.

EXAMPLES

% **floweditor /tmp/myflow.xml /flows/payupdt.xml**

opens the Flow Editor, and opens `myflow.xml` and `payupdt.xml` at the same time.

% **floweditor**

opens the Flow Editor with no files opened.

flowmanager

starts the Flow Manager.

SYNOPSIS

flowmanager

DESCRIPTION

You use the `flowmanager` command to start the Flow Manager, which allows you to monitor and control existing flows.

EXAMPLE

`%` **flowmanager**

opens the Flow Manager.

jadmin

controls the Process Manager daemon jfd on UNIX.

SYNOPSIS

jadmin start|**stop**

jadmin [**-h**|**-V**]

DESCRIPTION

You use the `jadmin` command to start and stop the Process Manager daemon. You must be `root` to start the Process Manager daemon, and either `root` or the `primary` Process Manager administrator to stop the Process Manager daemon.

OPTIONS

start

Starts the Process Manager daemon on UNIX. Ensure LSF is up and running before you start the Process Manager daemon. You must be `root` to use this option.

stop

Stops the Process Manager daemon on UNIX. You must be `root` or the primary Process Manager administrator to use this option.

-h

Prints the command usage to stderr and exits.

-V

Prints the Process Manager release version to stderr and exits.

EXAMPLES

#**jadmin start**

Starts the Process Manager daemon.

#**jadmin stop**

Stops the Process Manager daemon.

SEE ALSO

`jfd, js.conf`

jalarms

lists the open alarms in Process Manager.

SYNOPSIS

jalarms [**-u** *user_name*|**-u all**] [**-f** *flow_name*|**-i** *flow_id*] [**-t** *start_time*,*end_time*]

jalarms [**-h**]|[**-V**]

DESCRIPTION

You use the `jalarms` command to display an open alarm or a list of the open alarms. The following information is displayed:

- alarm name
- user who owns the flow
- the date and time the alarm occurred
- alarm type
- Description of the problem that caused the alarm, if it was specified by the creator of the flow

OPTIONS

-u *user_name*

Specifies the name of the user who owns the alarm. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-u all**, information is displayed about alarms owned by all users.

-f *flow_name*

Specifies the name of the flow definition for which to display alarm information. Displays alarm information for flow definitions with the specified name.

-i *flow_ID*

Specifies the ID of the flow for which to display alarm information. Displays alarm information for flows with the specified ID.

-t *start_time,end_time*

Specifies the span of time for which you want to display the alarms. If you do not specify a start time, the start time is assumed to be the time the first alarm was opened. If you do not specify an end time, the end time is assumed to be now.

Specify the times in the format "*yyyy/mm/dd/HH:MM*". Do not specify spaces in the time interval string.

The time interval can be specified in many ways. For more specific syntax and examples of time formats, see "TIME INTERVAL FORMAT" on page 62.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

TIME INTERVAL FORMAT

You use the time interval to define a start and end time for collecting the data to be retrieved and displayed. While you can specify both a start and an end time, you can also let one of the values default. You can specify either of the times as an absolute time, by specifying the date or time, or you can specify them relative to the current time.

Specify the time interval is follows:

start_time`,`*end_time*|*start_time*`,`|`,`*end_time*|*start_time*

Specify *start_time* or *end_time* in the following format:

[*year*`/`][*month*`/`][*day*][`/`*hour*`:`*minute*|`/`*hour*`:`]|`.`|`.-`*relative_int*

Where:

- *year* is a four-digit number representing the calendar year.
- *month* is a number from 1 to 12, where 1 is January and 12 is December.
- *day* is a number from 1 to 31, representing the day of the month.
- *hour* is an integer from 0 to 23, representing the hour of the day on a 24-hour clock.
- *minute* is an integer from 0 to 59, representing the minute of the hour.
- `.` (period) represents the current month/day/hour:minute.
- `.-`*relative_int* is a number, from 1 to 31, specifying a relative start or end time prior to now.

start_time`,`*end_time*

Specifies both the start and end times of the interval.

start_time`,`

Specifies a start time, and lets the end time default to now.

`,`*end_time*

Specifies to start with the first logged occurrence, and end at the time specified.

start_time

Starts at the beginning of the most specific time period specified, and ends at the maximum value of the time period specified. For example, `3/` specifies the month of March—start March 1 at 00:00 a.m. and end at the last possible minute in March: March 31st at midnight.

ABSOLUTE TIME EXAMPLES

Assume the current time is May 9 17:06 2002:

`1,8` = May 1 00:00 2002 to May 8 23:59 2002

`,4` = the time of the first occurrence to May 4 23:59 2002

`6` = May 6 00:00 2002 to May 6 23:59 2002

`3/` = Mar 1 00:00 2002 to Mar 31 23:59 2002

`/12:` = May 9 12:00 2002 to May 9 12:59 2002

`2/1` = Feb 1 00:00 2002 to Feb 1 23:59 2002

`2/1,` = Feb 1 00:00 to the current time

`,.` = the time of the first occurrence to the current time

,2/10: = the time of the first occurrence to May 2 10:59 2002

2001/12/31,2002/5/1 = from Dec 31, 2001 00:00:00 to May 1st 2002 23:59:59

RELATIVE TIME EXAMPLES

.-9, = April 30 17:06 2002 to the current time

,.-2/ = the time of the first occurrence to Mar 7 17:06 2002

.-9,.-2 = nine days ago to two days ago (April 30, 2002 17:06 to May 7, 2002 17:06)

EXAMPLES

```
% jalarms -u all -t ".-7,."
```

displays all of the opened alarms for the last seven days.

jcadd

creates a calendar and adds it to the set of Process Manager calendars for the user.

SYNOPSIS

jcadd [**-d** *description*] **-t** "*cal_expression*" "*cal_name*"

jcadd [**-h**]|[**-V**]

DESCRIPTION

You use the `jcadd` command when you need to define a new time expression for use in scheduling either a flow or a work item within a flow. You define a new time expression by creating a calendar with that expression. The calendar is owned by the user who runs this command. You must define a calendar expression when you use this command.

OPTIONS

-d *description*

Specifies a description for the calendar. Specify a meaningful description for the calendar that summarizes the expression.

-t *cal_expression*

Specifies the dates on which you want some action to take place. You can enter specific dates, a range of dates, or a more complex expression that resolves to a series of dates. See "Creating Calendar Expressions" on page 65 for more information.

Note: If you want the calendars you create to be viewable in the Calendar Editor, specify abbreviated month and day names in all uppercase. For example: MON for Monday, MAR for March.

cal_name

Specifies the name of the calendar you are creating. Specify a unique name for the calendar. The first character cannot be a number. You can also use an underscore (_) in the calendar name.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

Limitations

Note that only merged calendars or calendar expressions with the following format can be viewed through the Calendar Editor graphical user interface:

`RANGE(startdate[, enddate]):PERIOD(1,*,step):occurrence`

Some examples that follow this format are:

```
RANGE(2001/1/1,2002/1/1):day(1,*,3)
RANGE(2001/1/1,2002/1/1):week(1,*,3):MON,TUE
RANGE(2001/1/1,2002/1/1):week(1,*,3):ABC(1)
RANGE(2001/1/1,2002/1/1):month(1,*,3):1,3,5
RANGE(2001/1/1,2002/1/1):month(1,*,3):MON(1),TUE(1)
RANGE(2001/1/1,2002/1/1):month(1,*,3):ABC(1)
RANGE(2001/1/1,2002/1/1):JAN:1||RANGE(2001/1/1,2002/1/1):JAN:2
ABC && DEF || HIJ
```

where ABC, DEF, HIJ are predefined calendars.

Creating Calendar Expressions

You can create several types of calendar expressions when you are creating or modifying a calendar. You use these expressions within system calendar definitions or calendars defined or modified using the `jcadd` or `jcmod` commands:

- Absolute dates
- Schedules that recur daily
- Schedules that recur weekly
- Schedules that recur monthly
- Schedules that recur yearly
- Combined calendars

To create absolute dates:

Specify the date in the following standard format:

(*yyyy/mm/dd*)

For example:

(2001/12/31)

Specify multiple dates separated by commas. For example:

(2001/12/31,2002/12/31)

To create schedules that recur daily:

Specify the expression in the following format:

RANGE(*startdate*[,*enddate*])**:day**(1,*,*step*)

The ending date is optional. If it is not specified, the calendar is valid indefinitely. For example:

RANGE(2003/2/1,2003/12/31):day(1,*,2)

In the above example, the expression is true every other day, beginning February 1, 2003, until December 31, 2003.

To create schedules that recur weekly:

Specify the expression in one of the following formats:

RANGE(*startdate*[,*enddate*])**:week**(1,*,*step*)**:***day_of_week*

where *step* is the interval between weeks and *day_of_week* is one or more days of the week, separated by commas. For example:

RANGE(2002/12/31):week(1,*,2):MON,FRI,SAT

or

RANGE(*startdate*[,*enddate*])**:week**(1,*,*step*)*:abc*(*ii*)

where step is the interval between weeks, *abc* is a previously defined calendar name and *ii* is an integer indicating a specific occurrence of a day within that calendar. For example:

RANGE(2002/01/01):week(1,*,3):MON(-1)

In the above example, MON(-1) refers to last Monday.

To create schedules that recur monthly:

Specify the expression in one of the following formats:

RANGE(*startdate*[,*enddate*])**:month**(1,*,*step*)*:day_of_month*

where *step* is the interval between months and *day_of_month* is one or more days of the month by number, separated by commas. For example:

RANGE(2002/12/31):month(1,*,2):1,15,30

or

RANGE(*startdate*[,*enddate*])**:***month*(1,*,*step*)*:abc*(*ii*)

where step is the interval between months, abc is a previously defined calendar name or built-in keyword and ii is an integer indicating a specific occurrence of a day within that calendar. For example:

RANGE(2002/01/01):month(1,*,3):MON(-1)

In the above example, MON(-1) refers to last Monday.

or

RANGE(*startdate*[,*enddate*])**:month**(1,*,*step*)**:***day_of_week*(*ii*)

where *step* is the interval between months, *day_of_week* is one or more days of the week separated by commas, and *ii* is an integer indicating a specific occurrence of a day within that calendar. For example:

RANGE(2002/01/01):month(1,*,3):MON(-1)

In the above example, MON(-1) refers to last Monday. For a list of built-in keywords, see "Built-in keywords—reserved words" on page 67.

To create schedules that recur yearly:

Specify the expression in the following format:

RANGE(*startdate*[,*enddate*])**:***month***:***day*

where *month* is the name of the month (JAN, FEB, MAR...DEC) and *day* is the day of the month (1,2,3...29,30,31). For example:

RANGE(2002/1/1,2004/12/31):JAN:1

To merge calendar expressions:

You can use Boolean logic to further qualify your schedule expressions. For example:

Mondays@Sys||Fridays@Sys && !Holidays@Sys

where Mondays@Sys, Fridays@Sys and Holidays@Sys are all predefined system calendars.

Built-in keywords—reserved words

Process Manager reserves words that are used as building blocks to create calendars. You cannot use these reserved words in a calendar name. However, you can use them within calendar expressions, and they are recognized by Process Manager. The following are the reserved words:

- apr, april, APR
- aug, august, AUG
- dates, DATES
- day, DAY
- dec, december, DEC
- feb, february, FEB
- fri, friday, FRI
- fy, FY
- h, HH
- jan, january, JAN
- jul, july, JUL
- jun, june, JUN
- m, MM
- mar, march, MAR
- may, MAY
- mon, monday, MON
- month, MONTH
- nov, november, NOV
- oct, october, OCT
- quarter, QUARTER
- range, RANGE
- sat, saturday, SAT
- sep, september, SEP
- sun, sunday, SUN
- thu, thursday, THU
- tue, tuesday, TUE
- wed, wednesday, WED
- yy, YY
- zzz, ZZZZ

EXAMPLES

```
% jcadd -d "Mondays but not holidays" -t "Mondays@Sys && !
Holidays@Sys" Mon_Not_Holiday
```

Creates a calendar called `Mon_Not_Holiday`. This calendar resolves to any Monday that is not a holiday, as defined in the `Holidays` system calendar.

```
% jcadd -d "Mondays, Wednesdays and Fridays" -t "Mondays@Sys
|| Wednesdays@Sys || Fridays@Sys" Everyotherday
```

Creates a calendar called `Everyotherday` that resolves to Mondays, Wednesdays and Fridays.

```
% jcadd -d "Monday to Thursday" -t "*:*:MON-THU" Shortweek
```

Creates a calendar called `Shortweek` that resolves to Mondays, Tuesdays, Wednesdays and Thursdays, every month.

```
% jcadd -d “Db report dates” -t “*:JAN,JUN,DEC:day(1)” dbrpt
```

Creates a calendar called `dbrpt` that resolves to the first day of January, June and December, every year.

SEE ALSO

`jcdel`, `jcals`

jcals

displays the list of calendars in Process Manager. The calendars are listed by owning user ID.

SYNOPSIS

jcals [**-l**] [**-u** *user_name*|**-u all**] [*cal_name*]

jcals [**-h**]|[**-V**]

DESCRIPTION

You use the `jcals` command to display information about one or more calendars. When using the default display option, the following information is displayed:

- user name
- calendar name
- the expression

OPTIONS

-l

Specifies to display the information in long format. In addition to the information listed above, this option displays the status of calendar (whether it is true today or not), the last date the calendar resolved to, the next date the calendar resolves to, and the calendar description.

-u *user_name*

Specifies the name of the user who owns the calendar. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-u all**, information is displayed about calendars owned by all users.

cal_name

Specifies the name of the calendar. If you do not specify a calendar name, all calendars meeting the other criteria are displayed.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

OUTPUT

The output without **-l**:

bhorner@curie-64: **jcals**

```
CALENDAR NAME  OWNER     EXPRESSION
payday         bhorner   (2002/5/15,2002/5/31,2002/6/14,2002/6/28)
mgmtreport     bhorner   (RANGE(2002/6/7):week(1,3,2):THU,FRI)
```

The output with **-l**:

bhorner@curie-65: **jcals -l**

```
CALENDAR: payday
  -- Pay days
OWNER                    TODAY    LAST CAL DATE     NEXT CAL DATE
bhorner                  false                      Wed May 15 2002
EXPRESSION: (2002/5/15,2002/5/31,2002/6/14,2002/6/28)
-------------------------------------------------------------------
CALENDAR: mgmtreport
  -- Management report days
OWNER                    TODAY    LAST CAL DATE     NEXT CAL DATE
bhorner                  false                      Fri Jun 7 2002
EXPRESSION: (RANGE(2002/6/7):week(1,3,2):THU,FRI)
```

EXAMPLES

```
% jcals -u all
```

Displays all calendars in Process Manager.

jcdel

deletes an existing calendar. You cannot delete a calendar that is currently in use by a flow definition or flow, or another calendar.

SYNOPSIS

jcdel [**-u** *user_name*] *cal_name*[*cal_name*...]

jcdel [**-h**]|[**-V**]

DESCRIPTION

You use the `jcdel` command to delete one or more calendars from Process Manager. You must be the owner of a calendar to delete it.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the calendar. If you do not specify a user name, the user name defaults to the user who invoked this command.

cal_name

Specifies the name of the calendar you are deleting. You can specify multiple calendar names by separating the names with a space.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jcdel -u "barneyt" Rundays2001
```

Deletes the calendar `Rundays2001` owned by the user barneyt.

SEE ALSO

`jcadd`, `jcals`

jcmod

edits an existing calendar. Using this command, you can change the calendar expression and the description of the calendar. You cannot modify a calendar that is in use by a flow definition or flow, or another calendar.

SYNOPSIS

jcmod [**-d** *description*] [**-u** *user_name*] [**-t** *cal_expression*] *cal_name*

jcmod [**-h**]|[**-V**]

DESCRIPTION

You use the `jcmod` command when you need to change either the calendar expression or the description of an existing calendar. You must be the owner of the calendar or be a Process Manager administrator to change a calendar.

OPTIONS

-d *description*

Specifies a description for the calendar. Specify a meaningful description for the calendar that summarizes the expression.

-u *user_name*

Specifies the name of the user who owns the calendar. If you do not specify a user name, the user name defaults to the user who invoked this command.

-t *cal_expression*

Specifies the dates on which you want some action to take place. You can enter specific dates, a range of dates, or a more complex expression that resolves to a series of dates. See "Creating Calendar Expressions" on page 72 for more information.

cal_name

Specifies the name of the calendar you are changing. You cannot change the name of the calendar.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

Creating Calendar Expressions

You can create several types of calendar expressions when you are creating or modifying a calendar. You use these expressions within system calendar definitions or calendars defined or modified using the `jcadd` or `jcmod` commands:

- Absolute dates
- Schedules that recur daily
- Schedules that recur weekly
- Schedules that recur monthly
- Schedules that recur yearly

- Combined calendars

To create absolute dates:

Specify the date in the following standard format:

(*yyyy/mm/dd*)

For example:

(2001/12/31)

Specify multiple dates separated by commas. For example:

(2001/12/31,2002/12/31)

To create schedules that recur daily:

Specify the expression in the following format:

RANGE(*startdate*[,*enddate*])**:day**(1,*,*step*)

The ending date is optional. If it is not specified, the calendar is valid indefinitely. For example:

RANGE(2003/2/1,2003/12/31):day(1,*,2)

In the above example, the expression is true every other day, beginning February 1, 2003, until December 31, 2003.

To create schedules that recur weekly:

Specify the expression in one of the following formats:

RANGE(*startdate*[,*enddate*])**:week**(1,*,*step*)**:***day_of_week*

where *step* is the interval between weeks and *day_of_week* is one or more days of the week, separated by commas. For example:

RANGE(2002/12/31):week(1,*,2):MON,FRI,SAT

or

RANGE(*startdate*[,*enddate*])**:week**(1,*,*step*)**:***abc*(*ii*)

where step is the interval between weeks, *abc* is a previously defined calendar name and *ii* is an integer indicating a specific occurrence of a day within that calendar. For example:

RANGE(2002/01/01):week(1,*,3):MON(-1)

In the above example, MON(-1) refers to last Monday.

To create schedules that recur monthly:

Specify the expression in one of the following formats:

RANGE(*startdate*[,*enddate*])**:month**(1,*,*step*)**:***day_of_month*

where *step* is the interval between months and *day_of_month* is one or more days of the month by number, separated by commas. For example:

RANGE(2002/12/31):month(1,*,2):1,15,30

or

RANGE(*startdate*[,*enddate*])**:***month*(1,*,*step*)**:***abc*(*ii*)

where step is the interval between months, abc is a previously defined calendar name or built-in keyword and ii is an integer indicating a specific occurrence of a day within that calendar. For example:

RANGE(2002/01/01):month(1,*,3):MON(-1)

In the above example, MON(-1) refers to last Monday.

or

RANGE(*startdate*[,*enddate*])**:month**(1,*,*step*)**:***day_of_week*(*ii*)

where *step* is the interval between months, *day_of_week* is one or more days of the week separated by commas, and *ii* is an integer indicating a specific occurrence of a day within that calendar. For example:

RANGE(2002/01/01):month(1,*,3):MON(-1)

In the above example, MON(-1) refers to last Monday. For a list of built-in keywords, see "Built-in keywords—reserved words" on page 74.

To create schedules that recur yearly:

Specify the expression in the following format:

RANGE(*startdate*[,*enddate*])**:***month*:*day*

where *month* is the name of the month (JAN, FEB, MAR...DEC) and *day* is the day of the month (1,2,3...29,30,31). For example:

```
RANGE(2002/1/1,2004/12/31):JAN:1
```

To merge calendar expressions:

You can use Boolean logic to further qualify your schedule expressions. For example:

```
Mondays@Sys||Fridays@Sys && !Holidays@Sys
```

where Mondays@Sys, Fridays@Sys and Holidays@Sys are all predefined calendars.

Built-in keywords—reserved words

Process Manager reserves words that are used as building blocks to create calendars. You cannot use these reserved words in a calendar name. However, you can use them within calendar expressions, and they are recognized by Process Manager. The following are the reserved words:

- apr, april, APR
- aug, august, AUG
- dates, DATES
- day, DAY
- dec, december, DEC
- feb, february, FEB
- fri, friday, FRI
- fy, FY
- h, HH
- jan, january, JAN
- jul, july, JUL
- jun, june, JUN
- m, MM
- mar, march, MAR
- may, MAY
- mon, monday, MON

- month, MONTH
- nov, november, NOV
- oct, october, OCT
- quarter, QUARTER
- range, RANGE
- sat, saturday, SAT
- sep, september, SEP
- sun, sunday, SUN
- thu, thursday, THU
- tue, tuesday, TUE
- wed, wednesday, WED
- yy, YY
- zzz, ZZZZ

EXAMPLES

```
% jcmod -d "Valentines Day" -u "barneyt" -t "*:Feb:14"
SpecialDays
```

Modifies a calendar called `SpecialDays`. This calendar resolves to February 14th every year.

jcomplete

acknowledges that a manual job is complete and specifies to continue processing the flow.

SYNOPSIS

jcomplete [**-d** *description*] [**-u** *user_name*] **-i** *flow_id* *flow_name*[**:***subflow_name*]**:***manual_job_name*

jcomplete [**-h**]|[**-V**]

DESCRIPTION

You use the `jcomplete` command to mark a manual job complete, to tell Process Manager to continue processing that part of the flow. Only the branch of the flow that contains the manual job is affected by the manual job—other branches continue to process as designed. You must be the owner of the manual job or a Process Manager administrator to complete a manual job.

OPTIONS

-d *description*

Describes the manual process completed. You can use this field to describe results of the process, or any pertinent comments.

-i *flow_id*

Specifies the ID of the flow in which the manual job is to be completed. This option is required to differentiate between multiple occurrences of the flow, ensuring the correct job is completed.

flow_name:subflow_name:manual_job_name

Specifies the name of the manual job to complete. Specify the fully-qualified manual job name, which is the flow name followed by the subflow name, if applicable, followed by the name of the manual job. For example:

`myflow:prtcheck:prtpage`

Specify the manual job name in the same format as it is displayed by the `jmanuals` command.

-u *user_name*

Specifies the name of the user who owns the manual job you are completing. If you do not specify a user name, user name defaults to the user who invoked this command.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jcomplete -d "printed check numbers 4002 to 4532" -i 42
payprt:checkprinter
```

completes the manual job `checkprinter` in the flow `payprt` with flow ID 42, and adds the comment "printed check numbers 4002 to 4532".

SEE ALSO

`jmanuals jjob`

jdefs

displays information about the flow definitions stored in Process Manager for the specified user.

SYNOPSIS

jdefs [**-l**] [**-u** *user_name*|**-u all**] [**-s** *status*] [*definition_name*[*definition_name* ...]]

jdefs [**-h**]|[**-V**]

DESCRIPTION

You use the `jdefs` command to display information about flow definitions and any associated flows. When using the default display option, the following information is displayed:

- user name
- flow name
- the status of the flow definition
- flow IDs of any associated flows
- the state of each flow

OPTIONS

-l

Specifies to display the information in long format. In addition to the information listed above, this option displays the following information:

- any events defined to trigger the flow
- any exit conditions specified in the flow definition

-u *user_name*

Specifies the name of the user who owns the flow definitions. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-u all**, information is displayed about flow definitions owned by all users.

-s *status*

Specifies to display information about only the flow definitions that have the specified status. The default is to display all flow definitions regardless of status. Specify one of the following values for status:

ONHOLD

Displays information about flow definitions that are on hold: these are definitions that are not currently eligible to trigger automatically.

RELEASE

Displays information about flow definitions that are not on hold. This includes any flow definitions that were submitted with events and flow definitions that were submitted to be triggered manually. This does not include flows that were submitted on an adhoc basis—to be run once, immediately.

definition_name

Specifies the name of the flow definition. If you do not specify a flow name, all flow definitions meeting the criteria are displayed. To specify a list of flow definitions, separate the flow definition names with a space.

-h

Prints the command usage to `stderr` and exits.

-v

Prints the Process Manager release version to `stderr` and exits.

OUTPUT

The output without `-l`:

```
bhorner@curie-62: jdefs
NAME            USER            STATUS          FLOW IDS
myflow          bhorner         Onhold          6(Running)
                                                7(Running)
pay1            bhorner         Onhold          2(Done)
                                                8(Running)
untitled1       bhorner         Onhold          1(Done)
```

The output with **`-l`**:

```
bhorner@curie-63: jdefs -l
NAME: myflow
  -- (No description)
USER            STATUS          FLOW IDS
bhorner         Onhold          6(Running)
                                7(Running)
Triggering Events:
    <None>
Exit Condition:
    All jobs complete successfully
-----------------------------------------------------------------
-------

NAME: pay1
  -- (No description)
USER            STATUS          FLOW IDS
bhorner         Onhold          2(Done)
                                8(Running)
Triggering Events:
    <None>
Exit Condition:
    All jobs complete successfully
-----------------------------------------------------------------
-------
```

```
NAME: untitled1
  -- (No description)
USER           STATUS         FLOW IDS
bhorner        Onhold         1(Done)
Triggering Events:
    <None>
Exit Condition:
    All jobs complete successfully
```

EXAMPLES

```
% jdefs -u barneyt -s RELEASE
```

Displays all flow definitions owned by barneyt that are not on hold.

jflows

displays information about the flows in Process Manager for the specified user. The information listed includes the current state of the flow.

SYNOPSIS

jflows [**-l**] [**-u** *user_name*|**-u all**] [**-f** *flow_name*] [**-s** *state*]

jflows [**-l**] [*flow_id*[*flow_id* ...]|**0**]

jflows [**-h**]|[**-V**]

DESCRIPTION

You use the `jflows` command to display information about one or more flows. When using the default display option, the following information is displayed:

- user name
- flow name
- flow ID
- the state of the flow
- start and end time for each flow

OPTIONS

-l

Specifies to display the information in long format. In addition to the information listed above, this option displays the states of all jobs, job arrays and subflows in the flow.

-u *user_name*

Specifies the name of the user who owns the flow. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-u all**, information is displayed about flows owned by all users.

-f *flow_name*

Specifies the name of the flow definition. If you do not specify a flow definition name, all flow definitions meeting the other criteria you specify are displayed. This option is mutually exclusive with the other options—if you specify a flow name, you cannot specify a flow ID.

-s *state*

Specifies to display information about only the flows that have the specified state. If you do not specify a state, flows of all states that meet the other criteria you specify are displayed. Specify one of the following values for state:

`Done`

Displays information about flows that completed successfully.

`Exit`

Displays information about flows that failed.

`Killed`

Displays information about flows that were killed.

`Running`

Displays information about flows that are running.

`Suspended`

Displays information about flows that were suspended.

`Waiting`

Displays information about flows that are waiting.

`flow_id`

Specify the ID number of the flow. If you do not specify a flow ID, all flows meeting the other criteria you specify are displayed. This option is mutually exclusive with the other options—if you specify a flow ID, you cannot specify a flow name. To specify a list of flows, separate the flow IDs with a space.

`0`

Specifies to display all flows.

`-h`

Prints the command usage to `stderr` and exits.

`-V`

Prints the Process Manager release version to `stderr` and exits.

OUTPUT

The output without **`-l`**:

```
bhorner@curie-66: jflows
```

```
ID        USER                NAME                STATE
1         bhorner             untitled1           Done
2         bhorner             pay1                Done
3         bhorner             untitled2           Running
4         bhorner             untitled2           Running
5         bhorner             pay2                Done
6         bhorner             myflow              Done
7         bhorner             myflow              Done
8         bhorner             pay1                Done
```

The output with **`-l`**:

```
bhorner@curie-67: jflows -l
```

```
FLOW ID: 2
USER        NAME         STATE       START TIME            END TIME
bhorner     pay1         Done        May 3 04:49:03 2005   May 3 04:52:20 2005
DETAILS:
NAME                                          TYPE      STATE       JOBID
2:bhorner:pay1:J1                             Job       Done        462
2:bhorner:pay1:J2                             Job       Done        463
2:bhorner:pay1:J3                             Job       Done        464
2:bhorner:pay1:J4                             Job       Done        465
2:bhorner:pay1:pay2                           SubFlow   Done
2:bhorner:pay1:pay2:J1                        Job       Done        466
2:bhorner:pay1:pay2:J2                        Job       Done        467
2:bhorner:pay1:pay2:J3                        Job       Done        468
```

```
2:bhorner:pay1:pay2:J4                              Job       Done        469
-----------------------------------------------
FLOW ID: 5
USER        NAME         STATE       START TIME            END TIME
bhorner     pay2         Done        May 6 12:00:21 2005   May 6 12:18:00 2005
DETAILS:
NAME                                                TYPE      STATE       JOBID
5:bhorner:pay2:J1                                   Job       Done        470
5:bhorner:pay2:J2                                   Job       Done        476
5:bhorner:pay2:J3                                   Job       Done        477
5:bhorner:pay2:J4                                   Job       Done        491
-----------------------------------------------
FLOW ID: 7
USER        NAME         STATE       START TIME            END TIME
bhorner     myflow       Done        May 6 12:00:48 2005   May 6 12:20:46 2005
DETAILS:
NAME                                                TYPE      STATE       JOBID
7:bhorner:myflow:J1                                 Job       Done        473
7:bhorner:myflow:J2                                 Job       Done        484
7:bhorner:myflow:J3                                 Job       Done        485
7:bhorner:myflow:J4                                 Job       Done        494
7:bhorner:myflow:A1                                 JobArray  Done        486
7:bhorner:myflow:mytestflow                         SubFlow   Done
7:bhorner:myflow:mytestflow:J1                      Job       Done        474
7:bhorner:myflow:mytestflow:J2                      Job       Done        487
7:bhorner:myflow:mytestflow:J3                      Job       Done        488
7:bhorner:myflow:mytestflow:J4                      Job       Done        495
7:bhorner:myflow:mytestflow:A1                      JobArray  Done        489
-----------------------------------------------
FLOW ID: 8
USER        NAME         STATE       START TIME            END TIME
bhorner     pay1         Done        May 6 12:01:00 2005   May 6 12:22:51 2005
DETAILS:
NAME                                                TYPE      STATE       JOBID
8:bhorner:pay1:J1                                   Job       Done        475
8:bhorner:pay1:J2                                   Job       Done        490
8:bhorner:pay1:J3                                   Job       Done        496
8:bhorner:pay1:J4                                   Job       Done        497
8:bhorner:pay1:pay2                                 SubFlow   Done
8:bhorner:pay1:pay2:J1                              Job       Done        498
8:bhorner:pay1:pay2:J2                              Job       Done        499
8:bhorner:pay1:pay2:J3                              Job       Done        500
8:bhorner:pay1:pay2:J4                              Job       Done        501
```

EXAMPLES

% jflows -f myflow

Displays all flows associated with the flow definition `myflow`.

jhist

displays historical information about Process Manager Server, calendars, flow definitions, flows, and jobs.

SYNOPSIS

```
jhist -C category[,category,...]
      [-u user_name|-u all]
      [-c  calendar_name]
      [-f flow_name] [-i flow_ID ] [-j job_name]
      [-t start_time,end_time]
jhist [-h|-V]
```

DESCRIPTION

You use the `jhist` command to display historical information about the specified object, such as a calendar, job or flow. You can display information about a single type of work item or multiple types of work items, for a single user or for all users.

If you do not specify a user name, `jhist` displays information for the user who invoked the command. If you do not specify a time interval, `jhist` displays information for the past 7 days, starting at the time the `jhist` command was invoked.

OPTIONS

`-C` *`category`*

Specifies the type of object for which you want to see history. Choose from the following values:

- **`alarm`**—displays historical information about one or more alarms
- **`calendar`**—displays historical information about one or more calendars
- **`daemon`**—displays historical information about Process Manager Server
- **`flowdef`**—displays historical information about one or more flow definitions
- **`flow`**—displays historical information about one or more flows
- **`job`**—displays historical information about one or more jobs or job arrays

You can specify more than one category by separating categories with a comma (,).

`-u` *`user_name`*

Displays information about categories owned by the specified user. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **`-u all`**, information is displayed about flows owned by all users.

`-t` *`start_time,end_time`*

Specifies the span of time for which you want to display the history. If you do not specify a start time, the start time is assumed to be 7 days prior to the time the `jhist` command is issued. If you do not specify an end time, the end time is assumed to be now.

Specify the times in the format "*yyyy/mm/dd/HH:MM*". Do not specify spaces in the time interval string.

The time interval can be specified in many ways. For more specific syntax and examples of time formats, see "TIME INTERVAL FORMAT" on page 86.

-c *calendar_name*

Specifies the name of the calendar for which to display historical information. If you do not specify a calendar name when displaying calendars, information is displayed for all calendars owned by the specified user.

Valid only when used with the `calendar` category.

-f *flow_name*

Specifies the name of the flow definition for which to display historical information. Displays flow definition, flow, or job information for flow definitions with the specified name.

Valid only with the `flowdef`, `flow`, and `job` categories.

-i *flow_ID*

Specifies the ID of the flow for which to display historical information. Displays flow and job information for flows with the specified ID.

Valid only with the `flow` and `job` categories.

-j *job_name*

Specifies the name of the job, job array or alarm to display historical information about. Displays information about the job, job array or alarm with the specified name.

Valid with the `job` or `alarm` categories.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

USAGE

-C alarm

Displays the time when the alarm was raised and the type and description of the alarm.

-C calendar

Displays the times when calendars are added or deleted.

-C daemon

Displays the server startup and shutdown times. These values are only displayed when root invokes `jhist` or the `-u root` option is used.

-C flowdef

Displays information about when a flow definition state is:

- Submit—When a flow definition is submitted
- SubmitAndRun—When a flow runs immediately
- Remove—When a flow definition is removed from the system
- Release—When a flow definition is released from on hold
- Hold—When a flow definition is placed on hold
- Trigger—When a flow definition is triggered manually or by an event

- Instantiate—When a flow is created

-C flow

Displays information about when a flow state is:

- Start—When a flow is started
- Kill—When a flow is killed
- Suspend—When a flow is suspended
- Resume—When a flow is resumed from the Suspended state
- Finished—When a flow is completed

-C job

Displays information about when a job or job array is:

- Started
- Killed
- Suspended
- Resumed
- Finished

TIME INTERVAL FORMAT

You use the time interval to define a start and end time for collecting the data to be retrieved and displayed. Although you can specify both a start and an end time, you can also let one of the values default. You can specify either of the times as an absolute time, by specifying the date or time, or you can specify them relative to the current time.

Specify the time interval is follows:

start_time`,`*end_time*|*start_time*`,`|`,`*end_time*|*start_time*

Specify *start_time* or *end_time* in the following format:

[*year*`/`][*month*`/`][*day*][`/`*hour*`:`*minute*|`/`*hour*`:`]|`.`|`.-`*relative_int*

Where:

- *year* is a four-digit number representing the calendar year.
- *month* is a number from 1 to 12, where 1 is January and 12 is December.
- *day* is a number from 1 to 31, representing the day of the month.
- *hour* is an integer from 0 to 23, representing the hour of the day on a 24-hour clock.
- *minute* is an integer from 0 to 59, representing the minute of the hour.
- `.` (period) represents the current month/day/hour:minute.
- `.-`*relative_int* is a number, from 1 to 31, specifying a relative start or end time prior to now.

start_time`,`*end_time*

Specifies both the start and end times of the interval.

start_time`,`

Specifies a start time, and lets the end time default to now.

`,`*end_time*

Specifies to start with the first logged occurrence, and end at the time specified.

start_time

Starts at the beginning of the most specific time period specified, and ends at the maximum value of the time period specified. For example, `3/` specifies the month of March—start March 1 at 00:00 a.m. and end at the last possible minute in March: March 31st at midnight.

ABSOLUTE TIME EXAMPLES

Assume the current time is May 9 17:06 2005:

`1,8` = May 1 00:00 2005 to May 8 23:59 2005

`,4` = the time of the first occurrence to May 4 23:59 2005

`6` = May 6 00:00 2005 to May 6 23:59 2005

`3/` = Mar 1 00:00 2005 to Mar 31 23:59 2005

`/12:` = May 9 12:00 2005 to May 9 12:59 2005

`2/1` = Feb 1 00:00 2005 to Feb 1 23:59 2005

`2/1,` = Feb 1 00:00 to the current time

`,.` = the time of the first occurrence to the current time

`,2/10:` = the time of the first occurrence to May 2 10:59 2005

`2001/12/31,2005/5/1` = from Dec 31, 2001 00:00:00 to May 1st 2005 23:59:59

RELATIVE TIME EXAMPLES

`.-9,` = April 30 17:06 2005 to the current time

`,.-2/` = the time of the first occurrence to Mar 7 17:06 2005

`.-9,.-2` = nine days ago to two days ago (April 30, 2005 17:06 to May 7, 2005 17:06)

OUTPUT

The following is a sample of the output when `jhist` is used:

`jhist -C flowdef,flow,job`

```
Mon Aug 12 17:00:01 2005 EST bhorner     Instantiated flow definition bhorner:testflow
                                         FlowId=30
Mon Aug 12 17:00:01 2005 EST bhorner     Start flow 30:bhorner:testflow
Mon Aug 12 17:00:01 2005 EST bhorner     Start job 30:bhorner:testflow:J1
Mon Aug 12 17:00:01 2005 EST bhorner     Instantiated flow definition bhorner:sample3
                                         FlowId=31
Mon Aug 12 17:00:01 2005 EST bhorner     Start flow 31:bhorner:sample3
Mon Aug 12 17:00:01 2005 EST bhorner     Start job 31:bhorner:sample3:J1
Mon Aug 12 17:00:07 2005 EST bhorner     Started job 30:bhorner:testflow:J1
                                         JobId=1189
Mon Aug 12 17:00:12 2005 EST bhorner     Started job 31:bhorner:sample3:J1
                                         JobId=1190
Mon Aug 12 17:00:17 2005 EST bhorner     Execute job 30:bhorner:testflow:J1
                                         JobId=1189
                                         Host=curie
Mon Aug 12 17:00:27 2005 EST bhorner     Execute job 31:bhorner:sample3:J1
                                         JobId=1190
                                         Host=curie
```

```
Mon Aug 12 17:00:27 2005 EST bhorner     Finished job 30:bhorner:testflow:J1
                                         JobId=1189
                                         State=Done
                                         Status=0
                                         StartTime=Mon Aug 12 17:00:04 2005 EST
                                         FinishTime=Mon Aug 12 17:00:25 2005 EST
                                          CPUUsage=0.19000
```

EXAMPLES

Display information about the calendar mycalendar and all flows for user1:

```
# jhist -C calendar,flow -u user1 -c mycalendar
```

Display information about the daemon and calendar for the past 30 days:

```
# jhist -C calendar,daemon -t .-30,. -u all
```

Display information for all flows with the name flow1, for user1 in the past week (counting 7 days back from today):

```
# jhist -C flow -u user1 -f flow1 -t .-7,.
```

Display information for all flows with the ID 231 for the past 3 days:

```
# jhist -C flow -i 231 -t .-3,.
```

Display information for all flows with the ID 231 and all related jobs from March 25, 2005 to March 31, 2005:

```
# jhist -C flow,job -i 231 -t 2005/3/25,2005/3/31
```

Display information for all flows with the ID 101 and all related jobs with the name myjob:

```
# jhist -C flow,job -i 101 -j myjob
```

Display information for all flows associated with the flow definition `myflow` and flows dated later than January 31, 2005

```
# jhist -C flowdef,flow -f myflow 2005/1/31,.
```

jhold

places a previously submitted flow definition on hold. No automatic events can trigger this definition until it has been explicitly released. Use this command when you want to temporarily interrupt automatic triggering of a flow. When a flow is on hold, it can still be triggered manually, such as for testing purposes.

SYNOPSIS

jhold [**-u** *user_name*] *flow_name*[*flow_name*...]

jhold [**-h**]|[**-V**]

DESCRIPTION

You use the `jhold` command to place a submitted flow definition on hold. This prevents it from being triggered automatically by any events. You must be the owner of a flow definition or the Process Manager administrator to place a flow definition on hold.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the flow. Use this option if you have administrator authority and you are holding the flow on behalf of another user. If you do not specify a user name, user name defaults to the user who invoked this command.

flow_name

Specifies the name of the flow definition. To specify a list of flow definitions, separate the flow definition names with a space.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jhold myflow
```

Places the flow definition `myflow`, which is owned by the current user, on hold.

```
% jhold -u "user01" payupdt
```

Places the flow definition `payupdt`, which is owned by user01, on hold.

SEE ALSO

`jrelease`

jid

displays the host name, version number and copyright date of the current Process Manager Server.

SYNOPSIS

jid [**-h** | **-v**]

DESCRIPTION

You use the `jid` command to verify the connection between Process Manager Client and Process Manager Server. If the command returns the host name of Process Manager Server, you have successfully connected to the server. If server failover is enabled, the `jid` command displays the host where the server is currently running.

OPTIONS

-h

Prints command usage to `stderr` and exits.

-v

Prints Process Manager release version to `stderr` and exits.

jjob

controls a job in a running flow.

SYNOPSIS

jjob [**-u** *user_name*] **-i** *flow_id* **-c** | **-k** | **-r** *flow_name*[**:***subflow_name*]**:***job_name*

jjob [**-h**]|[**-V**]

DESCRIPTION

You use the `jjob` command to kill or run a job, or mark a job complete. You must be the owner of the job or a Process Manager administrator or control administrator to control it.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the job you are controlling. If you do not specify a user name, user name defaults to the user who invoked this command.

-c

Specifies to mark the job complete. You can only complete a job in a flow that has exited. you use this option before rerunning a flow, to continue processing the remainder of the flow.

-k

Specifies to kill the job.

-r

Specifies to run or rerun the job.

-i *flow_id*

Specifies the ID of the flow containing the job to be controlled. This option is required to differentiate between multiple occurrences of the flow, ensuring the correct job is selected.

flow_name:subflow_name:manual_job_name

Specifies the name of the job to control. Specify the fully-qualified job name, which is the flow name followed by the subflow name, if applicable, followed by the name of the job. For example:

`myflow:print:prtreport`

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

`%` **jjob -i 42 -k payprt:report**

kill the job report in the flow `payprt` with flow ID 42.

SEE ALSO

`jmanuals`

jkill

kills a flow.

SYNOPSIS

jkill [**-u** *user_name*|**-u all**] [**-f** *flow_name*]

jkill *flow_id*[*flow_id* ...]|**0**

jkill [**-h**]|[**-V**]

DESCRIPTION

You use the `jkill` command to kill all flows, all flows belonging to a particular user, all flows associated with a flow definition, or a single flow. Any incomplete jobs in the flow are killed. Any work items that depend on the successful completion of this flow do not run. Only users with administrator authority can kill flows belonging to another user.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the flow. Use this option if you have administrator authority and you are killing the flow on behalf of another user. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-u all**, and you have administrator authority, you can kill flows belonging to all users.

-f *flow_name*

Specifies the name of the flow definition. Use this option if you want to kill all flows associated with the same flow definition. This option is mutually exclusive with the other options—if you specify a flow name, you cannot specify a flow ID.

flow_id

Specifies the ID of the flow you want to kill. Use this option if you want to kill one or more specific flow IDs. This option is mutually exclusive with the other options—if you specify a flow ID, you cannot specify a flow name. To specify a list of flow IDs, separate the flow IDs with a space.

0

Specifies to kill all flows.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jkill -f myflow
```

Kills all flows associated with the flow definition `myflow`. Does not affect the flow definition.

jmanuals

displays all manual jobs that have not yet been completed.

SYNOPSIS

jmanuals [**-i** *flow_ID*] [**-u** *username* | **-u all**] [**-f** *flow_definition*] [**-r yes** | **-r no**]

jmanuals [**-h**]|[**-V**]

DESCRIPTION

You use the `jmanuals` command to list the flows that contain manual jobs that have not yet been completed.

OPTIONS

-i *flow_ID*

Specifies the ID of the flow for which to display manual jobs.

-u *user_name*

Displays manual jobs in flows owned by the specified user. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-u all**, manual jobs are displayed for flows owned by all users.

-f *flow_definition*

Specifies the name of the flow definition for which to display manual jobs. Manual jobs are displayed for all flows associated with this flow definition.

-r yes

Specifies to display only those manual jobs that require completion at this time.

-r no

Specifies to display only those manual jobs that do not require completion at this time.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

OUTPUT

The following is a sample of the output when `jmanuals` is used:

```
bhorner@curie-68: jmanuals
ID       USER             NAME                  COMPLETION
REQUIRED
3        bhorner           untitled2:M1         Yes
4        bhorner           untitled2:Chkprt     Yes
```

SEE ALSO

`jcomplete`

jreconfigalarm

reloads the alarm definitions.

SYNOPSIS

jreconfigalarm [**-h**| **-V**]

DESCRIPTION

You use the `jreconfigalarm` command to reload the alarm definitions. You use this command to add or change alarm definitions without restarting Process Manager Server. You must be a Process Manager administrator to use this command.

OPTIONS

-h

Prints the command usage to `stderr` and exits.

-v

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

jreconfigalarm

Loads the updated list of Process Manager alarms.

SEE ALSO

`jadmin`

jrelease

releases a previously held flow definition.

SYNOPSIS

jrelease [**-u** user_name] *flow_name*[*flow_name*...]

jrelease [**-h**]|[**-V**]

DESCRIPTION

You use the `jrelease` command to release a submitted flow definition from hold. The flow definition is now eligible to be triggered automatically by any of its triggering events. Use this command when you want to resume automatic triggering of a flow.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the flow. Use this option if you have administrator authority and you are releasing the flow on behalf of another user. If you do not specify a user name, user name defaults to the user who invoked this command.

flow_name

Specifies the name of the flow definition. To specify a list of flow definitions, separate the flow definition names with a space.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jrelease myflow
```

Releases the flow definition `myflow`, which is owned by the current user, from hold.

```
% jrelease -u "user01" payupdt
```

Releases the flow definition `payupdt`, which is owned by user01, from hold.

SEE ALSO

`jhold`

jremove

removes a previously submitted flow definition from Process Manager.

SYNOPSIS

jremove [**-u** user_name] **-f** *flow_name*[*flow_name*...]

jremove [**-h**]|[**-V**]

DESCRIPTION

You use the `jremove` command to remove a submitted flow definition from Process Manager. Issuing this command has no impact on any flows associated with the definition, but no further flows can be triggered from it. Use this command when you no longer require this definition, or when you want to replace a definition that was created by a user ID that no longer exists. If you want to temporarily interrupt the automatic triggering of a flow, use the `jhold` command.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the flow. Use this option if you have administrator authority and you are removing the flow on behalf of another user. If you do not specify a user name, user name defaults to the user who invoked this command.

-f

Forces the removal of a flow definition that other flows have dependencies upon.

flow_name

Specifies the name of the flow definition. To specify a list of flow definitions, separate the flow definition names with a space.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jremove myflow
```

Removes the definition `myflow` from Process Manager. In this example, `myflow` is owned by the current user.

```
% jremove -u "user01" payupdt
```

Removes the definition `payupdt` from Process Manager. In this example, `payupdt` is owned by user01.

SEE ALSO

`jsub, jhold`

jrerun

reruns an exited flow.

SYNOPSIS

jrerun [**-v "***var*=*value*[**;***var1*=*value1***;**...]**"**] *flow_id*[*flow_id*...]

jrerun [**-h**]|[**-V]**

DESCRIPTION

You use the `jrerun` command to rerun a flow that has exited. The flow must have a state of Exit, and all jobs in the flow must be finished running before you can use this command. The flow is rerun from the first exited job, or jobs if the flow contains multiple branches that failed, and continues to process as designed. You must be the owner of a flow or a Process Manager administrator to use this command.

You cannot use this command to rerun a flow that was killed—you must trigger the flow again.

OPTIONS

-v *var*=*value*

Specifies to pass variables and their values to the flow when rerunning it. To specify a list of variables, separate the variable and value pairs with a semi-colon (;). The value of the variable is available only within the scope of the flow itself—local variables only.

flow_id

Specifies the ID of the flow to rerun. To specify a list of flows, separate the flow IDs with a space.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

`% ` **`jrerun 1234`**

reruns the flow with the flow ID 1234.

`% ` **`jrerun -v "USER=bhorner" 277`**

reruns the flow with the flow ID 277 and passes it a value of `jdoe` for the `USER` variable.

jresume

resumes a suspended flow.

SYNOPSIS

jresume [**-u** *user_name*|**-u all**] [**-f** *flow_name*]

jresume *flow_id*[*flow_id*...]|**0**

jresume [**-h**]|[**-V**]

DESCRIPTION

You use the `jresume` command to resume all flows, all flows belonging to a particular user, all flows associated with a particular flow definition, or a single flow. Only users with administrator authority can resume flows belonging to another user.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the flow. Use this option if you have administrator authority and you are resuming the flow on behalf of another user. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-u all**, and you have administrator authority, you can resume flows belonging to all users.

-f *flow_name*

Specifies the name of the flow definition. Use this option if you want to resume all suspended flows associated with the same definition. This option is mutually exclusive with the other options—if you specify a flow name, you cannot specify a flow ID.

flow_id

Specifies the ID of the flow you want to resume. Use this option if you want to resume one or more specific flow IDs. This option is mutually exclusive with the other options—if you specify a flow ID, you cannot specify a flow name. To specify a list of flow IDs, separate the flow IDs with spaces.

0

Specifies to resume all suspended flows.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jresume 14 17 22
```

Resumes the flows with IDs 14, 17 and 22.

```
% jresume 0
```

Resumes all suspended flows owned by the user invoking the command.

```
% jresume -u all
```

Resumes all suspended flows owned by all users.

SEE ALSO

`jstop`

jrun

triggers a flow definition from a file and runs the flow immediately without storing the flow definition in Process Manager.

SYNOPSIS

jrun [**-v "***var*=*value*[**;***var1*=*value1***;**...]**"**] *flow_file_name*

jrun [**-h**]|[**-V]**

DESCRIPTION

You use the `jrun` command when you want to trigger and run a flow immediately, without storing the flow definition within Process Manager. A flow ID is displayed when the flow is successfully submitted. This command is most useful for flows that run only once, or for testing a flow definition prior to putting it into production. You must be the owner of a flow definition or have Process Manager administrator authority to use this command.

OPTIONS

-v *var=value*

Specifies to pass variables and their values to the flow when running it. To specify a list of variables, separate the variable and value pairs with a semi-colon (;). The value of the variable is available only within the scope of the flow itself—local variables only.

flow_file_name

Specifies the name of the file containing the flow definition.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

`%` **jrun /flows/backup.xml**

Runs the flow defined in `/flows/backup.xml`. It does not store the definition of the flow in Process Manager.

`%` **jrun -v "USER=bsmith;YEAR=2003" /flows/payupdt.xml**

Runs the flow defined in `/flows/payupdt.xml`, and passes it a value of bsmith and 2003 for the `USER` and `YEAR` variables respectively. It does not store the definition of the flow in Process Manager.

jsetvars

substitutes values for local variables during the runtime of a flow.

SYNOPSIS

jsetvars -i *flow_ID variable*=*value* [*variable2*=*value2*...]

jsetvars [**-h**]|[**-V**]

DESCRIPTION

You use the `jsetvars` command to change the value of one or more local variables in a flow at runtime.

OPTIONS

-i *flow_ID*

Required. Specifies the ID of the flow in which to change the variable.

variable=*value*

Specifies the name of the variable and the value you are substituting. The variable must be a local variable, within the scope of the flow. You cannot change the value of a global variable using this command.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jsetvars -i 1234 priority=10
```

Changes the value of the priority variable to 10 for the flow with the ID 1234.

jsinstall

runs `jsinstall`, the Platform Process Manager installation and configuration script

SYNOPSIS

`jsinstall -f install.config`

`jsinstall -h`

DESCRIPTION

`jsinstall` runs the Platform Process Manager installation scripts and configuration utilities to install a new Process Manager component. You should install as root.

Before installing and configuring Process Manager, `jsinstall` checks the installation prerequisites, outputs the results to `prechk.rpt`, writes any unrecoverable errors to the `Install.err file` and exits. You must correct these errors before continuing to install and configure Process Manager.

During installation, `jsinstall` logs installation progress in the `Install.log` file, uncompresses, extracts and copies Process Manager files, installs a Process Manager license, and configures Process Manager Server.

jstop

suspends a running flow.

SYNOPSIS

jstop [**-u** *user_name*|**-u all**] [**-f** *flow_name*]

jstop *flow_id*[*flow_id*...]|**0**

jstop [**-h**]|[**-V**]

DESCRIPTION

You use the `jstop` command to suspend all flows, all flows belonging to a user, all flows associated with a flow definition, or a single flow. All incomplete jobs within the flow are suspended. Only users with administrator authority can suspend flows belonging to another user.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the flows. Use this option if you have administrator authority and you are suspending the flow on behalf of another user. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-u all**, and you have administrator authority, you can suspend flows belonging to all users.

-f *flow_name*

Specifies the name of the flow definition. Use this option if you want to suspend all flows associated with a particular flow definition. This option is mutually exclusive with the other options—if you specify a flow name, you cannot specify a flow ID.

flow_id

Specifies the ID of the flow you want to suspend. Use this option if you want to suspend one or more specific flow IDs. This option is mutually exclusive with the other options—if you specify a flow ID, you cannot specify a flow name. To specify a list of flow IDs, separate the flow IDs with a space.

0

Specifies to suspend all flows.

-h

Prints the command usage to `stderr` and exits.

-v

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jstop -f "myflow"
```

Suspends all flows associated with the definition `myflow`. Does not affect the flow definition.

```
% jstop 14
```

Suspends flow ID 14.

```
% jstop 0
```

Suspends all flows.

SEE ALSO

`jresume`

jsub

submits a flow definition to Process Manager.

SYNOPSIS

jsub [**-H**] [**-r**|**-d**] [[[**-T** *time_event*]...] [[**-F** "*file_event*"]...] [[**-p** "*proxy_event*"]...] [**-C** *combination_type*]] *flow_file_name*

jsub [**-h**]|[**-V**]

DESCRIPTION

You use the `jsub` command to submit a flow definition to Process Manager. When you submit the flow definition, you may specify the event that triggers the flow, if applicable. If you do not specify an event to trigger the flow, it requires a manual trigger. You must be the owner of the flow definition, or have Process Manager administrator authority to submit a flow definition.

Note: The flow definition you are submitting may contain pre-defined events that trigger the flow. When you submit this flow using the `jsub` command, those events are overwritten by any specified in the command. If the flow definition contains triggering events, and you submit the flow definition without specifying a triggering event, those events are deleted from the definition that is submitted, and the flow definition requires a manual trigger.

OPTIONS

-H

Submits the flow definition on hold. No automatic events can trigger this definition until it has been explicitly released. Use this option when the flow definition is complete, but you are not yet ready to start running flows on its defined schedule. When a definition is on hold, it can still be triggered manually, such as for testing purposes.

-r

Replace. Specifies that, if a flow definition with the same name already exists in Process Manager, it is replaced with the definition being submitted. If you do not specify `-r` and the flow definition already exists, the submission fails.

-d

Duplicate. Specifies that, if a flow definition with the same name already exists in Process Manager, a unique number is appended to the flow definition name to make it unique. The new name of the flow definition is displayed in the confirmation message when the flow definition is successfully submitted.

-T *time_event*

Specifies to automatically trigger a flow when the specified time events are true. Specify the time event in the following format:

[*cal_name*[**@***user_name*]**:**]*hour***:***minute*[**%***duration*]

cal_name

Specify the name of an existing calendar, which is used to calculate the days on which the flow runs. If you do not specify a calendar name, it defaults to Daily@Sys. If you do not specify a user name, the submitter's user name is assumed. Therefore, the calendar must exist under that user name.

hour:*minute*

Specify the time within each calendar day that the time event begins. You can specify the time in the following formats:

- hour:minutes—for example, `13:30` for 1:30 p.m. You can also specify the wildcard character * in the hour or minutes fields to indicate every hour or every minute, respectively.
- A list of hours, separated by commas—for example, `5,12,23` for 5:00 a.m., noon and 11:00 p.m.
- A range of numbers—for example, `14-17` for on the hour, every hour from 2:00 p.m. to 5:00 p.m.

The value you specify for *hour* must be a number between 0 and 23. The value for *minute* must be a number between 0 and 59. All numbers are values in the 24-hour clock.

%*duration*

Specify the number of minutes for which the time event should remain valid after it becomes true. After the duration expires, the event can no longer trigger any activity. The default duration is 1 minute. The minimum duration you can specify is also 1 minute.

-F "*file_event***"**

Specifies to automatically trigger a flow when the specified file events are true.

When specifying the file name, you can also specify wildcard characters: ***** to represent a string or **?** to represent a single character. For example, **a*.dat*** matches `abc.dat`, `another.dat` and `abc.dat23`. **S??day*** matches `Satdays.tar` and `Sundays.dat`. ***e** matches `smile`.

Note: There are some differences between UNIX and Windows when using wildcard characters. Because UNIX is case-sensitive and Windows is not, if you specify **A***, on UNIX it matches only files beginning with `A`. On Windows, it matches files beginning with `A` and `a`. Also, on UNIX, if you specify **??**, it matches exactly two characters. On Windows, it matches one or two characters. These behaviors are consistent with UNIX `ls` command behavior, and Windows `dir` command behavior.

Specify the file event in one of the following formats:

arrival(*file_location***)**

Trigger a flow when the specified file arrives in the specified location, and subsequently only if the file is deleted and arrives again. This option looks for a transition from nonexistence of the file to existence. When the file is on a shared file system, specify the file location in the following format:

absolute_directory/*filename*

exist(*file_location***)**

Trigger a flow if the specified file exists in the specified location, and continue to trigger the flow every time the test for the file is performed, as long as the file continues to exist. When the file is on a shared file system, specify the file location in the following format:

absolute_directory/filename

! exist(*file_location***)**

Trigger a flow if the specified file does not exist in the specified location, and continue to trigger the flow every time the test for the file is performed, as long as the file does not exist. When the file is on a shared file system, specify the file location in the following format:

absolute_directory/filename

size(*file_location***)** *operator size*

Trigger a flow when the size of the file meets the criteria specified with *operator* and *size*. When the file is on a shared file system, specify the file location in the following format:

absolute_directory/filename

Valid values for operator are: >, <, >=, <=, == and `!=`.

Note: For csh, if you specify **!=** (not equal), you need to precede the operator with a backslash escape character.

Specify the size in bytes.

age(*file_location***)** *operator age*

Trigger a flow when the age of the file meets the criteria specified with *operator* and *age*.

When the file is on a shared file system, specify the file location in the following format:

absolute_directory/filename

Valid values for operator are: >, <, >=, <=, == and `!=`.

Note: For csh, if you specify **!=** (not equal), you need to precede the operator with a backslash escape character.

Specify the age in minutes.

-p "*proxy_event***"**

Specifies to automatically trigger a flow when the specified proxy event is true.

Specify the proxy event in one the following formats:

job(exit|done|start|end(*user_name***:***flow_name***:**[*subflow_name***:]***job_name***)** [*operator value*]**)**

Trigger a flow when the specified job meets the specified condition. You must specify the user name to fully qualify the flow containing the job. You only specify a subflow name if the job is contained within a subflow.

Valid operators are **>=**, **>**, **<=**, **<**, **!=** and **==**.

Note: For csh, if you specify **!=** (not equal), you need to precede the operator with a backslash escape character.

Example: on successful completion of J1:

```
-p "job(done(jdoe:myflow:J1))"
```

Example: if payjob exits with an exit code greater than 5:

```
-p "job(exit(jdoe:myflow:testflow:payjob)>5)"
```

jobarray(exit|done|end|numdone|numexit|numend|numstart(*user_name***:***flow_name***:[***subflow_name***:]***job_array_name* **)[***operator value***])**

Trigger a flow when the specified job array meets the specified condition. You must specify the user name to fully qualify the flow containing the job array. You only specify a subflow name if the job array is contained within a subflow.

Valid operators are **>=**, **>**, **<=**, **<**, **!=** and **==**.

Example: on successful completion of all jobs in Array1:

```
-p "jobarray(done(jdoe:myflow:Array1))"
```

Example: if arrayjob exits with an exit code greater than 5:

```
-p "jobarray(exit(jdoe:myflow:testflow:arrayjob)>5)"
```

Example: if more than 3 jobs in A1 exit:

```
-p "jobarray(numexit(jdoe:myflow:testflow:arrayjob)>3)"
```

flow(exit|done|end|numdone|numexit|numstart(*user_name***:***flow_name***:[***subflow_name***])[***operator value***])**

Trigger a flow when the specified flow or subflow meets the specified condition. You must specify the user name to fully qualify the flow. Specify a subflow name if applicable.

Valid operators are **>=**, **>**, **<=**, **<**, **!=**, **==**.

Example: on successful completion of all jobs in `myflow`:

```
-p "flow(done(jdoe:myflow))"
```

Example: if `myflow` exits with an exit code greater than 5:

```
-p "flow(exit(jdoe:myflow)>5)"
```

Example: if more than 3 jobs in the subflow testflow exit:

```
-p "flow(numexit(jdoe:myflow:testflow)>3)"
```

Note: When Process Manager calculates the number of jobs in a flow, for successful jobs, failed jobs, and so on, it does not count the jobs in a subflow, and it counts a job array as a single job. It also does not count other objects in the flow, such as events or alarms.

-f "*flow_event***"**

Specifies to automatically trigger a flow when the specified flow event(s) are true.

Specify the flow event in one of the following formats:

done(*flow_definition_name***)**

Trigger a flow when the specified flow completes successfully. Specify the flow definition name as follows:

*user_name***:***flow_definition*

If you do not specify a user name, it defaults to your own.

end(*flow_definition_name***)**

Trigger a flow when the specified flow ends, regardless of exit code. Specify the flow definition name as follows:

*user_name***:***flow_definition*

If you do not specify a user name, it defaults to your own.

numdone(*flow_definition_name***)** *operator nn*

Trigger a flow when the specified number of jobs in the specified flow complete successfully. Specify the flow definition name as follows:

*user_name***:***flow_definition*

If you do not specify a user name, it defaults to your own.

Valid operators are **>=**, **>**, **<=**, **<**, **!=**, **==**.

For example:

numdone(jdoe:payflow)>=5

will trigger the flow you are submitting when 5 jobs complete successfully in payflow.

numstart(*flow_definition_name***)** *operator nn*

Trigger a flow when the specified number of jobs in the specified flow have started. Specify the flow definition name as follows:

*user_name***:***flow_definition*

If you do not specify a user name, it defaults to your own.

Valid operators are **>=**, **>**, **<=**, **<**, **!=**, **==**.

numexit(*flow_definition_name***)** *operator nn*

Trigger a flow when the specified number of jobs in the specified flow exit. Specify the flow definition name as follows:

*user_name***:***flow_definition*

If you do not specify a user name, it defaults to your own.

Valid operators are **>=**, **>**, **<=**, **<**, **!=**, **==**.

For example:

numexit(jdoe:payflow)>=3

will trigger the flow you are submitting if more than 3 jobs in payflow exit.

exit(*flow_definition_name***)** *operator nn*

Trigger a flow when the specified flow ends with the specified exit code. Specify the flow definition name as follows:

*user_name***:***flow_definition*

If you do not specify a user name, it defaults to your own.

Valid operators are **>=**, **>**, **<=**, **<**, **!=**, **==**.

For example:

exit(jdoe:payflow)>=2

will trigger the flow you are submitting if payflow has an exit code greater than or equal to 2.

Note: When Process Manager calculates the number of jobs in a flow, for successful jobs, failed jobs, and so on, it does not count the jobs in a subflow, and it counts a job array as a single job. It also does not count other objects in the flow, such as events or alarms.

-C *combination_type*

When multiple events are specified, the combination type specifies whether one event is sufficient to trigger a flow, or if all of the events must be true to trigger it. The default is all.

AND

Specifies that all events must be true before a flow is triggered. This is the default.

OR

Specifies that a flow will trigger when any event is true.

flow_file_name

Specifies the name of the file containing the flow definition.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jsub -r -T "Weekends@Sys:0-8:30%30" -F "exists(/tmp/1.dat)"
-C AND myflow.xml
```

Submits the flow definition in `myflow.xml`, to be triggered when both of the following are true:

- Saturdays and Sundays every hour on the half hour, beginning at midnight until 8:00 a.m.
- The file `/tmp/1.dat` exists

Any triggering information defined within the flow definition is overwritten. If this flow definition already exists, replace it.

```
% jsub -d -F "size(/data/tmp.log) >3500000" -F
"arrival(/tmp/1.dat)" -C OR backup.xml
```

Submits the flow definition in `backup.xml`, to be triggered when one of the following is true:

- The size of `/data/tmp.log` exceeds 3.5 MB

- The `file /tmp/1.dat` arrives

Any triggering information defined within the flow definition is overwritten. If this flow definition already exists, create a duplicate.

jtrigger

manually triggers a previously submitted flow definition.

SYNOPSIS

jtrigger [**-u** *user_name*] [**-v "***var***=***value*[**;***var1***=***value1***;**...]**"**] *flow_name*[***f****low_name*...]

jtrigger [**-h**]|[**-V**]

DESCRIPTION

You use the `jtrigger` command to trigger a submitted flow definition, which creates a flow associated with that definition. Any events normally used to trigger this definition are ignored at this time.

If the flow definition is on hold, you can use this command to trigger a flow. If the flow definition is not on hold, this command triggers an additional execution of the flow. If you want to trigger a flow whose definition is not yet stored in Process Manager, use the `jrun` command.

OPTIONS

-u *user_name*

Specifies the name of the user who owns the flow definition. Use this option if you have administrator authority and you are triggering the flow on behalf of another user.

-v *var*=*value*

Specifies to pass variables and their values to the flow when triggering it. To specify a list of variables, separate the variable and value pairs with a semi-colon (;). The value of the variable is available only within the scope of the flow itself—local variables only.

flow_name

Specifies the name of the flow definition. To specify a list of flow definitions, separate the flow definition names with a space.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Process Manager release version to `stderr` and exits.

EXAMPLES

```
% jtrigger myflow
```

Triggers the flow definition `myflow`, which is owned by the current user.

```
% jtrigger -u "user01" payupdt
```

Triggers the flow definition `payupdt`, which is owned by user01.

```
% jtrigger -v "PMONTH=October" payflow
```

Triggers the flow definition `payflow`, which is owned by the current user, and passes it a value of October for the variable `PMONTH`.

SEE ALSO

`jrun`

CHAPTER

35

Files

This chapter describes the Process Manager file structure, and provides descriptions and formats of those files you may be required to change while administering Process Manager.

Contents

File Structure

When Process Manager is installed, it creates several directories under its top directory. Some of these directories contain scheduling data, others contain working files, or historical data. Some directories are created when the Process Manager server is started, rather than immediately after installation.

Files created on the server host

The following show the file structure created during installation. The directories on the left are those that exist on UNIX after the Process Manager server has been started. The directories on the right are those that exist on a Windows server after installation is complete:

The following describes what each directory contains:

Directory	Contents
`3.0/app`	Contains the files required to run Process Manager Client.
`3.0/bin`	Contains the executables for all of the Process Manager commands and the Process Manager Client applications.
`3.0/etc`	Contains the Process Manager messages and the data specification used by the Process Manager software when creating flows.
`3.0/examples`	Contains example flows you can use and customize.
`3.0/jre`	On Windows only, contains the Java runtime environment files for the client applications.
`3.0/install`	On UNIX only, contains the Process Manager README file and `install.config` and other installation-specific information.
`3.0/lib`	Contains the Process Manager Java files.
`3.0/lresources`	Contains the properties files used by Process Manager.
`3.0/man`	On UNIX only, contains the man pages for each of the Process Manager commands.
`conf`	Contains the configuration files used by the install script to define the Process Manager environment, including `js.conf` and `fod.conf`, (if failover is installed) `cshrc.js` and `profile.js`.
`log`	Contains the log files created by Process Manager to store Process Manager Server and failover error logs. Process Manager creates a log file called `jfd.log.`*`hostname`*, which contains the error logs.
`work`	Contains working information required by Process Manager to complete its processing, including the following directories: ◆ `alarms`—contains all alarm definitions ◆ `calendar`—contains all system calendar definitions ◆ `history`—contains all historical data ◆ `storage`—contains copies of active and completed flows ◆ `templates`—contains templates for inserting custom applications in a flow ◆ `var_comm`—contains temporary values for user variables ◆ `variable`—contains the current values of any global or local user variables

Process Manager history files

The log files containing Process Manager audit data are located in *JS_TOP*/work/history. Process Manager writes audit data to a history file called history.log.1. When the file reaches the maximum size specified in the configuration file js.conf (the default is 500 KB), a new file is created, and the suffix is incremented by 1. Periodically, you may want to manually archive or delete these files. To change the frequency at which a new history log file is created, see "To change the history setting:" on page 42.

Process Manager log files

Process Manager creates a log file called jfd.log.*hostname*, which contains the error logs. The file is located within the directory defined by the JS_LOGDIR configuration setting in js.conf. By default, this directory is *JS_TOP*/log. However, after installation, you can change the value in js.conf to use a different directory.

history.log.*n*

Process Manager Server stores audit data in a history log file. This log file contains a record of all of the work items that run in the system. It tracks each work item as it enters the Process Manager system, is submitted to LSF, and tracks its state as it completes. It records the CPU usage of each job in the system, start time, finish time, and other pertinent information.

When the history log file reaches the maximum size specified in JS_HISTORY_SIZE or the maximum number of hours of data, as specified in JS_HISTORY_LIFETIME in the `js.conf` file, a new history log file is created. The numeric suffix of the file increases as each new file is created.

Example

The following is an excerpt from a history log file:

```
"JOB" "bhorner" "1035277212" "5:bhorner:daily:J1" "Started job" "JobId=1360"
"JOB" "bhorner" "1035277222" "5:bhorner:daily:J1" "Execute job" "JobId=1360|Host=curie"
"JOB" "bhorner" "1035277242" "5:bhorner:daily:J1" "Finished job"
"JobId=1360|State=Done|Status=0|StartTime=1035277208|FinishTime=1035277237|CPUUsage=0.170000 sec"
"FLOW" "bhorner" "1035277242" "5:bhorner:daily" "Finished flow"
"State=Done|Status=0|StartTime=1035277202|FinishTime=1035277242"
"FLOWDEF" "bhorner" "1035309105" "bhorner:untitled1" "Remove flow definition" ""
"FLOWDEF" "bhorner" "1035309105" "bhorner:untitled1" "Submit flow definition" ""
"FLOWDEF" "bhorner" "1035309127" "bhorner:untitled1" "Instantiated flow definition" "FlowId=6"
"FLOWDEF" "bhorner" "1035309127" "bhorner:untitled1" "Trigger flow definition" ""
"FLOW" "bhorner" "1035309127" "6:bhorner:untitled1" "Start flow" ""
```

Description

Data in the file is listed from top (earliest events) to bottom (latest events).

In the above example, the first line shows when J1 in the flow `daily` was submitted to LSF. The second line indicates when LSF dispatched the job, and the name of the host to which it was dispatched. When the job completes, the job ID and its resulting state and CPU usage are listed, as shown in the third line.

install.config

Process Manager configuration file for installation on UNIX or Linux. Run `jsinstall -f install.config` to install Process Manager using the options specified in `install.config`.

Template location

A template `install.config` is located in the installation script directory created when extracting the Process Manager installation script tar file. Edit the file to specify the options for your Process Manager installation.

Format

Each entry in `install.config` has one of the following formats:

```
NAME=VALUE
NAME=
NAME="STRING1 STRING2 ..."
```

The equal sign `(=)` must follow each `NAME` even if no value follows and there should be no space beside the equal sign.

Lines starting with a pound sign (#) are comments and are ignored. Do not use `#if` as this is reserved syntax.

Parameters

JS_ADMINS

Syntax **`JS_ADMINS=`***primary_admin*[*admin2 admin3* ...]

Description REQUIRED.

Specifies the administrators who run Process Manager. The first entry is the primary Process Manager administrator, and must be a valid user ID. This name is set at installation time. Any additional administrators specified can be user IDs or UNIX user group names. To specify a list, separate the names with a space.

Default There is no default for this parameter. A value for the primary Process Manager administrator is set at installation time.

JS_CONTROL_ADMINS

Syntax **`JS_CONTROL_ADMINS=`***cadmin*[*cadmin1 cadmin2* ...]

Description OPTIONAL.

Specifies one or more control administrators who can control any flows or jobs in the Process Manager system, regardless of who the owner is. These administrators cannot submit or remove flows belonging to other users.

Any administrators specified can be user IDs or UNIX user group names.

To specify a list, separate the names with a space.

Default There is no default for this parameter.

See also JS_ADMINS

JS_FAILOVER

Syntax **JS_FAILOVER=false|true**

Description OPTIONAL if failover is not used. REQUIRED if failover is used.

Specifies that the failover feature is to be enabled. The failover feature provides automatic failover in the event the Process Manager Server host becomes unavailable.

Default The default is JS_FAILOVER=false—no failover.

See also JS_FAILOVER_HOST, JS_FOD_PORT

JS_FAILOVER_HOST

Syntax **JS_FAILOVER_HOST=***hostname*

Description OPTIONAL if failover is not used. REQUIRED if failover is used.

Specifies the fully-qualified hostname of the failover host.

If you specified JS_FAILOVER=true, specify the name of the host where Process Manager Server will run if the primary Process Manager Server host is unavailable.

Default The default is the same hostname as that specified for Process Manager Server.

See also JS_FAILOVER, JS_FOD_PORT

JS_FOD_PORT

Syntax **JS_FOD_PORT=***number*

Description OPTIONAL if failover is not used. REQUIRED if failover is used.

Specifies the port number of the failover daemon `fod`.

If you specified JS_FAILOVER=true, specify the port number to be used for communication between the failover daemon and the Process Manager Server daemon.

Default The default is 1999.

See also JS_FAILOVER, JS_FAILOVER_HOST

JS_TOP

Syntax **JS_TOP=***/path*

Description REQUIRED.

Specifies the full path to the top-level installation directory.

Corresponds to JS_HOME in `js.conf`.

Default There is no default for this parameter.

JS_HOST

Syntax **JS_HOST=***hostname*

Description REQUIRED.

Specifies the fully-qualified domain name of the host on which Process Manager Server runs—the name of the host to which the clients connect under normal operations. You cannot specify more than one host.

Default There is no default for this parameter.

See also JS_PORT

JS_LICENSE

Syntax JS_LICENSE=*/path/filename*

Description Specifies the location of the copy that Process Manager makes of the `license.dat` file.

Default The default is the parent directory of the current working directory where `jsinstall` is run.

JS_MAILHOST

Syntax **`JS_MAILHOST=`***hostname*

Description OPTIONAL.

Specifies the name of the mail server host.

On Windows, specify the protocol and name of the mail server host. For an SMTP mail host, specify **`SMTP:`***hostname*. For an exchange mail host, specify **`Exchange:`***hostname*.

On UNIX, specify just the name of the mail server host.

Default If Process Manager Server is installed on Windows, the default is Exchange:*localhostname*. If Process Manager Server is installed on UNIX, the default is *localhostname*.

JS_PORT

Syntax **`JS_PORT=`***number*

Description REQUIRED.

Specifies the port number to be used by Process Manager Client to connect with Process Manager Server.

Default The default port number is 1966.

See also JS_HOST

JS_TARDIR

Syntax **`JS_TARDIR=`***/path*

Description OPTIONAL.

Specifies the full path to the directory containing the Process Manager distribution files to be installed.

Default The default is the parent directory of the current working directory where `jsinstall` is run.

LSF_ENVDIR

Syntax **LSF_ENVDIR=***/path*

Description REQUIRED.

Default Specifies the directory where LSF configuration files are stored. There is no default for this value.

js.conf

configuration file for Process Manager. Process Manager Server receives its configuration information on startup from its configuration file `js.conf`. The file `js.conf` is created automatically during the installation of Process Manager. The values in `js.conf` are set automatically when you install Process Manager Server as follows:

- On UNIX, from the values you specify in `install.config` before running `jsinstall`
- On Windows, from the values you specify when prompted by the installation program
- Some values default during installation

If, for example, when you installed the failover daemon, the default port was already in use, you can change that value directly in `js.conf`. The next time Process Manager Server is started, the new values take effect.

Some values in `js.conf` are generated and cannot be changed without causing problems. This is indicated in the parameter description.

Format

Each entry in `js.conf` has one of the following formats:

```
NAME=VALUE
NAME=
NAME="STRING1,STRING2,..."
```

The equal sign `(=)` must follow each `NAME` even if no value follows and there should be no space beside the equal sign.

Lines starting with a pound sign (#) are comments and are ignored. Do not use `#if` as this is reserved syntax.

Parameters

JS_ADMINS

Syntax **`JS_ADMINS=`***primary_admin*[,*admin2*,*admin3*,...]

Description REQUIRED.

Specifies the administrators who run Process Manager. The first entry is the primary Process Manager administrator, and must be a valid user ID. This name is set at installation time. Any additional administrators specified can be user IDs or UNIX user group names. To specify a list, separate the names with a comma without any space.

Default There is no default for this parameter. A value for the primary Process Manager administrator is set at installation time.

JS_ALARMS_DIR

Syntax **`JS_ALARMS_DIR=`***/path*

Description Specifies the directory where the configured alarms are stored.

Default The default is `JS_HOME/work/alarms`.

JS_CACHE_LIFETIME

Syntax **JS_CACHE_LIFETIME=***hours*

Description Specifies the maximum time in hours for which cache data is collected before a new cache file is created. A new cache file is created when the size of the cache file exceeds the size specified in JS_CACHE_SIZE, or the number of hours specified here is reached, whichever comes first.

Default The default is 168 hours (7 days).

See also JS_CACHE_SIZE

JS_CACHE_SIZE

Syntax **JS_CACHE_SIZE=***bytes*

Description Specifies the maximum number of bytes of cache data that is collected before a new cache file is created. A new cache file is created when the size of the cache file exceeds the size specified in JS_CACHE_SIZE, or the number of hours specified here is reached, whichever comes first.

Default The default is 10 MB.

See also JS_CACHE_LIFETIME

JS_CALENDAR_DIR

Syntax **JS_CALENDAR_DIR=***/path*

Description Specifies the directory where the calendars are stored.

Default The default is JS_HOME/work/calendar.

JS_TIME_ZONE

Syntax **JS_TIME_ZONE=client|server|UTC**

Description Specifies client time zone. Case sensitive.

Default The default is server.

JS_CONN_TIMEOUT

Syntax **JS_CONN_TIMEOUT=***seconds*

Description Specifies the maximum number of seconds a Process Manager Client waits for a response from Process Manager Server.

Default The default is 1024 seconds.

JS_CONTROL_ADMINS

Syntax **JS_CONTROL_ADMINS=***cadmin*[,c*admin1*,c*admin2*,...]

Description OPTIONAL.

Specifies one or more control administrators who can control any flows or jobs in Process Manager, regardless of who the owner is. These administrators cannot submit or remove flows belonging to other users.

Any administrators specified can be user IDs or UNIX user group names.

To specify a list, separate the names with a comma without any space.

Default There is no default for this parameter.

See also JS_ADMINS

JS_DATACAPTURE_TIME

Syntax **JS_DATACAPTURE_TIME="***cal_name***@***user_name***:***hour***[:***minute***]"**

Description Periodically, Process Manager Server interrupts its processing to take an image of the workload in Process Manager, and saves it for recovery purposes. Depending on the amount of workload that passes through your server, recovery of Process Manager following an outage may take some time. The more recent the system image, the shorter the recovery time.

JS_DATACAPTURE_TIME specifies the schedule that determines when an image of the workload in the system is saved for recovery purposes. The schedule is specified in the form of a calendar name and owner and time, and is enclosed in double quotes. You can specify one or more schedules in a comma-separated list.

During data capture, Process Manager Server does not submit new work. Ideally, schedule this activity at a time when Process Manager is least busy. You may need to adjust this schedule to find the balance between frequency and duration of the process, to ensure server productivity.

Default The default is Daily@Sys:0:0 (daily at midnight).

JS_DTD_DIR

Syntax **JS_DTD_DIR=***JS_HOME*`/3.0/etc`

Description DO NOT CHANGE THIS VALUE.

Specifies the directory containing the DTD files required by Process Manager.

Default The default is *JS_HOME*`/3.0/etc`

JS_ENCRYPTION

Syntax **JS_ENCRYPTION=true|false**

Description Specifies whether to encrypt communication between Process Manager Server and Process Manager Client. If you set this value to true, ensure that the strong encryption package is installed.

Default The default is false—do not encrypt communication.

JS_FAILOVER

Syntax **JS_FAILOVER=false|true**

Description OPTIONAL if failover is not used. REQUIRED if failover is used.

Specifies that the failover feature is to be enabled. The failover feature provides automatic failover in the event the Process Manager Server host becomes unavailable.

Default The default is JS_FAILOVER=false—no failover.

See also JS_FAILOVER_HOST, JS_FOD_PORT

JS_FAILOVER_HOST

Syntax **JS_FAILOVER_HOST=***hostname*

Description OPTIONAL if failover is not used. REQUIRED if failover is used.

Specifies the fully-qualified hostname of the failover host.

If you specified JS_FAILOVER=true, specify the name of the host where Process Manager Server will run if the primary Process Manager Server host is unavailable.

Default The default is the same hostname as that specified for Process Manager Server.

See also JS_FAILOVER, JS_FOD_PORT

JS_FLOW_STATE_MAIL

Syntax **JS_FLOW_STATE_MAIL=true** | **false**

Description Specifies whether or not to allow flow email notifications. When set to true, flow email notification occurs as specified by the user in each flow. When set to false, flow email notification does not occur. This setting has no effect on individual job email notifications or alarm email notifications.

Default The default is true—enable flow email notification.

See also JS_MAIL_SIZE

JS_FILEAGENT_SENSITIVITY

Syntax **JS_FILEAGENT_SENSITIVITY=***seconds*

Description Specifies the time interval in seconds at which Process Manager checks for changes in the file system. This value is used when testing file events.

Default The default is 30 seconds.

JS_FOD_PORT

Syntax **JS_FOD_PORT=***number*

Description OPTIONAL if failover is not used. REQUIRED if failover is used.

Specifies the port number of the failover daemon `fod`.

If you specified JS_FAILOVER=true, specify the port number to be used for communication between the failover daemon and the Process Manager Server daemon.

Default The default is 1999.

See also JS_FAILOVER, JS_FAILOVER_HOST

JS_FY_MONTH

Syntax **JS_FY_MONTH=***n*

Description OPTIONAL.

Specifies the number that corresponds to the starting month of the fiscal year. This value is used in certain system calendars. Specify a value from 1 (January) to 12 (December). For example, to specify March, specify **JS_FY_MONTH=3**.

Default The default is 7 (July).

JS_HISTORY_DIR

Syntax **JS_HISTORY_DIR=***/path/***work/history**

Description Specifies the directory where the history log files are stored.

Default The default is *JS_HOME*`/work/history`.

JS_HISTORY_LIFETIME

Syntax JS_HISTORY_LIFETIME=*hours*

Description Specifies the time period in hours for which history data is collected before a new history log file is created. If the size of the log file exceeds the file size specified in JS_HISTORY_SIZE, a new log file is created, regardless of how many hours of data it contains.

Default The default is 24 hours.

See also JS_HISTORY_SIZE

JS_HISTORY_SIZE

Syntax **JS_HISTORY_SIZE=***bytes*

Description Specifies the maximum number of bytes a history log file can grow to before a new log file is created. If the number of hours of data exceeds the time period specified in JS_HISTORY_LIFETIME, a new log file is created, regardless of its size.

Default The default is 500000 bytes (500 KB).

See also JS_HISTORY_LIFETIME

JS_HOME

Syntax **JS_HOME=***/path*

Description Specifies the full path to the top-level installation directory.

Corresponds to JS_TOP in `install.config`.

Default There is no default for this parameter. A value is set at installation time.

JS_HOST

Syntax **JS_HOST=***hostname*

Description REQUIRED.

Specifies the fully-qualified domain name of the host on which Process Manager Server runs—the name of the host to which the clients connect under normal operations. You cannot specify more than one host.

Default There is no default for this parameter. A value is set at installation time.

See also JS_PORT

JS_IM_ACTIVEPOLICY

Syntax **JS_IM_ACTIVEPOLICY=JF_IM_IPolicy|JF_IM_TPolicy**

Description Specifies the criteria used by Process Manager to determine when to delete a copy of a completed flow from the working set.

Specify JF_IM_IPolicy if you want to use the number of occurrences of the flow as the criteria to delete the flow. The oldest occurrence is deleted first.

Specify JF_IM_TPolicy if you want to use the length of time since the flow completed as the criteria to delete the flow. The oldest occurrence is deleted first.

Default The default policy is JF_IM_IPolicy.

See also JS_IM_POLICY_CHECKING_INTERVAL

JS_IM_POLICY_CHECKING_INTERVAL

Syntax **JS_IM_POLICY_CHECKING_INTERVAL=***minutes*

Description Specifies the time interval in minutes at which Process Manager applies the policy specified in JS_IM_ACTIVEPOLICY.

Default The default interval is 12 minutes.

See also JS_IM_ACTIVEPOLICY, JS_IM_POLICY_LIFETIME, JS_IM_POLICY_NOOFFLOWS

JS_IM_POLICY_LIFETIME

Syntax **JS_IM_POLICY_LIFETIME=***days*

Description Specifies the time interval in days after which completed flows are deleted from the Process Manager working set.

This value takes effect when JS_IM_ACTIVEPOLICY = JF_IM_TPolicy. The oldest occurrence is deleted first.

Default The default is 5 days.

See also JS_IM_ACTIVEPOLICY, JS_IM_POLICY_CHECKING_INTERVAL, JS_IM_POLICY_NOOFFLOWS

JS_IM_POLICY_NOOFFLOWS

Syntax **JS_IM_POLICY_NOOFFLOWS=***number*

Description Specifies the number of copies of a completed flow that are retained within the Process Manager working set. Specify a number greater than 0.

This value takes effect when JS_IM_ACTIVEPOLICY = JF_IM_IPolicy. The oldest occurrence is deleted first.

Default The default is 36 copies.

See also JS_IM_ACTIVEPOLICY, JS_IM_POLICY_LIFETIME, JS_IM_POLICY_CHECKING_INTERVAL

JS_LICENSE_FILE

Syntax JS_LICENSE_FILE=/*path*/*filename*

Description DO NOT CHANGE THIS VALUE.

Specifies the location of the copy that Process Manager makes of the license.dat file.

Default The default is *JS_HOME*/conf.

JS_LIMIT_USER_VIEW

Syntax **JS_LIMIT_USER_VIEW=true | false**

Description Specifies whether a user's view of flows is limited to their own flows, or includes all flows in Process Manager.

Default The default is false—not limited to their own flows.

JS_LOGDIR

Syntax **JS_LOGDIR=***/path*

Description Specifies the name of the directory containing the `jfd.log` file, the error log file for the Process Manager Server daemon.

Default The default is *JS_HOME*`/log`.

JS_LOGIN_REQUIRED

Syntax **JS_LOGIN_REQUIRED=true | false**

Description Specifies if a user login is required to access Process Manager. Set as true if you want to require users to log in before using Process Manager.

Default The default is false—users do not have to log in to use Process Manager.

JS_LOG_MASK

Syntax **JS_LOG_MASK**=*value*

Description Specifies the error logging level used. Change this value only as directed by Platform Technical Support. Valid values from highest to lowest are:

- LOG_EMERG
- LOG_ALERT
- LOG_CRIT
- LOG_ERR
- LOG_WARNING
- LOG_NOTICE
- LOG_INFO
- LOG_DEBUG
- LOG_DEBUG1
- LOG_DEBUG2
- LOG_DEBUG3

The level specified by the log mask determines which messages are recorded and which are discarded. All messages logged at the specified level or higher are recorded, while lower level messages are discarded.

For debugging purposes, the level LOG_DEBUG contains the fewest number of debugging messages and is used for basic debugging. The level LOG_DEBUG3 records all debugging messages, and can cause log files to grow very large; it is not often used. Most debugging is done at the level LOG_DEBUG2.

Default The default is JS_LOG_MASK=LOG_NOTICE.

JS_MAILHOST

Syntax **JS_MAILHOST=**[**SMTP**|**Exchange:**]*hostname*

Description OPTIONAL.

Specifies the name of the mail server host.

On Windows, specify the protocol and name of the mail server host. For an SMTP mail host, specify **SMTP:***hostname*. For an exchange mail host, specify **Exchange:***hostname*.

On UNIX, specify just the name of the mail server host.

Default If Process Manager Server is installed on Windows, the default is Exchange:*localhostname*. If Process Manager Server is installed on UNIX, the default is *localhostname*.

JS_MAIL_SIZE

Syntax **JS_MAILSIZE=***bytes*

Description OPTIONAL.

Specifies the maximum size allowed for a flow email notifications. An email larger than the maximum size specified is truncated.

Default The default is 1000000 (1MB).

JS_MAX_VAR_SUBSTITUTIONS

Syntax **JS_MAX_VAR_SUBSTITUTIONS=***number*

Description OPTIONAL.

Specifies the maximum number of variable substitutions that can be performed in a single job definition field.

Default 10 substitutions

JS_PORT

Syntax **JS_PORT=***number*

Description REQUIRED.

Specifies the port number to be used by the Process Manager Client to connect with Process Manager Server.

Default The default port number is 1966.

See also JS_HOST

JS_PROXY_DURATION

Syntax **JS_PROXY_DURATION=***minutes*

Description Specifies the length of time for which to publish events that occur in Process Manager, keeping the event information available for flows that contain proxies looking for that event. This is required if the event can occur before the flow looking for it requires it.

Default The default is 0—no duration.

JS_REALTIME_UPDATE

Syntax **JS_REALTIME_UPDATE=true | false**

Description Specifies whether or not to enable real-time updates to the data displayed in the Flow Manager. When enabled, the status of work items in the Flow Manager updates automatically as a change occurs. Users can choose real-time updates, automatic refreshes at a specified time interval, or manual refreshes. If you disable this option, and a user has selected real-time updates, the client updates automatically at the specified refresh interval instead.

Default The default is false—not enabled.

JS_REALTIME_OBJECT_URL

Syntax **JS_REALTIME_OBJECT_URL=***url*

Description Required when JS_REALTIME_UPDATE is set to **true**. Specifies the url to the JMS (Java Message Service), used by Process Manager Server when obtaining status updates. This url must match the url specified when configuring the JMS broker—IMQ_JNDI_URL.

Default There is no default for this parameter.

JS_RERUN_RETRY

Syntax **JS_RERUN_RETRY=***tries*

Description Specifies the maximum number of times (including the first time) Process Manager tries to run a work item that has a rerun exception.

Default The default is 30 times.

JS_START_RETRY

Syntax **JS_START_RETRY=***retries*

Description Specifies the maximum number of times Process Manager tries again to start a job or job array before raising a Start Failed exception.

Default The default is 20 times.

JS_STORAGE_DIR

Syntax **JS_STORAGE_DIR=**/*path*/**work/storage**

Description Specifies the directory where the flow definitions and flows are stored for use by Process Manager Server.

Default The default is *JS_HOME*/work/storage.

JS_TIME_ZONE

Syntax **JS_TIME_ZONE=server | client | UTC**

Description Specifies the time zone displayed by the client. The time zone is displayed and used to define and schedule flows.

Server time zone is the time at the server.

Client time zone is the time at the client.

UTC time zone is Coordinated Universal Time (also known at Greenwich Mean Time or GMT).

Note: If you are scheduling a future event that takes place after a seasonal time change (such as Daylight Savings Time) and you have configured either server or client time zones, the time displayed at submission is the time at which the job runs.

When the server and the client are in the same time zone, the server time zone is displayed.

Default The default is **`server`**.

JS_VARIABLE_COMM_DIR

Syntax **`JS_VARIABLE_COMM_DIR=`**/***`path`***

Description Specifies the shared directory to which jobs communicate variable information.

Default The default is *JS_HOME*`/work/var_comm`.

JS_VARIABLE_DIR

Syntax **`JS_VARIABLE_DIR=`**/***`path`***

Description Specifies the directory where variable data is stored.

Default The default is *JS_HOME*`/work/variable`.

LSF_ENVDIR

Syntax **`LSF_ENVDIR=`**/*path*

Description REQUIRED.

Default Specifies the directory where the LSF configuration files are stored. There is no default for this value. A value is set at installation time.

name.alarm

When you define an alarm, you create an individual file for each alarm. The file name is the name of the alarm and the file type is alarm. For more information about alarms, see "Alarm" on page 22. For instructions on creating an alarm, see "Configuring an Alarm" on page 57.

Format

Each alarm file has the following format:

DESCRIPTION=<*description*>
NOTIFICATION=Email[*user1,user2,user3***]**

Example

The following example shows a database failure alarm definition. The alarm is called `DBMSfail.alarm`. It's contents are:

```
DESCRIPTION=Send DBA a message indicating DBMS failure
NOTIFICATION=Email[bsmith,ajones]
```

Index