

*IBM Spectrum LSF Process Manager*  
*10.2*

*Administering*





---

# Tables of Contents

|  |    |
|--|----|
| <b>About IBM Spectrum LSF Process Manager</b>                              | 1  |
| Components   | 1  |
| Security   | 3  |
| About Failover   | 5  |
| About Calendars  | 6  |
| About Exceptions   | 9  |
| User-specified conditions  | 9  |
| Behavior when an exception occurs  | 10 |
| About Exception Handling   | 11 |
| IPv6 support   | 14 |
| <b>Access Control</b>  | 14 |
| Users and administrators in Process Manager                                | 14 |
| Configure a Process Manager administrator or Control administrator         | 20 |
| Configure a Group administrator  | 21 |
| Sign on as a guest   | 22 |
| Maintain User Passwords  | 23 |
| Allow users to trigger other users' flows                                  | 23 |
| Restrict who can see the flow chart view                                   | 24 |
| Integrating Kerberos with Process Manager                                  | 24 |
| LSF Kerberos parameters used by Process Manager                            | 25 |
| Requirements to integrate with Kerberos                                    | 25 |
| Configure to use Kerberos when user login is not required                  | 26 |
| Configure to use Kerberos when user login is required                      | 26 |
| Automatically generate and renew user TGTs with the keytab file            | 28 |
| <b>Maintaining Process Manager</b>   | 29 |
| Failover   | 29 |
| Configure a failover host (managed by EGO)                                 | 29 |
| Install and configure a failover host on UNIX (managed by failover daemon) | 30 |
| Configure the primary host   | 31 |
| Prepare the installation files on the failover host                        | 31 |
| Prepare the configuration on the failover host                             | 31 |
| Install the software on the failover host                                  | 32 |
| Client and server hosts  | 33 |
| Add a Windows client   | 33 |
| Run the Process Manager server on system startup                           | 33 |
| Control the Process Manager Server   | 34 |
| Start and stop the Server on Windows                                       | 34 |
| Forcing a system snapshot  | 35 |
| Variables  | 36 |
| About variables in Process Manager   | 36 |
| Types of variables   | 36 |
| Scope of variables and variable override order                             | 37 |
| How variables are set  | 38 |
| How environment variables are set  | 41 |
| Configuration  | 42 |
| Change the Configuration   | 42 |
| Output and error file generation for work items in a flow                  | 43 |

|   |    |
|---|----|
| Define a default working directory for flows                                      | 45 |
| Configure an alarm  | 45 |
| Specify the mail host   | 46 |
| Change the job start retry value  | 47 |
| Converting the job command line to native encoding when jobs are submitted to LSF | 47 |
| Calendars and time zones  | 48 |
| Create system calendars   | 49 |
| Calendar names  | 49 |
| Update the Holidays@Sys calendar  | 49 |
| Delete a calendar   | 50 |
| Updating time zone data   | 50 |
| Local Jobs  | 53 |
| About local jobs on Linux and UNIX  | 53 |
| About local jobs on Windows   | 54 |
| History   | 55 |
| Change the history setting  | 55 |
| View History  | 56 |
| View the history of a flow definition   | 56 |
| View the history of a flow  | 57 |
| View the history of a job or job array  | 57 |
| <b>Using Process Manager with Other Batch Systems</b>                             | 58 |
| How Process Manager works with Other Batch Systems                                | 58 |
| About Other Batch Jobs  | 59 |
| Configuring Process Manager to work with Other Batch Systems                      | 61 |
| Step Summary and Configuration Checklist  | 61 |
| 1. Configure password-less SSH connections  | 63 |
| 2. Enable another Batch System  | 64 |
| 3. Configure connection to the Other Batch System                                 | 64 |
| 4. Customize job submission, control, and query scripts                           | 64 |
| 5. Restart the Process Manager Server   | 68 |
| 6. Test the Other Batch System  | 69 |
| Configure Data Transfer to and from the Other Batch System                        | 69 |
| Troubleshooting Other Batch System Jobs   | 70 |
| <b>Mainframe support</b>  | 71 |
| Configure for Mainframe   | 71 |
| <b>Daemons</b>  | 72 |
| jfd   | 72 |
| fod   | 73 |
| <b>Commands</b>   | 73 |
| caleditor   | 76 |
| floweditor  | 77 |
| flowmanager   | 77 |
| jadmin  | 78 |
| jalarms   | 79 |
| jcadd   | 81 |
| jcals   | 86 |
| jcdel   | 87 |
| jcmmod  | 88 |
| jcommit   | 91 |
| jcomplete   | 96 |

|                                  |     |
|----------------------------------|-----|
| jdefs                            | 98  |
| jexport                          | 99  |
| jflows                           | 100 |
| jhist                            | 102 |
| jhold                            | 107 |
| jid                              | 108 |
| jjob                             | 108 |
| jkill                            | 111 |
| jlicenseupdate                   | 112 |
| jmanuals                         | 113 |
| jpublish                         | 114 |
| jreconfigadmin                   | 115 |
| jreconfigalarm                   | 115 |
| jrelease                         | 116 |
| jremove                          | 117 |
| jrerun                           | 118 |
| jresume                          | 119 |
| jrun                             | 120 |
| jsetvars                         | 121 |
| jsetversion                      | 123 |
| jsinstall                        | 124 |
| jstop                            | 125 |
| jsub                             | 126 |
| jsubmit                          | 126 |
| jtrigger                         | 126 |
| junpublish                       | 128 |
| licenseinfo                      | 129 |
| ppmsetvar                        | 130 |
| <b>Files</b>                     | 132 |
| File Structure                   | 132 |
| Files created on the server host | 132 |
| Process Manager history files    | 133 |
| Process Manager log files        | 134 |
| history.log                      | 134 |
| install.config                   | 135 |
| js.conf                          | 141 |
| name.alarm                       | 176 |

---

# About IBM Spectrum LSF Process Manager

This chapter introduces IBM Spectrum LSF Process Manager (Process Manager) concepts, contains an overview of the architecture, and describes the client components and their use.

## Overview

---

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

Process Manager has Java clients (Flow Editor, Flow Manager, and Calendar Editor), and a web client integrated with IBM Spectrum LSF Application Center. Using the Java or web clients, users can create and submit complex flow definitions to the Process Manager Server, which manages the dependencies within a flow and controls the submission of jobs to the IBM Spectrum LSF(LSF) management host. LSF® provides resource management and load balancing, and runs the jobs and returns job status to the Process Manager Server. From the 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.

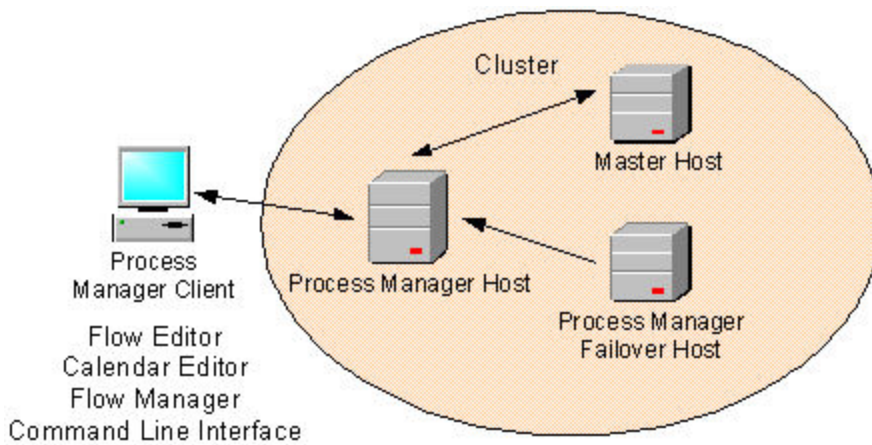
- [Components](#)
- [Security](#)
- [About Failover](#)
- [About Calendars](#)
- [About Exceptions](#)
- [User-specified conditions](#)
- [Behavior when an exception occurs](#)
- [About Exception Handling](#)

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.

- [IPv6 support](#)

---

## Components



The system is made up of the following components:

- The Process Manager (Server) host
- The Process Manager (Server) failover host
- The management host
- Process Manager Client, which consists of the following:
  - Process Manager Java Clients
    - The Flow Editor
    - The Calendar Editor
    - The Flow Manager
    - The Command Line Interface (CLI)

## 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.

### Running multiple Process Manager servers and daemons

You can have multiple Process Manager servers in a single IBM® LSF cluster, and you can install and run multiple instances of `jfd` on one or more Process Manager servers. This is useful, for example, if you have different Process Manager environments running in one cluster.

To avoid conflicts and to ensure that each job is unique among multiple Process Manager servers, you must ensure that each combination of user name and flow name is unique within the cluster.

## 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.

## Management host

The management host receives jobs from the Process Manager Server, manages any resource dependencies the job may have, and dispatches the job to an appropriate LSF host.

### **LSF management host**

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 Clients**

---

### **Process Manager Designer**

The Process Manager Designer allows users to edit Process Manager flows and calendars by using the Flow Editor and the Calendar Editor.

### **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.

Tip:

Flow Editor may not be installed if you purchased the Platform Suite for SAS. For more information, contact your sales representative.

### **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.

### **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.

---

## **Security**

Process Manager, in its default configuration, provides security through the following methods:



- User authentication
- Role-based access control

## User authentication

---

Two models for user authentication are supported. In `js.conf`, specify `JS_LOGIN_REQUIRED=true|false`, which indicates whether a user is asked to log in when they start Process Manager Clients or not.

If `JS_LOGIN_REQUIRED=false`, no login is required.

If `JS_LOGIN_REQUIRED=true`, when the user starts Calendar Editor or Flow Manager they are prompted for a user name and password which is verified by the Process Manager Server. If the user name is a Windows user name, it must also include the domain name. The domain name and user name are passed to the server for verification. The Process Manager Server tries to verify the user name from the domain.

## LDAP

---

Process Manager supports LDAP authentication through PAM (Pluggable Authentication Modules, a 3rd-party tool) if `JS_LOGIN_REQUIRED=true`.

To enable LDAP authentication, you need to configure your PAM policy to add a service name `eauth_userpass` for the module type: `auth`.

For example, in a Solaris system, you may add the following entry in the `/etc/pam.conf` file:

```
eauth_userpass    auth    required    pam_ldap.so.1
```

## Role-based access control

---

In addition to authentication, Process Manager uses role-based access control to secure certain types of objects.

By default, any user in Process Manager can create and submit their own flow definitions, and monitor and control their own flows, as long as their user ID is recognized by LSF®. In addition, by default all users can view calendars and flows submitted by another user. However, 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 roles:

- Normal user
- Primary Process Manager administrator
- Process Manager administrator
- Process Manager Control administrator
- Process Manager Group administrator

## Encrypted communications

---

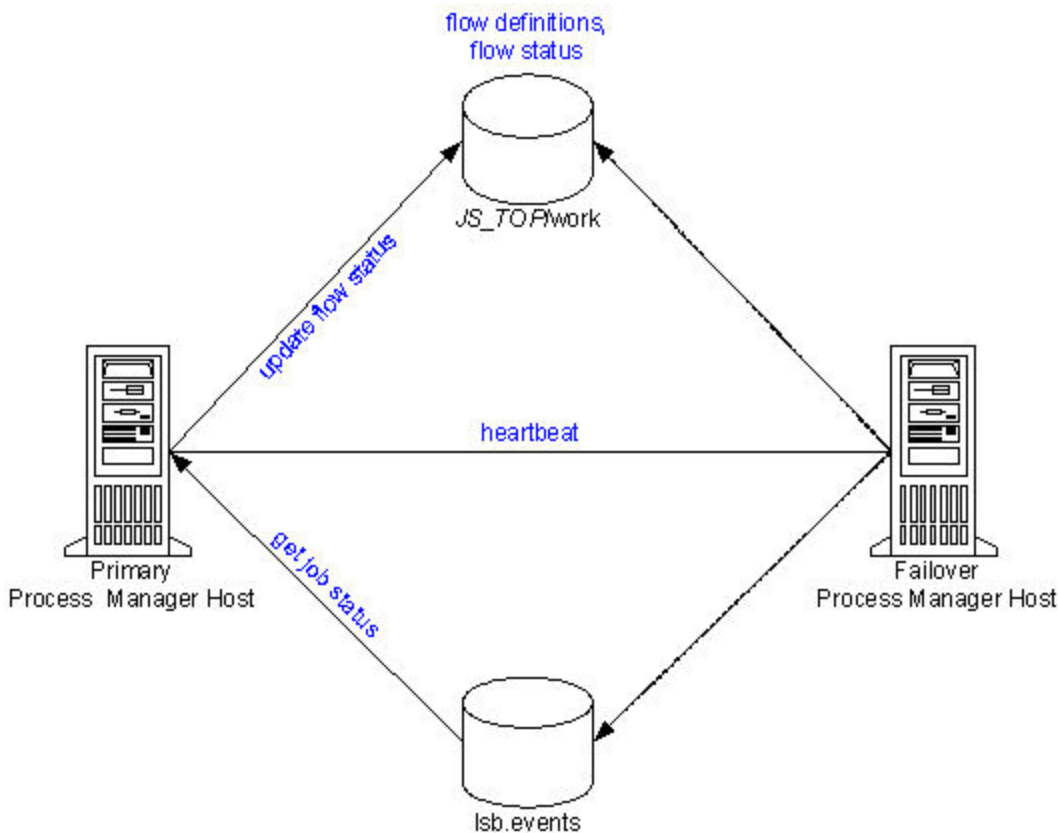
You can enable encrypted communications between Process Manager Server and its clients to further secure the Process Manager network. Set the parameter `JS_ENCRYPTION=true` in the configuration file `js.conf` on the Server, and also set `JS_ENCRYPTION=true` in the `js.conf` file on all clients.

# About Failover

Process Manager supports an optional failover feature, 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 Enterprise Grid Orchestrator(EGO) or failover daemon (fod, in case of UNIX), 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 or EGO on the failover host monitors 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 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 (in time or number of occurrences). 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.

---

## 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.

Process Manager comes with a set of pre-defined system calendars that you can use as is 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, if you change or delete a calendar when it is in use (that is, 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), your changes will only take effect on any new instances; current instances will continue to use the previous calendar definition.

---

## Time zones

It is possible for users to run their Process Manager Clients from a different geographic time zone than the Process Manager Server. Therefore it is important to note that, by default, all time events specified in a flow definition are based on the time zone set in JS\_TIME\_ZONE. For example, Joe is in Los Angeles and is connected to a Process Manager server in New York. He has set JS\_TIME\_ZONE=server. When Joe defines a flow to trigger at 5 p.m, it triggers at 5 p.m. New York time, not Los Angeles time.

If you change the time zone, you must restart Process Manager.

You can also change the time zone of a specific time event when you create that time event.

All start times displayed for a work item in Flow Manager are in GMT (Universal Time).

Tip:

Note that the time used with the calendars is based on the time zone set in JS\_TIME\_ZONE. The time zone can be set as server, client (default), or Universal Time (UTC also known as GMT).

## 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 |

| Types of Calendars | Calendar Names                 |
|--------------------|--------------------------------|
| 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.

## 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.

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 server 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.

---

## 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                       |

## 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 or Subflow | Misschedule                   | The flow or subflow is not run.  |
|                 | Overrun                       | The flow or subflow continues to run after the exception occurs. The run time is calculated from when the flow or 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 flow or 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.   |
|                 | 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.  |

| When a... | Experiences this exception... | This happens...   |
|-----------|-------------------------------|---|
|           | 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.                           |
|           | 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.  |

## 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.

### 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. When selecting the Rerun exception handler, you can specify the maximum number of times the exception handler reruns the work item.

#### 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.

If you are running z/OS® mainframe jobs on Windows, you need to configure a special queue and submit jobs to that queue to be able kill them.

#### Alarm

An *alarm* provides both a visual cue that an exception has occurred, and either sends an email notification or executes a script. 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.

### Flows

| When a Flow Experiences this Exception... | and the Handler Used is... | This Happens...  |
|---|----------------------------|--|
| 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.  |

### Subflows

| When a Subflow Experiences this Exception... | and the Handler Used is... | This Happens...  |
|--|----------------------------|--|
| 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. |

| When a Subflow Experiences this Exception... | and the Handler Used is... | This Happens...   |
|--|----------------------------|---|
| 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

| When a Job or Job Array Experiences this Exception... | and the Handler Used is... | This Happens...  |
|---|----------------------------|--|
| 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.

---

# IPv6 support

The Process Manager Server daemon (JFD) handles communication between the IPv4 and IPv6 hosts in the following manner:

- IPv4 only

JFD listens on an IPv4 socket and can only accept connections from IPv4 clients.

- IPv6 only

JFD listens on an IPv6 socket and can only accept connections from IPv6 clients.

- IPv4/IPv6 dual stack

JFD can accept connections from both IPv4 and IPv6 clients. Most operating systems that support IPv6 can accept both IPv6 and IPv4 connections by emulating an IPv6 address: the operating system converts the IPv4 address to an IPv4-mapped IPv6 address.

---

# Access Control

- [Users and administrators in Process Manager](#)
- [Configure a Process Manager administrator or Control administrator](#)
- [Configure a Group administrator](#)
- [Sign on as a guest](#)
- [Maintain User Passwords](#)
- [Allow users to trigger other users' flows](#)
- [Restrict who can see the flow chart view](#)
- [Integrating Kerberos with Process Manager](#)

---

# Users and administrators in Process Manager

In addition to authentication, Process Manager uses role-based access control to secure certain types of objects.

By default, any user in Process Manager can create and submit their own flow definitions, and monitor and control their own flows, as long as their user ID is recognized by LSF®. In addition, by default all users can view calendars and flows submitted by another user. However, special permissions are required to install and configure Process Manager, or to modify Process Manager items on behalf of another user.

## Configuration of user roles

| Role                                  | Where defined   | Summary of permissions  |
|---------------------------------------|---|---|
| Normal user                           | Any operating system user on LSF hosts. All users are automatically assigned this role.   | Normal users can view flow definitions, flows, calendars, and jobs that are owned by all users but can control only work items that they own.   |
| Primary Process Manager administrator | First user that is specified in <b>JS_ADMINS</b> in the file js.conf.   | 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 that is required to stop the Process Manager Server.<br><br>The Primary Process Manager administrator has full control over all work items of all users and can view, control, and modify flow definitions, flows, calendars, and jobs on behalf of other users. |
| Process Manager administrator         | Users that are specified in <b>JS_ADMINS</b> in the file js.conf after the first one that is listed.                                | Process Manager administrators have full control over all Process Manager items of all users.<br><br>Process Manager administrators can view, control, and modify flow definitions, flows, calendars, and jobs on behalf of other users.  |
| Process Manager Control administrator | Users that are specified in <b>JS_CONTROL_ADMINS</b> in the file js.conf.   | Process Manager Control administrators can view flow definitions, flows, calendars, and jobs that are owned by all users and can control flows (not flow definitions) and jobs on behalf of other users.  |
| Process Manager Group administrator   | Users that are specified as GROUP_ADMIN in LSF user groups in the lsb.users file when <b>JS_ENABLE_GROUP_ADMIN=true</b> in js.conf. | Group administrators can view flow definitions, flows, and calendars owned by all users.<br><br>Group administrators can control flow definitions, flows and jobs on behalf of users who are members of the same LSF user group.  |

## Normal users permissions details

| Item | View | Control | Specify Owner(when <b>JS_ENABLE_GROUP_ADMIN=true</b> in js.conf) |
|------|------|---------|--|
|      |      |         |  |

| Item                    | View  | Control   | Specify Owner(when JS_ENABLE_GROUP_ADMIN=true in js.conf)   |
|-------------------------|---|---|---|
| <b>Flow definitions</b> | <p>All flow definitions of all users.</p> <p>When JS_LIMIT_USER_VIEW=true in js.conf, can view only flow definitions that they own.</p> <p>When JS_LIMIT_FLOW_CHART_VIEW=true in js.conf, can view only the flow chart of flow definitions that they own.</p>                 | <p>Can perform all operations only on flow definitions that they own(saved by that user account and as a result, owned by that user account).</p> <p>When JS_CHANGE_FLOW_OWNER=true in js.conf and the flow definition is published, can additionally trigger flows from other users' flow definitions and own those flows.</p> | Cannot set a flow definition owner. The default owner of the flow is the submission user.                   |
| <b>Flows</b>            | <p>All flows of all users.</p> <p>When JS_LIMIT_USER_VIEW=true in js.conf, can view only flows that they own.</p> <p>When JS_LIMIT_FLOW_CHART_VIEW=true in js.conf, can view only the flow chart of flows if they are the owner of both the flow definition and the flow.</p> | <p>Only flows owned by themselves(from flow definitions that they submitted), all operations.</p> <p>When JS_CHANGE_FLOW_OWNER=true in js.conf and the flow definition is published, can own and perform all operations on flows triggered from other users' flow definitions.</p>  | Cannot set a flow owner. Flows are owned by the user that is specified as the owner in the flow definition. |
| <b>Jobs</b>             | All jobs of all users.  | Only jobs that they own(running as their user account), all operations.   | Cannot set a job owner. The job is owned by the user specified as the Run As user in the job definition.    |
| <b>Calendars</b>        | All calendars of all users.   | Only calendars that they own(that they added), all operations.  | Cannot set a calendar owner. The calendar is owned by the user specified as the owner in Calendar Editor.   |

## Process Manager Administrators permissions details

| Item                    | View                               | Control  | Specify Owner(when JS_ENABLE_GROUP_ADMIN=true in js.conf)   |
|-------------------------|------------------------------------|--|---|
| <b>Flow definitions</b> | All flow definitions of all users. | All flow definitions of all users, all operations. | The default owner of the flow definition is the submission user, but Process Manager administrators can specify any valid user name as the owner. |

| Item             | View                        | Control                                     | Specify Owner(when<br>JS_ENABLE_GROUP_ADMIN=true in js.conf)  |
|------------------|-----------------------------|---|---|
| <b>Flows</b>     | All flows of all users.     | All flows of all users, all operations.     | Flows are owned by the user that is specified as the owner in the flow definition.  |
| <b>Jobs</b>      | All jobs of all users.      | All jobs of all users, all operations.      | Jobs are owned by the user that is specified as the Run As user owner in the job definition.  |
| <b>Calendars</b> | All calendars of all users. | All calendars of all users, all operations. | The logged on user is set by default as the calendar owner, but Process Manager administrators can specify any valid user name as the calendar owner. |

## Process Manager Control Administrators permissions details

| Item                    | View  | Control   | Specify Owner(when<br>JS_ENABLE_GROUP_ADMIN=true in js.conf)   |
|-------------------------|---|---|--|
| <b>Flow definitions</b> | <p>All flow definitions of all users.</p> <p>When<br/>JS_LIMIT_USER_VIEW=true in js.conf, can view all flow definitions of all users.</p> <p>When<br/>JS_LIMIT_FLOW_CHART_VIEW=true in js.conf, can view only the flow chart of flow definitions that they own.</p> | <p>All flow definitions of all users, but cannot submit or remove flow definitions that belong to other users.</p> <p>When<br/>JS_CHANGE_FLOW_OWNER=true in js.conf, can trigger flows from other users' flow definitions and own those flows.</p> <p>When<br/>JS_CHANGE_FLOW_OWNER=false in js.conf, can trigger flows from other users' flow definitions. The flow owner is the user who submitted the flow definition.</p> | Cannot specify an owner for the flow definition. The flow definition is owned by the user who saves the flow definition. |

| Item             | View  | Control                                     | Specify Owner(when JS_ENABLE_GROUP_ADMIN=true in js.conf)  |
|------------------|---|---|--|
| <b>Flows</b>     | <p>All flows of all users.</p> <p>When JS_LIMIT_USER_VIEW=true in js.conf, can view all flows of all users.</p> <p>When JS_LIMIT_FLOW_CHART_VIEW=true in js.conf, can view only the flow chart of flows if they are the owner of both the flow definition and the flow.</p> | All flows of all users, all operations.     | <p>Cannot specify an owner for the flow. The flow is owned by the user that is specified as owner in the flow definition.</p> <p>When JS_CHANGE_FLOW_OWNER=true in js.conf, flows are owned by the triggering user.</p> <p>When JS_CHANGE_FLOW_OWNER=false in js.conf, the flow owner is the user who submitted the flow definition.</p> |
| <b>Jobs</b>      | All jobs of all users.  | All jobs of all users, all operations.      | Cannot specify an owner for the job. The job is owned by the user that is specified as the Run As user in the job definition.  |
| <b>Calendars</b> | All calendars of all users.   | All calendars of all users, all operations. | Cannot specify a calendar owner. The calendar is owned by the user that is specified as the owner in Calendar Editor.  |

## Process Manager Group Administrators permissions details

| Item | View | Control | Specify Owner |
|------|------|---------|---------------|
|------|------|---------|---------------|

| Item                    | View  | Control  | Specify Owner   |
|-------------------------|---|--|---|
| <b>Flow definitions</b> | <p>All flow definitions of all users.</p> <p>When <b>JS_LIMIT_USER_VIEW=true</b> in js.conf, can view only flow definitions owned by members of their user groups.</p> <p>When <b>JS_LIMIT_FLOW_CHART_VIEW=true</b> in js.conf, can see the flow chart of a flow definition if a member of their group is the owner of the flow definition.</p>   | <p>All flow definitions that they own and that are owned by users in their user group, all operations.</p> <p>When <b>JS_CHANGE_FLOW_OWNER=false</b> in js.conf and the flow definition is published or unpublished, Group administrators can trigger flows that are owned by members of their user groups and the flow owner is that defined in the flow definition. Can perform all operations on those flows.</p> <p>When <b>JS_CHANGE_FLOW_OWNER=true</b> in js.conf and the flow definition is published or unpublished, Group administrators can trigger flows from flow definitions that are owned by members of their user groups and flows are owned by the triggering user. Can perform all operations on those flows.</p> | <p>The default owner of the flow definition is the submission user.</p> <p>The Process Manager Group administrator can specify a different owner. Valid users for owners are members of the same groups as the Group administrator.</p> |
| <b>Flows</b>            | <p>All flows of all users.</p> <p>When <b>JS_LIMIT_USER_VIEW=true</b> in js.conf, can view only flows owned by members of their user groups.</p> <p>When <b>JS_LIMIT_FLOW_CHART_VIEW=true</b> in js.conf, can see the flow chart of a flow only if both the flow definition and flow are owned by members of their group. If the Group member triggers a flow from a published flow definition that is owned by someone not belonging to the user group, the Group administrator and the group member will not be able to see the flow chart of the flow.</p> | <p>All flows that are owned by themselves and users in their user groups, all operations.</p>  | <p>The owner of the flow is the user that is specified as owner in the flow definition.</p>   |



| Item             | View                        | Control  | Specify Owner   |
|------------------|-----------------------------|--|---|
| <b>Jobs</b>      | All jobs of all users.      | All jobs running as themselves and as user accounts in their user groups, all operations.            | The owner of the job is the user that is specified as the Run As user in the job definition.  |
| <b>Calendars</b> | All calendars of all users. | All calendars that are owned by themselves and by user accounts in their user group, all operations. | By default, the user who adds the calendar is the calendar owner.<br><br>Process Manager Group administrators can specify a different calendar owner. Valid users for owners are members of the same groups as the Group administrator. |

## Configure a Process Manager administrator or Control administrator

### About this task

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.

### Procedure

1. Stop the Process Manager Server and edit js.conf.
2. To add a Process Manager administrator, for the JS\_ADMIN parameter, specify one or more user IDs or user group names after the primary administrator name.  
To specify a list, separate the names with a comma. If the Windows user ID or active directory user group contains spaces, enclose the user ID or group name in quotation marks.

For example, to specify Windows users and user groups:

```
JS_ADMIN=DOMAIN\lsfadmin, "DOMAIN\Engineering Group", DOMAIN\userA
```

3. For JS\_CONTROL\_ADMIN, specify one or more user IDs or UNIX user group names.

To specify a list, separate the names with a comma. If the Windows user ID or active directory user group contains spaces, enclose the user ID or group name in quotation marks.

For example, to specify Windows users and user groups:

```
JS_CONTROL_ADMINS=DOMAIN\admin, "DOMAIN\QA Group", DOMAIN\userA
```

4. Complete the instructions for changing your configuration, saving `js.conf` and run the command **jreconfigadmin**.

---

## Configure a Group administrator

### Procedure

---

1. Open the configuration file `js.conf` and enable Group administrators in Process Manager.
  - a. Set the parameter `JS_ENABLE_GROUP_ADMIN=true` to enable Group administrators and allow them to operate on flow definitions, flows, jobs, and calendars owned by accounts that are listed in `GROUP_MEMBER` in `lsb.users`. The Owner field is also enabled in the Flow Attributes in Flow Editor, and in the Calendar description in Calendar Editor.
  - b. Restart the Process Manager Server `jfd`.

```
jadmin stop  
jadmin start
```

2. Configure LSF® user groups to identify the accounts that are the administrators of the user groups and the accounts that are the members.
  - a. As the LSF administrator, log in to any host in the cluster.
  - b. Open `lsb.users` and configure user groups.
    - In `GROUP_MEMBERS` for the LSF user group, specify the accounts that are members of the user group.
    - In `GROUP_ADMINS`, specify the accounts that are the group administrators.

For example, `userA` and `usergroupB` are group administrators of flow definitions, flows, jobs, and calendars owned by account `ma1`, and `userD` and `userE` are group administrators of flow definitions, flows, jobs, and calendars owned by accounts `ma2` and `ma3`.

```
Begin UserGroup
```

| GROUP_NAME  | GROUP_MEMBER | USER_SHARES | GROUP_ADMIN   |
|-------------|--------------|-------------|---------------|
| ugroup1     | (ma1)        | ()          | (userA        |
| usergroupB) |              |             |               |
| ugroup2     | (ma2 ma3)    | ()          | (userD userE) |

```
...
```

```
End UserGroup
```

- c. Save your changes.
  - d. Run **badminkconfig** to check the new user group definition. If any errors are reported, fix the problem and check the configuration again.
  - e. Run **badmin reconfig** to reconfigure the cluster.
  - f. Run **jreconfigadmin** to reload LSF user group information in Process Manager and apply changes.
3. Test your configuration.
  - a. In Flow Editor, open any flow as a Group administrator.  
For example, Group administrator `userA`.

- b. Submit the flow.

The owner of the flow is the account that you specified in the Flow Attributes.

For example, if you specified ma1 in the Flow Attributes, and your flow name is Sample, your flow name is ma1:Sample after submission.

- c. In Flow Manager, trigger the flow as the Group administrator account.

For example, as user userA, trigger the flow ma1:Sample. The flow is owned by account ma1.

- d. In Flow Manager, operate on the flow as the Group administrator account.

For example, as user userA, suspend and resume the flow ma1:Sample.

- e. View information about who submitted the flow, who triggered the flow, and who is the owner of the flow:

- In Flow Manager, Runtime Attributes, you can see the triggering user.
- From the command line, use the command **jhist** with the `-o` all or `-o operator_name` option to view which user owns the flow, submitted the flow, and triggered the flow.

---

## Sign on as a guest

### About this task

---

A guest account allows you to have view access to flows and jobs.

As a guest, you have access to the view-only functionality of Flow Manager and Calendar Editor. You can view but not operate on flow definitions, flows, and jobs. You can view but not create, modify, or delete calendars.

Guest accounts also have access to the following commands:

- jid
- jalarms
- jflows
- jdefs
- jmanuals
- jcal

Guest accounts do not have access to the Flow Editor or to any other commands.

JS\_LOGIN\_REQUIRED must be set to true. You can only sign on to the Calendar Editor or Flow Manager. You cannot log on to the Flow Editor.

## Procedure

---

1. Start Calendar Editor or Flow Manager.
2. Login user name: guest  
The user name is case-sensitive.
3. Leave the password blank.
4. Click OK.

## Limit the guest account

---

Administrators can limit the guest account so that it cannot view any flows.

### Procedure

1. Open `js.conf` for editing.
2. Set the parameter `JS_LIMIT_USER_VIEW=true`.

---

## Maintain 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.

## Update 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.

### Run **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.

### Run 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.

---

## Allow users to trigger other users' flows

By default, only Process Manager administrators and Process Manager control administrators can trigger flows created by other users.

Non-administrator users can only trigger flows from flow definitions that they have submitted to Process Manager.

There are situations, however, in which you may want some users to create and submit flow definitions and other users to be able to trigger flows from these flow definitions and control them. In these cases, you want to create flow definitions that can be shared across users and you want the users who triggered the flow to own the flow.

When a user owns the flow, the user also has permission to control the flow and jobs in that flow. See the description of `JS_CHANGE_FLOW_OWNER` in this guide, under *Files, js.conf* for more details on permissions.

To allow users to trigger flows from flow definitions created by other users:

1. Set the parameter `JS_CHANGE_FLOW_OWNER=true` in `js.conf`. When this parameter is set to true:

Note: This feature only applies to flow definitions that have the status Published.

- Users other than the user who submitted the flow definition can trigger the flow.
  - When the flow is triggered, the flow owner is the user who triggered the flow. Jobs in the flow run as the user who triggered the flow.
  - In Flow Manager, the value defined in the job definition `Run As` field is replaced with the user name of the user who triggered the flow.
2. Restart the Process Manager Server.
  3. Publish the flow definition to Process Manager.

---

## Restrict who can see the flow chart view

By default, users who can view a flow or flow definition in Flow Manager can see everything about the flow: the flow chart, general information, subflows and jobs, flow data, and flow history.

In some cases, however, you may not want users to see the chart view of a flow.

It is possible to restrict viewing the chart view of a flow and flow definition, to only the Process Manager administrator and users who are both the flow definition owner and flow owner.

This restriction takes effect in Flow Manager. In Flow Manager, if the user does not have permission to see the flow chart, the related menu items will be grayed out.

To restrict who can see the flow chart view:

1. Set the parameter `JS_LIMIT_FLOW_CHART_VIEW=true` in `js.conf`.
2. Restart the Process Manager Server.

---

## Integrating Kerberos with Process Manager

When the Kerberos integration is enabled, Process Manager acquires and renews user TGTs for operating system user accounts used to run jobs in flows. You can enable the Kerberos integration by setting the LSF parameter `LSB_KRB_TGT_FWD=Y` in the LSF configuration file `lsf.conf`. If you change the value of `LSB_KRB_TGT_FWD` in `lsf.conf`, reconfigure LSF, then restart Process Manager Server to make changes take effect. The Kerberos integration only applies to LSF jobs, job arrays, job submission scripts, and job submission script arrays in flows.

- [LSF Kerberos parameters used by Process Manager](#)
- [Requirements to integrate with Kerberos](#)
- [Configure to use Kerberos when user login is not required](#)  
Complete these configuration steps to enable Process Manager to work with Kerberos when users are not required to log in to Process Manager with a password(`JS_LOGIN_REQUIRED=false` in `js.conf`).
- [Configure to use Kerberos when user login is required](#)  
Complete these configuration steps to enable Process Manager to work with Kerberos when users are required to log in to Process Manager with a password(`JS_LOGIN_REQUIRED=true` in `js.conf`).
- [Automatically generate and renew user TGTs using the keytab file](#)  
In cases when you run flows repeatedly over a long period of time (monthly, annually) or when you have flows that run for a very long time, it is possible that jobs may fail when user Ticket Granting Tickets (TGTs) have reached their maximum renewal lifetime period and can no longer be renewed. By configuring a Kerberos keytab file, you can authenticate users with the Kerberos server without prompting for a password. Additionally, you can configure LSF Process Manager to acquire a new TGT for users before their TGT expires so that jobs in flows always have the required credentials to run.

---

## LSF Kerberos parameters used by Process Manager

Process Manager uses the following parameters set in the LSF configuration file `lsf.conf` for integration with Kerberos:

- `LSB_KRB_TGT_FWD` to identify whether the Kerberos integration is enabled. When set to Y, Kerberos integration is considered to be enabled by Process Manager.
- `LSB_KRB_LIB_PATH` to determine the location of the krb5 libraries. If not set, it defaults to `/usr/local/lib`.
- `LSB_KRB_RENEW_MARGIN` to determine whether a user TGT can be renewed before it expires. If the TGT expiration date is within this margin, the Process Manager Server automatically renews the user TGT. This parameter is optional. If not set, the default value of 1 hour is used.
- `LSB_KRB_CHECK_INTERVAL` to determine how often the Process Manager Server checks whether the Kerberos TGT is within the `LSB_KRB_RENEW_MARGIN` for TGT renewal. This parameter is optional. If not set, the default value of 15 minutes is used.

---

## Requirements to integrate with Kerberos

- LSF 9.1.1.1 or higher is installed and running
- All user TGTs are renewable and forwardable
- Process Manager Server is installed on Linux/UNIX

---

# Configure to use Kerberos when user login is not required

Complete these configuration steps to enable Process Manager to work with Kerberos when users are not required to log in to Process Manager with a password(JS\_LOGIN\_REQUIRED=false in js.conf).

---

## Before you begin

Ensure you have met the requirements for using Process Manager with Kerberos. See [Requirements to integrate with Kerberos](#) for details.

---

## About this task

When a user can log in to Process Manager without a password (JS\_LOGIN\_REQUIRED=false in js.conf on both the Process Manager client and Process Manager Server) Process Manager attempts to locate the user TGT on the client host in /tmp/krb5cc\_user\_UID, and if not found, in the environment variable KRB5CCNAME.

Once the user TGT is located, the Process Manager client forwards the user TGT to the Process Manager Server. The user TGT is forwarded with every client request along with the creation time of the user TGT file. The TGT is then copied to the Process Manager Server's work directory, where it is periodically renewed, and forwarded to LSF when jobs in the flow are submitted.

If no user TGT can be located, the client request still proceeds but messages are logged in the history file and in jfd.log.host\_name.

---

## Procedure

1. Enable Kerberos authentication in LSF.  
Refer to *Administering IBM Spectrum LSF* for more details.
2. Set the parameter LSB\_KRB\_TGT\_FWD=Y in the LSF configuration file lsf.conf and reconfigure LSF to make the changes take effect.  
This setting identifies to Process Manager that Kerberos is enabled.
3. Restart the Process Manager Server to make changes take effect.

```
jadmin stop  
jadmin start
```

---

# Configure to use Kerberos when user login is required

Complete these configuration steps to enable Process Manager to work with Kerberos when users are required to log in to Process Manager with a password(JS\_LOGIN\_REQUIRED=true in js.conf).

---

## Before you begin

Ensure you have met the requirements for using Process Manager with Kerberos. See [Requirements to integrate with Kerberos](#) for details.

## About this task

---

When a user is required to log in to Process Manager with a password (JS\_LOGIN\_REQUIRED=true in js.conf), Process Manager uses the Pluggable Authentication Module (PAM) on the Process Manager server to generate a valid user TGT.

Whenever a user successfully logs in to Process Manager, Process Manager generates a valid user TGT in /tmp/krb5cc\_user\_UID. The TGT is then copied to the Process Manager server's work directory, where it is periodically renewed, and forwarded to LSF when jobs in the flow are submitted.

Should Process Manager fail to acquire or renew the user TGT, Process Manager logs history messages and messages in jfd.log.host\_name.

## Procedure

---

1. Enable Kerberos authentication in LSF.  
Refer to *Administering IBM Spectrum LSF* for more details.
2. Set the parameter LSB\_KRB\_TGT\_FWD=Y in the LSF configuration file lsf.conf and reconfigure LSF to make the changes take effect.  
This setting identifies to Process Manager that Kerberos is enabled.
3. On the Process Manager Server host, configure the PAM Kerberos module (pam\_krb5.so) so that every time a user logs in to the host, a valid user TGT is generated.  
Refer to your Kerberos documentation for more details.
4. Configure a service eauth\_userpass file, then add the module pam\_krb5.so. This enables Process Manager to authenticate with PAM.

- On Red Hat Linux:

- Check that pam\_krb5.so is listed in the file /etc/pam.d/password-auth. For example:

```
#%PAM-1.0# This file is auto-generated.
# # User changes will be destroyed the next time authconfig is run.
auth required pam_env.so
auth sufficient pam_unix.so nullok try_first_pass
auth requisite pam_succeed_if.so uid >= 500 quiet
auth sufficient pam_krb5.so use_first_pass

auth required pam_deny.so
...
```

- Create the /etc/pam.d/eauth\_userpass file and include the following line in the file:

```
auth include password-auth
```

- On SUSE Linux, AIX, Solaris, and HP-UX operating systems, add an entry to the /etc/pam.conf file. For example:

```
eauth_userpass auth required pam_krb5.so
```

5. Restart the Process Manager Server to make changes take effect.

```
jadmin stop
```



```
jadmin start
```

Note:

If for some reason the system does not allow Process Manager to generate a user TGT, set `JS_KRB_USE_KINIT=true` in `js.conf`.

---

## Automatically generate and renew user TGTs using the keytab file

In cases when you run flows repeatedly over a long period of time (monthly, annually) or when you have flows that run for a very long time, it is possible that jobs may fail when user Ticket Granting Tickets (TGTs) have reached their maximum renewal lifetime period and can no longer be renewed. By configuring a Kerberos keytab file, you can authenticate users with the Kerberos server without prompting for a password. Additionally, you can configure LSF Process Manager to acquire a new TGT for users before their TGT expires so that jobs in flows always have the required credentials to run.

---

### Procedure

1. Log on as root to the LSF Process Manager server host and set up the Kerberos keytab file for all user accounts that will be used to run jobs, job arrays, job submission scripts, or job submission script arrays in flows.

Note: The keytab file must be readable by the execution user of the job. LSF Process Manager will change to the execution user to run the command `kinit -k -t <key.tab.file> username`. Refer to your Kerberos documentation for details on how to configure a keytab file.

2. Edit the `js.conf` file, and add parameters to indicate you are using a keytab file with Kerberos.

Add the following parameters in `js.conf`:

- `JS_KRB_USE_KEYTAB=true`: When set to true, this parameter specifies to LSF Process Manager to use the Kerberos keytab file specified by the `JS_KRB_KEYTAB_FILE` parameter to generate user TGTs on behalf of the user before reaching the maximum renewal lifetime. The maximum renewal lifetime is specified in the system's Kerberos configuration file. LSF Process Manager will generate a new ticket using the command `kinit -k -t <key.tab.file>` for the user when the ticket expiration time is within the value set by the parameter `LSB_KRB_RENEW_MARGIN` in `lsf.conf`. By default the `LSB_KRB_RENEW_MARGIN` parameter is set to 60 which will renew the ticket 1 hour before expiration.

- `JS_KRB_KEYTAB_FILE`: Specify the path to the Kerberos keytab file on the LSF Process Manager server host. If you do not specify it, the default value is `/etc/krb5.keytab`.

Note: The `JS_KRB_KEYTAB_FILE` parameter supports `%u` for the job execution user's username. For example, the user can place their own keytab file in the home directory or shared directory and specify `JS_KRB_KEYTAB_FILE=/home/%u/%u.keytab`.

Note: For user, "user1", if `JS_KRB_KEYTAB_FILE=/home/%u/%u.keytab` the file `/home/user1/user1.keytab` must have read permissions by user1:

```
-rw----- user1 group1 /home/user1/user1.keytab
```

For example:

```
JS_KRB_USE_KEYTAB=true
JS_KRB_KEYTAB_FILE=/share/mykrb5.keytab
```

3. Restart the LSF Process Manager server to make changes take effect.

```
jadmin stop  
jadmin start
```

---

## 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.

- [Failover](#)
- [Client and server hosts](#)
- [Variables](#)
- [Configuration](#)
- [Calendars and time zones](#)
- [Local Jobs](#)
- [History](#)

---

## Failover

- [Configure a failover host \(managed by EGO\)](#)  
For enterprise grid orchestrator (EGO) failover to function correctly, Process Manager must have its conf and work directories installed in a shared location. When you install Process Manager as an EGO service, Process Manager benefits from the failover features of EGO. If the server running Process Manager fails, EGO relocates and restarts Process Manager on another host.
- [Install and configure a failover host on UNIX \(managed by failover daemon\)](#)

---

## Configure a failover host (managed by EGO)

For enterprise grid orchestrator (EGO) failover to function correctly, Process Manager must have its conf and work directories installed in a shared location. When you install Process Manager as an EGO service, Process Manager benefits from the failover features of EGO. If the server running Process Manager fails, EGO relocates and restarts Process Manager on another host.

---

## Before you begin

Prerequisites:

Process Manager must have been installed as an EGO service.

- On Windows, this is done by specifying to install Process Manager as an EGO service during installation.
- On UNIX, this is done by specifying EGO\_DAEMON\_CONTROL=true in install.config at installation.

---

## Procedure

1. In js.conf, set JS\_FAILOVER=true and define JS\_FAILOVER\_HOST.

The hosts that you define for JS\_HOST and JS\_FAILOVER\_HOST must both belong to the EGO ManagementHosts resource group.

2. Edit the processmanager.xml file.

- Windows: %LSF\_ENVDIR%\ego\cluster\_name\eservice\esc\conf\services\processmanager.xml
- UNIX: \$LSF\_ENVDIR/ego/cluster\_name/eservice/esc/conf/services/processmanager.xml

3. Change the StartType from MANUAL to AUTOMATIC.

Navigate to the following line:

```
<sc:StartType>MANUAL</sc:StartType>
```

Change it to:

```
<sc:StartType>AUTOMATIC</sc:StartType>
```

4. Add JS\_FAILOVER\_HOST to the ResourceRequirement select statement.

Navigate to the following line:

```
<ego:ResourceRequirement>select('JS_HOST')</ego:ResourceRequirement>
```

Change this line to the following:

```
<ego:ResourceRequirement>select('JS_HOST' || 'JS_FAILOVER_HOST')  
order('JS_FAILOVER_HOST')</ego:ResourceRequirement>
```

5. Save and close the file.

6. Open the Services page from the Windows Control Panel, and change the Startup Type for the Process Manager service from Automatic to Manual.

7. Restart EGO to apply your changes.

You will need to log in to EGO first with the command **egosh user login**. The default user name is **Admin**, and the default password is **Admin**.

```
egosh ego restart all
```

---

## Install and configure a failover host on UNIX (managed by failover daemon)

Note: This failover method is not supported with a SAS license. A full license upgrade of LSF and Process Manager is required.

EGO is the preferred and recommended service for failover. See [Configure a failover host \(managed by EGO\)](#) for instruction on configuring failover through EGO.

Follow this procedure only if you have not installed Process Manager as an EGO service.

Important: Process Manager must have its conf and work directories installed in a shared location.

When you install Process Manager Server, the failover daemon **fod** is automatically installed. You only need to configure the failover host. It is recommended that you do this prior to installing a large number of Process Manager clients, because each client needs to be configured to connect to the failover host automatically if the primary host is unavailable.

Procedure overview:

1. Configure the primary host to recognize the failover host.
2. Prepare the installation files on the failover host.

3. Prepare the configuration on the failover host.
  4. Install Process Manager Server on the failover host, and start the failover host.
- [Configure the primary host](#)
  - [Prepare the installation files on the failover host](#)
  - [Prepare the configuration on the failover host](#)
  - [Install the software on the failover host](#)

---

## Configure the primary host

### Procedure

---

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. For the `JS_FAILOVER` parameter, specify `true`. Be sure to remove the comment character `#`.
5. For the `JS_FAILOVER_HOST` parameter, specify the fully-qualified name of the failover host.
6. Optional. Add `JS_FOD_PORT` parameter and specify the port number of the failover daemon. If you do not specify a port number, it defaults to 1999.
7. Save `js.conf`.
8. Run `jadmin start` to start Process Manager Server and make your changes take effect.

---

## Prepare the installation files on the failover host

### Procedure

---

Make sure that you have access to the Process Manager distribution files.

- a. Copy the installer to the Process Manager directory.
- b. Untar the package (for example, `pm10.2_sas_lnx26-x64.tar`).

```
% tar xvf /usr/share/pmanager/pm10.2_sas_lnx26-x64.tar
```

This creates a directory called `pm10.2_sas_pinstall`. For example:

```
% ls /usr/share/pmanager/pm10.2_sas_pinstall/
```

- c. Copy the Process Manager Server and Process Manager Client distribution files for your operating system to the Process Manager directory. *Do not* untar these files.

---

## Prepare the configuration on the failover host

### Procedure

---

1. Log on to the failover host as `root` or as the primary Process Manager administrator.

2. Make the Process Manager directory current. For example:

```
# cd /usr/share/pmanager/pm10.2_sas_pinstall
```

Logging installation sequence in

/usr/share/pmanager/pm10.2\_sas\_pinstall/pm10.2\_sas\_install/Install.log

3. Copy install.config from the Process Manager Server host to the failover host, replacing the one in the installation package.
4. Edit install.config as follows:
  - a. Add JS\_FAILOVER parameter and specify `true`.
  - b. Optional. For the JS\_FOD\_PORT parameter, specify the port number of the failover daemon. If you do not specify a port number, it defaults to 1999. Be sure to remove the comment character `#`.
5. Save install.config.

---

## Install the software on the failover host

### Procedure

---

1. Run **jsinstall** to start the installation:

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

Logging installation sequence in

/usr/share/pmanager/pm10.2\_sas\_pinstall/pm10.2\_sas\_install/Install.log

2. Select the Process Manager Server. For example:

```
Searching for Process Manager tar files in
/usr/share/pmanager/pm10.2_sas_pinstall please wait ...
1) [SAS] Linux 2.6-glibc2.3-x86 Server
2) [SAS] Linux 2.6-glibc2.3-x86 Client
List the numbers separated by spaces that you want to install. (E.g. 1 3 7,
or
press Enter for all): 1 2
```

3. After the installation is complete, set the Process Manager environment:

- On **cs**h or **tc**sh:

```
# 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.

4. Run **jadmin start** to start the Process Manager daemon on the failover host:

```
# jadmin start
```

---

# Client and server hosts

- [Add a Windows client](#)
- [Run the Process Manager server on system startup](#)
- [Control the Process Manager Server](#)
- [Forcing a system snapshot](#)

---

## Add a Windows client

### Procedure

---

1. Copy pm10.2\_pinstall\_sas\_client\_win64.msi to the desktop or a shared file location from which you can run it.
2. Run pm10.2\_pinstall\_sas\_client\_win64.msi to start the installation.
3. In the Welcome dialog, click Next
4. In the Software License Agreement dialog, click Accept.
5. In the Choose Destination Location dialog, click Next to use the default location; or click Browse... to select a different directory. Click Next.
6. In the Select Components dialog, select the Process Manager Client. Click Next.
7. 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. Optional. In the Failover field, specify a failover host that will be used if the primary host is not available.
  - d. Click Next.
8. Verify that the settings are correct, and click Next to complete the installation.
9. Click Finish.
10. When the installation is complete, if you have a Linux/UNIX Process Manager Server, complete the following steps to connect to the Process Manager Server from a Windows client. Otherwise, skip to step 14.
  - a. Open the configuration file %JS\_ENVDIR%/js.conf and set the parameter JS\_LOGIN\_REQUIRED=true.
  - b. From the Start menu, select IBM Corporation and IBM Spectrum LSF Process Manager, and the appropriate application: Flow Editor, Flow Manager, or Calendar Editor.  
Enter your user name and password when prompted to log on to the Linux/UNIX Process Manager Server.
11. When the installation is complete, from the Start menu, select IBM Corporation and IBM Spectrum LSF 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.

---

## Run the Process Manager server on system startup

## About this task

---

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.

## Procedure

---

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.

---

## Control the Process Manager Server

- [Start and stop 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.

## 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.

### Start the Process Manager daemon

#### Procedure

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

### Stop the Process Manager daemon

#### Procedure

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`.

---

## Start and stop 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.

## Start the Process Manager service

---

### Procedure

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

## Stop the Process Manager service

---

### Procedure

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

---

## Forcing a system snapshot

### About this task

---

Periodically, Process Manager automatically takes a snapshot of the workload in the system and the current status of each work item. The time period between automatic snapshots is determined by the value set in `JS_DATACAPTURE_TIME` in `js.conf`. A snapshot is also taken automatically when Process Manager Server is shut down normally. The information captured is stored in `JS_HOME/work/system`. The information captured in the snapshot is used for recovery purposes, to reconcile job and flow status. The more current the data in the snapshot, the faster the recovery time. When a snapshot is being performed, Process Manager Server pauses its processing—jobs that are running continue to run, but no new work is submitted.

When considering snapshots, you need to balance the time it takes to process the snapshot versus the time it may take to recover from a failure.

It is recommended that you force a snapshot at a time when Process Manager Server is least busy—if that time occurs at a regular interval, schedule it then using the `JS_DATACAPTURE_TIME` parameter in `js.conf`.

### Procedure

---

1. Log on to the Process Manager Server host as **root** or as the primary Process Manager administrator.
2. Run `jadmin snapshot`. The following text appears in the log file:

**Starting  
data capture. This may take a while depending upon system workload.**

When the snapshot is completed, the following text appears in the log file:

**Data capture completed.**



---

# Variables

- [About variables in Process Manager](#)

---

## About variables in Process Manager

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

You can use variables as part or all of a file name to make file names flexible, or you can use them to pass arguments to and from scripts. You 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. You can also use variables in the index expression of a job array definition, in the message sent when a manual job requires completion, or when a job runs.

You 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](#)
- [Scope of variables and variable override order](#)
- [How variables are set](#)
- [How environment variables are set](#)

---

## Types of variables

Process Manager supports three types of variables:

- Built-in variables
- User variables
- Environment variables

---

## Built-in variables

Built-in variables are those defined by Process Manager, where the value is obtained automatically by Process Manager and made available for use by a flow. No special setup is required to use Process Manager built-in variables. You can use these variables in many of the job definition fields in Flow Editor.

---

## User variables

User variables are those created by a user. To use a user variable, you must first create a job that sets a runtime value for the variable and exports it to Process Manager. Once a value has been set for the variable, you can use the variable in many of the job definition fields in Flow Editor.

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

- Flow variables—variables with values that are available only to jobs, job arrays, subflows or events within the current flow. These variables are set with the command **ppmsetvar** -f or in a file specified by JS\_FLOW\_VARIABLE\_FILE.
  - Parent variables are local variables with values that are set at the parent flow scope. If the current flow is the main flow, the variables are set at the main flow scope. These variables are set with the command **ppmsetvar** -p or in a file specified by JS\_PARENT\_FLOW\_VARIABLE\_FILE.

You use the built-in variable JS\_FLOW\_SHORT\_NAME when you need to use the shortened version of the flow name to avoid a potential name conflict issue when using JS\_PARENT\_FLOW\_VARIABLE\_FILE to set parent flow variables.

- Global variables—variables with values that are available to all the flows within the Process Manager Server. These variables are set with the command **ppmsetvar** -g or in a file specified by JS\_GLOBAL\_VARIABLE\_FILE.

User variables can also be used inside environment variables.

Important: When selecting names for user variables, take care not to use the JS\_ prefix in your variable names. This prefix is reserved for system use.

## Environment variables

---

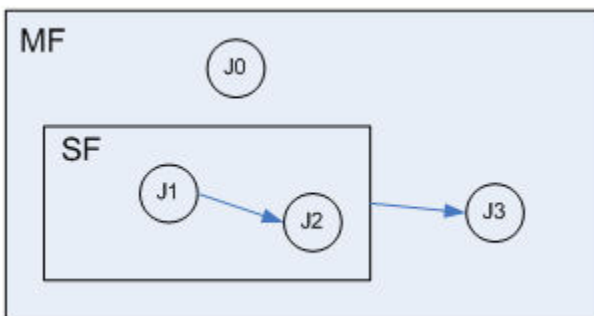
You can submit a job that has environment variables that are used when the job runs. Environment variables can contain user variables.

---

## Scope of variables and variable override order

Variables of the same name specified at different scope levels may override one another. Variables set at an inner subflow scope override those set at an outer subflow scope. This variable override order also applies to default values of input variables.

For example, consider the following flow and job scope levels:



- If the J0 job sets a flow variable A=100, the variable is visible to the main flow MF scope and all subflow scopes (including SF). Therefore, J1, J2, and J3 will all use A=100.
- If J1 sets A=50, J2 will use A=50 because the variable set at the MF\_SF subflow scope overrides the variable set at the main flow MF outer scope. However, J3 still uses A=100 because the value at the

main flow MF scope is still A=100. J2 uses A=50 even if J0 sets A=100 after J1 sets A=50.

This variable override order also applies to default values of input variables. For example,

- If main flow MF has an input variable IV with a default value of 200, and SF does not have input variables, J0, J1, J2, and J3 will all use IV=200.
- If subflow SF now has the same input variable IV with a default value of 20, J0 and J3 will use IV=200, while J1 and J2 will now use IV=20.
- If J0 sets IV=30, it overrides the default value at the MF scope, but not at the MF:SF subflow scope. Therefore, J1 and J2 will use IV=20, while J3 will use IV=30.
- If J1 sets IV=5, J2 will use IV=5, while J3 still uses IV=30.

Similarly, if you trigger a flow with variables, the variables will only override the default values at the main flow level, but not the default values at subflows. However, if you specified no default values in the subflow, then the specified values are also visible to the subflow.

The variables set by the job have similar scope to variables in any programming language (C, for example). If the job sets the variable with the command **ppmsetvar** -f or in the file specified by JS\_FLOW\_VARIABLE\_FILE within a subflow, the scope of the variable is limited to the jobs and events within the subflow. This means that the variable is only visible to that subflow and is not visible to the main flow or any other subflows. 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.

If the job sets variables with the command **ppmsetvar** -p or in the file specified by JS\_PARENT\_FLOW\_VARIABLE\_FILE within a subflow, the user variable is passed to the parent flow.

Flow 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.

Environment variables are set in the job definition and the job runs with the variables that are set.

If you use **ppmsetvar** to set user variables and you use **ppmsetvar** multiple times, the variables will be appended. For example, if you run the following, the end result will be a=10, b=2, c=7, and d=100:

```
ppmsetvar -f a=1 b=2
ppmsetvar -f a=10 c=7
ppmsetvar -f d=100
```

If you use **ppmsetvar** in conjunction with other methods of setting user variables in Process Manager, such as a variable file or job starter, note that the variable file can override any variables set with **ppmsetvar** as it is read last.

## Dynamic subflows

---

When specifying input variable values for dynamic subflows, the same rules apply because the specified values are effectively treated as default values of the input variables.

## How variables are set

### How user variables are set

---

User variables are set using the following methods:

- External file
- The command `ppmsetvar`

## External file

This method requires a shared filesystem. The `jfd` work directory must be on a shared filesystem accessible by all your jobs.

Process Manager can set user variables by writing to an external file.

Any binary or script will work, as long as it can write to the file. Process Manager sets environment variables for each job or job array: `JS_FLOW_VARIABLE_FILE`, `JS_GLOBAL_VARIABLE_FILE`, and `JS_PARENT_FLOW_VARIABLE_FILE`. In addition, LSF® sets the `LSB_JOBINDEX` environment variable for job arrays to indicate the index of each job array element.

For jobs to set flow variables, the job must write to the file specified by the `JS_FLOW_VARIABLE_FILE` environment variable. For jobs to set global variables, the job must write to the file specified by the `JS_GLOBAL_VARIABLE_FILE` environment variable. For jobs to set parent flow variables, the job must write to the file specified by the `JS_PARENT_FLOW_VARIABLE_FILE` environment variable.

For job arrays to set flow variables, the job array must be able to write to the file specified by the `JS_FLOW_VARIABLE_FILE[LSB_JOBINDEX]` environment variable; for job arrays to set global variables, the job array must write to the file specified by the `JS_GLOBAL_VARIABLE_FILE[LSB_JOBINDEX]` environment variable; and for job arrays to set variables for parent flows, the job array must write to the file specified by `JS_PARENT_FLOW_VARIABLE_FILE[LSB_JOBINDEX]`.

The jobs or job arrays write to the files in the following format (each line contains a variable-value pair):

```
VAR1=VALUE1
VAR2=VALUE2
...
```

The values must not contain semicolons (;) or control characters. Process Manager will not initially create these files — the files need to be created by the jobs.

The following example illustrates a script fragment for jobs that assigns file names to set flow, global, and parent flow variables:

```
${JS_FLOW_VARIABLE_FILE};
${JS_GLOBAL_VARIABLE_FILE};
${JS_PARENT_FLOW_VARIABLE_FILE};
```

The following example illustrates a Perl script fragment for job arrays that assigns file names to set flow, global, and parent flow variables:

```
$flowVarFile = $ENV{JS_FLOW_VARIABLE_FILE} . "[" . $ENV{LSB_JOBINDEX} . "];
$globalVarFile=$ENV{JS_GLOBAL_VARIABLE_FILE} . "[" . $ENV{LSB_JOBINDEX} . "];
$parentflowVarFile=$ENV{JS_PARENT_FLOW_VARIABLE_FILE} . "[" . $ENV{LSB_JOBINDEX} .
"]";
```

## The `ppmsetvar` command

Only available with Process Manager 9.1 and LSF 9.1 and higher.

You can use the command **`ppmsetvar`** from an LSF job, job script, job array and job script array to pass user variables from a subflow to a parent flow, to set user variables that are used only within a flow, or to set global user variables used by all flows in the system. You can also use **`ppmsetvar`** to remove specific user variables. You do not need a shared filesystem with `ppmsetvar`.

Important: This command uses the LSF **bpost** command with slots 4, 5, and 6. If anyone is using **bpost** in your LSF cluster, ensure the slots 4, 5, 6 are not used as this will interfere with the **ppmsetvar** command and may lead to unexpected results.

## Setting variables that can be used only by work items within a flow

The following example shows how set a user variable that can be used by all work items within a flow using the command **ppmsetvar**.

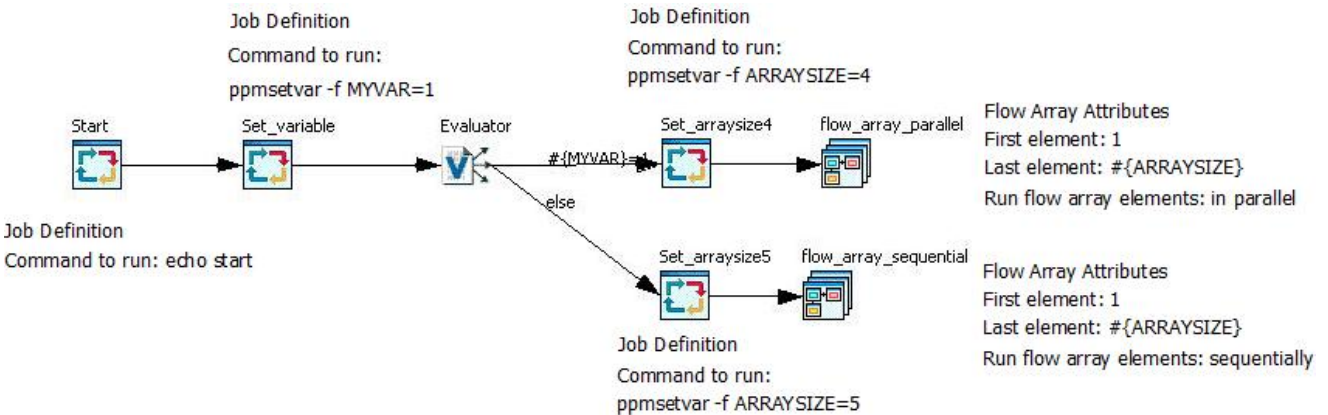
This flow contains two subflows to be run as arrays and a condition evaluator that decides whether to run the arrays in parallel or sequentially. The Set\_variable job sets the variable MYVAR=1, which indicates to run the array in parallel. This flow also sets the arraysize at the time the flow array runs.

In this example, the job Set\_variable sets MYVAR=1 with ppmsetvar.

In the variable evaluator, when MYVAR=1, the job Set\_arraysize4 runs. The job Set\_arraysize4 sets the variable ARRAYSIZE=4 with **ppmsetvar**.

In the variable evaluator, when MYVAR is equal to any other number, the job Set\_arraysize5 runs. The job Set\_arraysize5 sets the variable ARRAYSIZE=5 with **ppmsetvar**.

The flow arrays that follow use the variable set by the the jobs Set\_arraysize4 or Set\_arraysize5 to define how many times the subflows are run as flow arrays.



## Passing Variables between parent flows and subflows

The following example shows how to pass variables from a parent flow to be used by a subflow, and then how to pass a variable from a subflow to its parent using the command **ppmsetvar**.

This flow contains two dynamic subflows and passes the variable MYVAR=100 to one subflow as an input variable to the flow, and MYVAR=200 to the other subflow as an input variable.

Jobs J1 and J2 write the value passed from the subflow to an output file. The output of J1 is xyz100 and the output of J2 is xyz200.

The last job in the subflow passes the variable result\_#{JS\_FLOW\_SHORT\_NAME}=xyz#{MYVAR} to the parent flow and also writes the variable to a file. The parent flow accesses the user variable set by the subflow

by indicating the subflow name such as **echo result\_Dynamic\_Subflow1** and **echo result\_Dynamic\_Subflow2**.

Note: The subflows use the built-in variable `{JS_FLOW_SHORT_NAME}` to avoid potential naming conflicts with the parent flow.

Figure 1. Parent flow

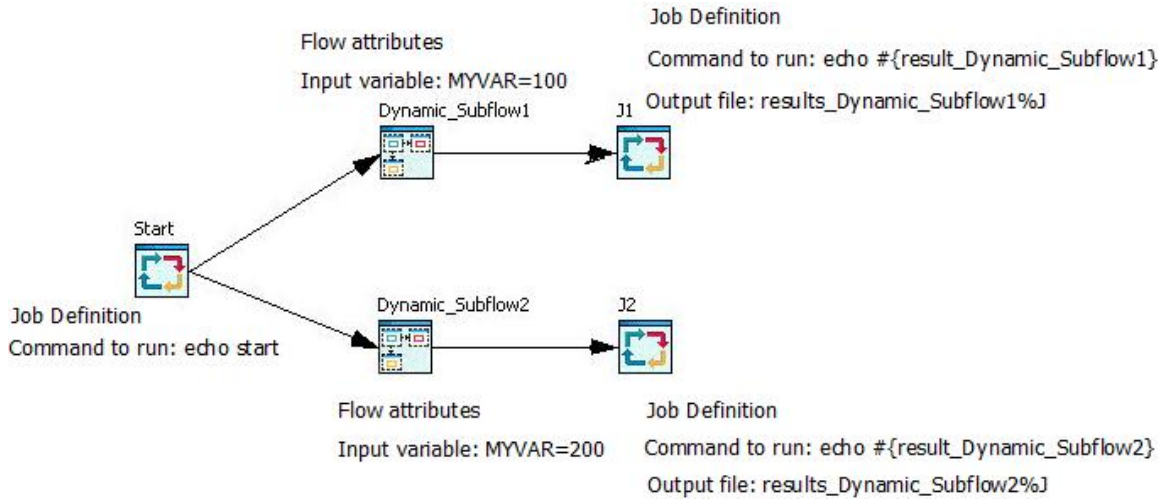
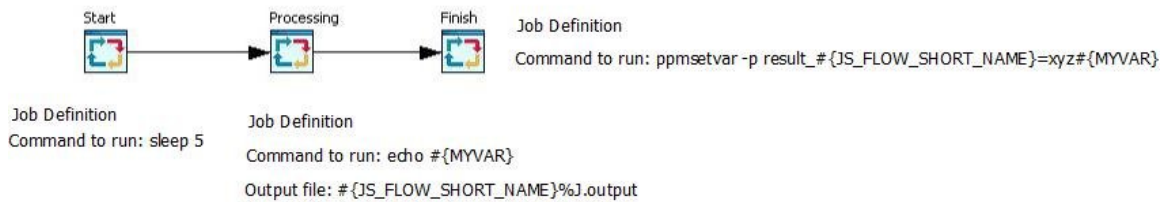


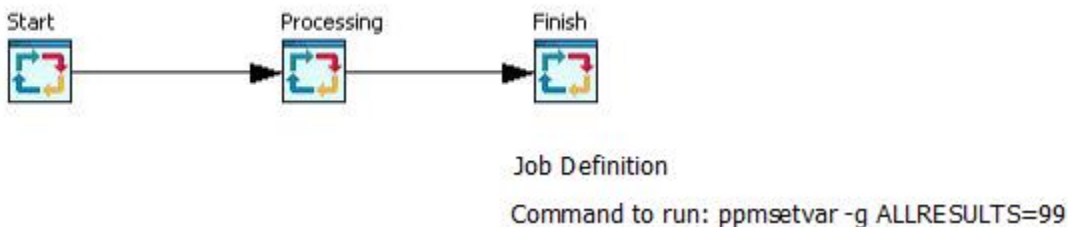
Figure 2. Subflows Dynamic\_Subflow1 and Dynamic\_Subflow2



## Setting a global variable that can be used by any flows in the system

The following example shows how set a variable that can be used by all flows in the system with the command **ppmsetvar**.

In the following example, the last job sets the global variable `ALLRESULTS` to the value 99. This variable can be used by any flow in the system.



## How environment variables are set

For environment variables, a new job attribute is created to store the environment variables. In a Linux® environment, a script file is written to a temporary directory to run the `bsub` command. In a Windows

environment, a temporary directory is used to create and run batch files. The system tries the following directories until it finds one that is writable:

- %TEMP%
- %TMP%
- C:\

---

## Configuration

- [Change the Configuration](#)

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

- [Output and error file generation for work items in a flow](#)

- [Define a default working directory for flows](#)

- [Configure an alarm](#)

- [Specify the mail host](#)

- [Change the job start retry value](#)

- [Converting the job command line to native encoding when jobs are submitted to LSF](#)

By default, the job command line that is specified in the job definition is converted to Unicode by Process Manager. To support localized command-lines, the job command line needs to be converted to native encoding when submitted to LSF. You can configure Process Manager and LSF to support native encoding.

---

## Change the Configuration

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

---

## Change a configuration value on UNIX

### Procedure

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.

---

## Change a configuration value on Windows

### Procedure

1. Stop the Process Manager Server service.
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.

---

## Output and error file generation for work items in a flow

By default, output and error files are not generated for flows or individual work items.

To troubleshoot flows, however, it is useful to always generate output and error files for work items in the flow.

You can set output and error file generation in the Flow Attributes. The behavior to create output and error files is the same as using the LSF® bsub command options -o and -e.

Output and error file settings that are defined in the Flow Attributes are inherited by the following work items in the flow:

- Jobs
- Local jobs
- Job arrays
- Static subflows
- Static flow arrays
- Dynamic subflows
- Dynamic flow arrays

Output and error file settings do not apply to job scripts, job array scripts, template jobs, or manual jobs.

Users can override output and error file settings that were defined in the flow in individual work items.

---

## Default location of output and error files

By default, output and error files are generated in the working directory of the work item.

If the working directory is not specified, the location that is used for the working directory is the execution user's home directory:

- Linux®: \$HOME
- Windows: %HOMEDRIVE%%HOMEPATH%

You can define a default working directory for flows with the parameter JS\_DEFAULT\_FLOW\_WORKING\_DIR in js.conf. Users can override the default working directory in individual work items.

---

## Override order for output and error file generation settings



The override order for Process Manager to determine output and error file generation, location, and naming is as follows (in order of highest precedence):



1. Use output and error file settings defined at the job level, in the Job Definition.
2. Use output and error file settings defined in the static subflow's Flow Attributes.
3. Use output and error file settings defined in the dynamic subflow's Flow Attributes:
  - If Use parent flow's settings is selected, use settings in the parent flow's Flow Attributes.
  - If Use inserted flow's settings is selected, use settings in the inserted target flow's Flow Attributes.
  - If Override parent flow's settings and inserted flow's settings is selected, use settings in the Dynamic subflow Flow Attributes.
4. Use output and error file settings defined in the flow's Flow Attributes.

## Configuring output and error file generation for work items in a flow

---

### Procedure

1. Define a default working directory for the flow by setting JS\_DEFAULT\_FLOW\_WORKING\_DIR in js.conf. This is useful to centralize output and error file generation when no working directory is defined for the flow in the Flow Definition or with the parameter JS\_FLOW\_WORKING\_DIR.

Refer to [JS\\_DEFAULT\\_FLOW\\_WORKING\\_DIR](#) for details on requirements for the default working directory.

2. Restart the Process Manager server to make changes take effect.
3. In Flow Editor, from the menu select Action...>Add Flow Attribute, and define output and error file settings.

| Field                                       | Description  |
|---|--|
| Create output files for jobs and job arrays | Select Yes to configure output and error file generation and naming pattern.<br><br>Important: If you select to only generate output files, no error files are generated but the content of the error file is appended to the output file(same behavior as the LSF <b>bsub -o</b> option). |
| Create error files for jobs and job arrays  |  |
| Directory                                   | Specify a directory name or path relative to the flow's working directory. Process Manager creates specified subdirectories in the working directory if the directories do not exist.  |

| Field     | Description   |
|-----------|---|
| File Name | <p>Specify the naming pattern for files.</p> <p>Built-in variables you can use:</p> <ul style="list-style-type: none"> <li>• %u for user name</li> <li>• %t for time stamp</li> <li>• %J for job ID</li> <li>• %I for job array element</li> </ul> <p>Note: If you specify the file naming pattern to start with a path and use a slash (/), this is interpreted as an absolute path by Process Manager. For example: if your working directory is /home/user1, and you specify the file naming pattern for output files to be /test/output/output.#{JS_FLOW_FULL_NAME}.%J, the output file will be created outside of the working directory in the directory /test/output, not within the working directory.</p> <p>Example file naming pattern for output files:</p> <ul style="list-style-type: none"> <li>• Job: output.#{JS_FLOW_FULL_NAME}.%J</li> <li>• Local job: output.#{JS_FLOW_FULL_NAME}</li> <li>• Job array element: output.#{JS_FLOW_FULL_NAME}.%J[%I]</li> </ul> <p>Example file naming pattern for error files:</p> <ul style="list-style-type: none"> <li>• Job: error.#{JS_FLOW_FULL_NAME}.%J</li> <li>• Local job: error.#{JS_FLOW_FULL_NAME}</li> <li>• Job array element: error.#{JS_FLOW_FULL_NAME}.%J[%I]</li> </ul> |

4. Click OK to save your changes.

## Define a default working directory for flows

You can define a default working directory for flows with the parameter `JS_DEFAULT_FLOW_WORKING_DIR` in `js.conf`. The default working directory is used when no working directory is defined in the flow definition or passed to Process Manager with the variable `JS_FLOW_WORKING_DIR`.

Work items inherit the flow settings but users can override the default working directory in individual work items.

## Configure an alarm

### About this task

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.

Alarms are stored in the directory `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 either notify one or more email addresses, or execute a script.

The alarm file contains the following parameters:

```
DESCRIPTION=<description>  
NOTIFICATION=CMD [command_name command_parameters]
```

Any alarm files with an invalid alarm definition will not be registered. Any extra unrecognized parameters are ignored, but the alarm will still be registered.

## Procedure

---

1. As the Process Manager administrator, create a new file in the directory *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 the alarm type and definition.

- Email notification

```
NOTIFICATION=Email[user_name ...]
```

Specify the "**Email**" command, followed by a space-delimited list of email addresses to notify regarding the exception. Specify the complete email address, or just 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.

- Script execution

```
NOTIFICATION=CMD[/file_path/script_file user_variable ...]
```

Specify the "**CMD**" command, followed by the path to the script file and any user variables (such as the error code). For example:

```
NOTIFICATION=CMD[/home/admin/pageadmin.sh #{ERRORCODE}]
```

Variable values cannot contain the backquote character (```).

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

---

## Specify the mail host

### About this task

---

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.

## Procedure

---

1. 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.

---

## Change the job start retry value

### About this task

---

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.

## Procedure

---

1. 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.

---

## Converting the job command line to native encoding when jobs are submitted to LSF

By default, the job command line that is specified in the job definition is converted to Unicode by Process Manager. To support localized command-lines, the job command line needs to be converted to native encoding when submitted to LSF. You can configure Process Manager and LSF to support native encoding.

## Procedure

---

1. Set the parameter JS\_UNICODE\_CMD\_UPLUS=true in the LSF configuration file js.conf. This configuration enables the job command to run in native encoding when a job is submitted to the LSF unicodecmd queue.

This queue is configured at installation in the LSF configuration file lsb.queues with the exact same specifications as the queue normal with an extra job starter jsunicodestarter on UNIX, jsunicodestarter.bat on Windows. Before a job is submitted to the operating system on the execution host, the job starter converts the job command to native encoding.

unicodecmd queue configuration on UNIX:

```
Begin Queue
QUEUE_NAME      = unicodecmd
PRIORITY        = 30
JOB_STARTER     = "$JS_HOME/10.2/bin/jsunicodestarter"
NICE             = 20
DESCRIPTION     = For normal low priority jobs submitted from Process Manager \
with Unicode escape sequence on command line.
End Queue
```

unicodecmd queue configuration on Windows:

```
Begin Queue
QUEUE_NAME      = unicodecmd
PRIORITY        = 30
JOB_STARTER     = "%JS_HOME%\10.2\bin\jsunicodestarter.bat"
NICE            = 20
DESCRIPTION     = For normal low priority jobs that are submitted from Process
Manager \
with Unicode escape sequence on command line.
End Queue
```

2. Submit a job the unicodecmd queue.

## What to do next

---

If you require both Unicode to native encoding conversion and user variable handling, define a queue in LSF and add the job starter jsunivarstarter in the queue configuration. Submit the job to the queue. The command line is converted to native encoding along with user variables before the job is submitted.

For example, on UNIX, add the following job starter to the queue configuration in the LSF configuration file lsb.queues:

```
JOB_STARTER = "$JS_HOME/10.2/bin/jsunivarstarter"
```

For example, on Windows, add the following job starter to the queue configuration in the LSF configuration file lsb.queues:

```
JOB_STARTER = "%JS_HOME%\10.2\bin\jsunivarstarter.bat"
```

---

## Calendars and time zones

- [Create 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, then save it as a system calendar.

- [Calendar names](#)
- [Update the Holidays@Sys calendar](#)
- [Delete a calendar](#)
- [Updating time zone data](#)

Time zone and daylight savings time (DST) are often adjusted by individual governments around the world according to their local rules. LSF Process Manager provides a build method for International Components for Unicode (ICU) data and adds a dynamic method for applying ICU data updates. It also provides a parameter to offset all time events in LSF Process Manager, if necessary.

---

## Create 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, then save it as a system calendar.

---

## Calendar names

---

### About this task

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, underscore (`_`), and dash (`-`)
- Calendar names cannot begin with a number
- System calendars are named as follows:

`calendar_name@Sys`

---

## Procedure

1. Start Calendar Editor as the Process Manager administrator.
2. Create the calendar.
3. Select Calendar > Save as System Calendar to save it.

---

## Update the Holidays@Sys calendar

---

### Procedure

1. Open the Holidays@Sys calendar.
2. Change the owner of the calendar to a user name other than `Sys`.

3. Save the calendar with the new owner.
4. Edit the list of dates to include all those dates that are company-wide holidays.
5. Select Calendar > Save as System Calendar to save the calendar with the name Holidays with **Sys** as the owner.

---

## Delete a calendar

### About this task

---

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.

You cannot delete a calendar that is currently in use by a flow definition, flow, or another calendar. A calendar is in use under the following conditions:

- If a flow definition is triggered by a time event that uses the calendar, or uses a calendar that references this calendar
- If a flow is running, and contains a time event that uses the calendar or uses a calendar that references this calendar
- If another calendar references this calendar to build a schedule statement

You can temporarily delete a system calendar—installing a new version of Process Manager Server reinstalls the system calendars that come with Process Manager.

### Procedure

---

1. Stop Process Manager Server.
2. In *JS\_TOP/work/calendars*, locate the calendar you want to delete.
3. Delete the file from the calendars directory.
4. Restart the Process Manager to have the change take effect.

---

## Updating time zone data

Time zone and daylight savings time (DST) are often adjusted by individual governments around the world according to their local rules. LSF Process Manager provides a build method for International Components for Unicode (ICU) data and adds a dynamic method for applying ICU data updates. It also provides a parameter to offset all time events in LSF Process Manager, if necessary.

### About this task

---

ICU 57.1 for Unix and Windows is used as an example in the following procedure. The procedure for other versions of ICU are similar. Find instructions for other versions of ICU:

<http://userguide.icu-project.org/datetime/timezone#TOC-Updating-the-Time-Zone-Data>

<http://download.icu-project.org/files/icu4c/>

Important: Rebuilding the ICU data lib l(icu4c) with the data (2019a) can be performed on all platforms except HP-UX.

## Procedure

---

1. Download the ICU source code.

Download the ICU packages from <http://download.icu-project.org/files/icu4c/57.1>

The following packages are required:

- icu4c-57\_1-src.zip for Windows
- icu4c-57\_1-src.tgz for Unix

2. Download the latest ICU data files.

Go to <https://github.com/unicode-org/icu-data/tree/master/tzdata/icunew>

For example, go to the directory 2018i/44/le (where **2018i** is the latest ICU data for the year 2018, **44** is the directory for updates to ICU version 4.4 and newer, and **le** is the directory for Little Endian processors, including all Intel processors).

The following ICU data files are required:

- zoneinfo64.res
- windowsZones.res
- timezoneTypes.res
- metaZones.res

3. Build the ICU data library for Windows.

- a. Unzip the package icu4c-57\_1-src.zip to a convenient location.

In this example, they are extracted to: C:/icu4c-57\_1-src

- b. Open the source/allinone/allinone.sln workspace file in Microsoft Visual Studio.

- c. Set the active platform to "Win32" or "x64" and configuration to "Release" or "Debug".

- i. Choose Build. Select Configuration Manager.

- ii. Select Release or Debug for the Active Solution Configuration.

- iii. Select Win32 or x64 for the Active Solution Platform.

- d. Choose Build. Select Build Solution to build ICU.

- e. Update the ICU data files:

```
set PATH=%PATH%;C:/icu4c-57_1-src/bin
```

Check that the icupkg tool is available with the command `icupkg -h`

- f. Copy the new ICU data files \*.res to the directory source/data/in/

- g. Execute the following commands to update the ICU data file:

```
icupkg -a zoneinfo64.res icudt571.dat
icupkg -a windowsZones.res icudt571.dat
icupkg -a timezoneTypes.res icudt571.dat
icupkg -a metaZones.res icudt571.dat
```

- h. Choose Build. Select Rebuild Solution to rebuild the ICU.

The new ICU data library is located at C:/icu4c-57\_1-src/bin/icudt57.dll

4. Build the ICU data library for UNIX.

- a. Compile the ICU tool and libraries.

Execute the following commands to compile the ICU:

```
gunzip -d < icu4c-57_1-src.tgz | tar xvf -
cd icu/source
chmod +x runConfigureICU configure install-sh
./runConfigureICU Linux --with-data-packaging=library
make or gmake
```



Note: Linux is used in the following example for runConfigureICU. For other UNIX platforms get help using the command `./runConfigureICU -h`

b. Install the ICU tool.

Note: The new **icupkg** tool is located at `./bin/icupkg`. The libraries that the **icupkg** tool uses are located at `./lib/`.

Execute the following commands to install the **icupkg** tool:

```
cd icu/source
chmod +x ./bin/icupkg
cp ./bin/icupkg /usr/sbin/
```

Run the command `ldd /usr/sbin/icupkg` to check the dependencies for the **icupkg** tool. For example:

```
ldd icupkg
linux-vdso64.so.1 (0x00007b813bc10000)
libcicutu.so.57 => not found
libstdc++.so.6 => /usr/lib/powerpc64le-linux-gnu/libstdc++.so.6
(0x00007b813b990000)
libgcc_s.so.1 => /lib/powerpc64le-linux-gnu/libgcc_s.so.1
(0x00007b813b950000)
libc.so.6 => /lib/powerpc64le-linux-gnu/libc.so.6 (0x00007b813b720000)
libm.so.6 => /lib/powerpc64le-linux-gnu/libm.so.6 (0x00007b813b5d0000)
/lib64/ld64.so.2 (0x00007b813bc30000)
```

Note: If you encounter the error `libcicutu.so.57 not found`, copy the ICU libraries to `/lib/` in the same directory as `libc.so.6`: `cp ./lib/libicu*.so.57 /lib/`

Run the command `icupkg -h` to check that the tool is now available.

c. Update the ICU data files.

Copy the new ICU data files `*.res` to the directory `source/data/in/`.

Execute the following commands to update the ICU data file:

```
cp ./source/data/in/icudt571.dat ./source/data/in/icudt571.dat.bak
icupkg -a zoneinfo64.res icudt571.dat
icupkg -a windowsZones.res icudt571.dat
icupkg -a timeZoneTypes.res icudt571.dat
icupkg -a metaZones.res icudt571.dat
```

d. Compile the ICU data library.

Execute the following commands to compile ICU data:

```
cd icu/source/data
make or gmake clean
make or gmake
```

The new ICU data library is located at `icu/source/lib/libicudata.so.57.1`.

5. Apply the ICU data library.

a. Copy the new ICU data library `libicudata.so.57.1` to `$JS_SERVERDIR/./lib/` for UNIX.

b. Copy the new ICU data library `icudt57.dll` to `%JS_SERVERDIR%` for Windows.

c. Restart the LSF Process Manager server.

All previously triggered flows will not be affected. Any newly triggered flows by a time event will use the new ICU time zone data.

6. Configure the LSF Process Manager properties file for the correct time zone.

a. Open the `timezones.properties` file for the LSF Process Manager client.

If the name and time zone of cities needs to be corrected, update the `timezones.properties` file. Search for city time zones using the site <https://greenwichmeantime.com/time-zone>

b. Update the package properties files for LSF Application Center

Update the following files with the latest time zones:

- export PPM\_DIR=\$GUI\_TOP/3.0/wlp/usr/servers/platform/apps/platform.war/WEB-INF/classes/com/platform/gui/ppm/
- \$PPM\_DIR/view/package\_zh\_CN.properties
- \$PPM\_DIR/view/package.properties
- \$PPM\_DIR/view/package\_en.properties

c. Restart the application.

**Restart LSF Application Center:**

```
pmcadm stop;pmcadm start
```

**Restart LSF Process Manager:**

```
jadmin stop;jadmin start
```

## What to do next

---

### Configure JS\_TIME\_EVENT\_OFFSET

If a country temporarily changes their time zone (for example, delaying Daylight Savings Time (DST) by a week), or the latest time zone data from <https://github.com/unicode-org/icu-data/tree/master/tzdata/icunew> does not contain the required time zone changes, it may be necessary to use the JS\_TIME\_EVENT\_OFFSET parameter as a temporary measure.

The JS\_TIME\_EVENT\_OFFSET parameter specifies the time event offset to adjust the LSF Process Manager server time. The server time will add the offset to its time and all time events of a flow will be triggered according to the adjusted server time. The valid range is -180 to 180 minutes.

When the time zone offset is no longer required, JS\_TIME\_EVENT\_OFFSET can be disabled (set to 0) and LSF Process Manager restarted so that time events are scheduled at the current time zone.

---

## Local Jobs

- [About local jobs on Linux and UNIX](#)
- [About local jobs on Windows](#)

---

## About local jobs on Linux® and UNIX

You can include a local job in the flow diagram.

A local job is a job that will execute immediately on the Process Manager host without going through LSF®. A local job is usually a short and small job. It is not recommended to run long, computational-intensive or data-intensive local jobs as it can overload the Process Manager host.

A local job is non-blocking: that is, several local jobs can run in parallel.

## Controlling a local job

---

You can kill a local job in the same way as you kill any other job. The local job may also be killed as a result of the flow being killed.

If you suspend or resume a flow that contains local jobs, the local jobs will also be suspended or resumed.

In some cases, you may not want to suspend a local job when the flow is suspended. You can select Disable suspension for this job in the Job Definition. If job suspension is disabled, and the job is running when the flow is suspended, the job will not be suspended. The job will continue to run.

The following signals are sent to the local job:

- Kill—The system sends SIGINT, waits for 10 seconds, SIGTERM, waits for 10 seconds, then SIGKILL. The 10 second delay between signals allows you to catch the signal and perform any cleanup required by the job before it is terminated.
- Suspend—The system sends SIGSTOP.
- Resume—The system sends SIGCONT.

In the job's runtime attributes, you can view the exit status and CPU usage of a local job after the job completes. The process ID identifies the local job and you can view CPU usage for the job. You can also view the process ID of the job and CPU usage information with `jflows -l flow_id` and `jhist -C job`.

## Parameters related to local jobs

---

By default, a local job can run indefinitely, it does not have a timeout. To define a timeout value for a local job so that it will be killed if it was running for too long, use the parameter `JS_LOCAL_EXECUTION_TIMEOUT` in `js.conf`.

To avoid overloading the Process Manager host with too many local jobs, the parameter `JS_LOCAL_JOBS_LIMIT` in `js.conf` controls the maximum number of local jobs that can run concurrently on the Process Manager host.

For security reasons, you may want to disable local jobs altogether. You can disable local jobs by setting the parameter `JS_LOCAL_JOBS_LIMIT=0` in `js.conf`.

## jfd and eem.local

---

To monitor local jobs, `jfd` communicates with `eem.local`. This binary is started by `jfd`, handles job submission, control, and status checking for local jobs, and reports back to `jfd`.

`jfd` listens on the port number `JS_PORT + 1` to receive status updates from `eem.local`, and `eem.local` listens on port number `JS_PORT + 2`. The parameter `JS_PORT` is defined in `js.conf`.

Should `jfd` terminate abnormally, when it restarts it can recover running and finished local jobs and determine their status and resource usage.

## About local jobs on Windows

---

You can include a local job in the flow diagram.

A local job is a job that will execute immediately on the Process Manager host without going through LSF®. A local job is usually a short and small job. It is not recommended to run long, computational-intensive or data-intensive local jobs as it can overload the Process Manager host.

A local job is blocking: each local job has its own thread for execution, but the dedicated local job thread will not be freed up to execute another local job until the local job that is executing has completed.

## Controlling a local job

---

You cannot directly kill a local job in the same way as you kill any other job. The local job can only be killed as a result of the flow being killed, or if it runs for longer than the configured timeout value.

If you suspend or resume a flow that contains local jobs, the local jobs will be killed and rerun.

You can view a local job's runtime attributes in Flow Manager. Note, however, that no resource usage is available for the local job.

## Parameters related to local jobs

---

By default, a local job has a timeout so that it will be killed if it was running for too long. The parameter `JS_LOCAL_EXECUTION_TIMEOUT` in `js.conf` defines how long a local job is allowed to run before it is killed by the system.

For security reasons, you may want to disable local jobs altogether. You can disable local jobs by setting the parameter `JS_LOCAL_JOBS_LIMIT=0` in `js.conf`.

---

## History

- [Change the history setting](#)
- [View History](#)
- [View the history of a flow definition](#)
- [View the history of a flow](#)
- [View the history of a job or job array](#)

---

## Change the history setting

### About this task

---

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.

## Procedure

---

1. Follow the instructions in [“Changing the Configuration”](#) 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
# JS_HISTORY_ARCHIVE_DIR=/share/archive
```

```
# JS_HISTORY_CLEAN_PERIOD=15
```

and set them :

- 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.
- d. Set JS\_HISTORY\_ARCHIVE\_DIR to specify the path and name to the directory in which history log files are archived.

The directory specified by JS\_HISTORY\_ARCHIVE\_DIR must have the same owner and permission as JS\_HOME/work/history/. The directory must be owned and must be writable by the Process Manager administrator, and must be readable by everyone.

Any history log files older than the time period specified by JS\_HISTORY\_CLEAN\_PERIOD are moved to the directory specified by JS\_HISTORY\_ARCHIVE\_DIR.

Important: When JS\_HISTORY\_ARCHIVE\_DIR is not defined, any history log files older than the time period specified by JS\_HISTORY\_CLEAN\_PERIOD are deleted by Process Manager.

- e. Set JS\_HISTORY\_CLEAN\_PERIOD to indicate the time period in days for which history log files are stored before they are moved to the archive directory. The default is 15 days.  
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.

---

## View History

You can see the history of a work item, which shows details about when and how the item was run, by using the Flow Manager or **jhist**.

When you use the **jhist** command with no time interval specified, you see data for the past seven days.

---

## 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 Client

---

## 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.

---

## 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 Client

## 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.

---

## 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 Client

## 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.

# Using Process Manager with Other Batch Systems

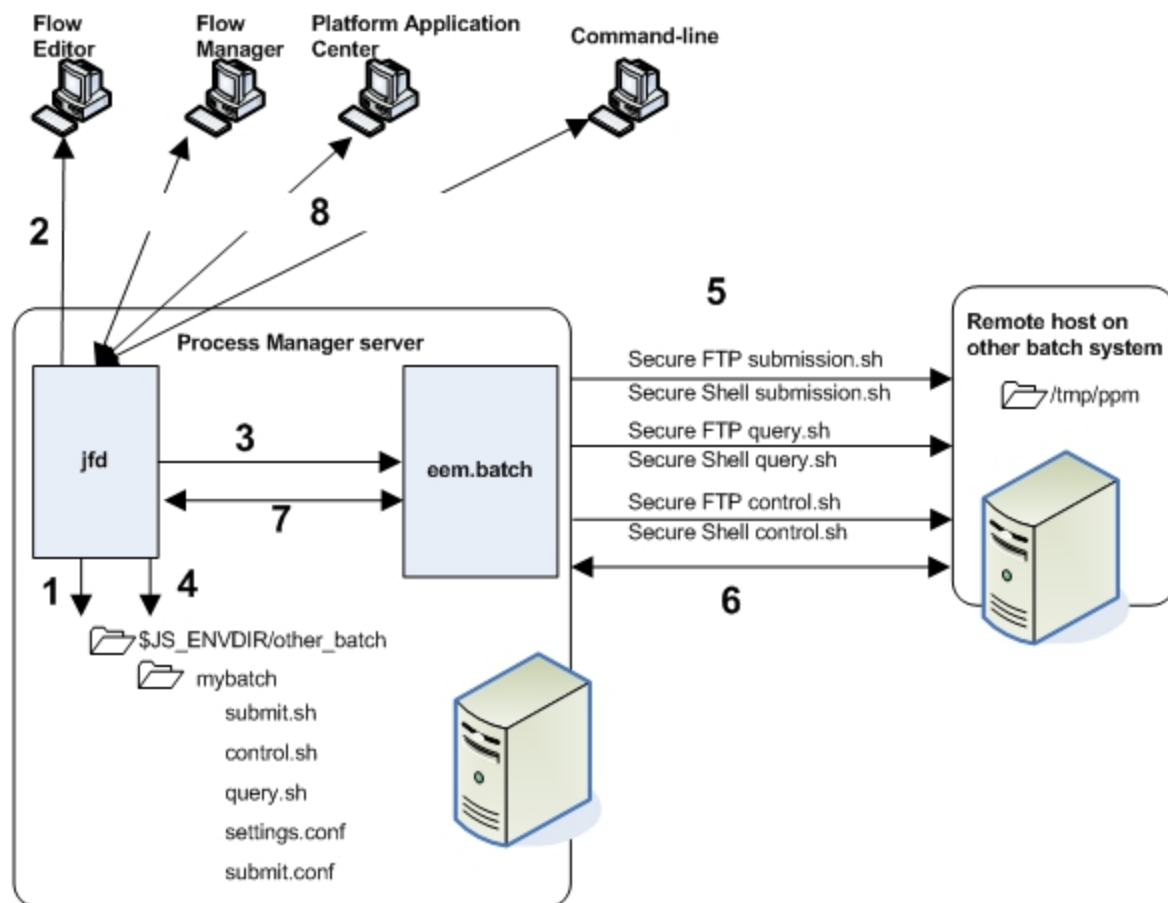
In your flow, you may have jobs that you need to run on remote batch systems or workload managers that are not LSF®. Process Manager can be configured by the administrator to communicate with the Other Batch System so that users can add a job in their flow and submit a job to the Other Batch System, and track and control the job like any other job in their flow. The Job Definition for Other Batch jobs can be customized by the administrator.

- [How Process Manager works with Other Batch Systems](#)
- [About Other Batch Jobs](#)
- [Configuring Process Manager to work with Other Batch Systems](#)
- [Configure Data Transfer to and from the Other Batch System](#)

Before running a job on the Other Batch System, you may need to transfer data from the local system to the Other Batch System as input to the job.

- [Troubleshooting Other Batch System Jobs](#)

## How Process Manager works with Other Batch Systems



1. When `jfd` starts up, it scans the `$JS_ENVDIR/other_batch` directory for any configured batch systems and makes them available in Flow Editor.

`jfd` starts up one `eem.batch` for each configured batch system.



The Other Batch Job work item in Flow Editor is now enabled.

2. In Flow Editor, the user selects the Other Batch Job work item and adds it to their flow.

In the Job Definition, the user indicates the command to run, other job submission options, and the user account under which the job is to run: the same user as on the local system or a specific user on the Other Batch System.

3. eem.batch handles job submission, control, and status checking for jobs submitted to the specific batch system, and reports back to jfd.
4. To connect to the Other Batch System, jfd uses the file settings.conf located in the subdirectory for the specific batch system under \$JS\_ENVDIR/other\_batch.

The settings.conf file indicates which host on the Other Batch System to connect to, the Other Batch System administrator account with which to connect for job querying, and the temporary directory to use on the Other Batch System.

jfd uses Secure Shell(SSH) to connect to the Other Batch System.

5. For job submission, control, and status checking, jfd uses the configured bash shell scripts submit.sh, control.sh, and query.sh and populates the environment variables in these scripts with the correct values for the Other Batch System.

As required, jfd uses SSH to securely copy submit.sh, control.sh, and query.sh to the temporary directory, and to execute them. jfd creates temporary submission, control, and query scripts in the temporary directory.

jfd creates a ppm subdirectory. All temporary job submission, control, and query scripts are stored under /tmp/ppm.

6. eem.batch queries the remote host on the Other Batch System for job status updates.
7. jfd listens on the port number JS\_PORT + 1 to receive status updates from eem.batch.
8. The user can view the status of the Other Batch job through Flow Manager, the Process Manager command-line, or IBM Spectrum LSF Application Center.

---

## About Other Batch Jobs

### Limitations

---

Other Batch jobs are only supported when the Process Manager Server is on UNIX/Linux.

### Job Definition

---

When Process Manager is configured to communicate with the Other Batch System, you will see the Other

Batch Job work item enabled in Flow Editor:





In the Job Definition, you specify the command to run, other job submission options, and the user account under which the job is to run in the Other Batch System.

## Job monitoring and control

---

You monitor an Other Batch job in the same way as you monitor any other job. In the job's runtime attributes, you can view the same job information that you can view for any LSF® job.

The following actions can be taken on an Other Batch job:

- **Kill:** You can kill Other Batch jobs in the same way as you kill any other job. The job may also be killed as a result of the flow being killed. Note that the state of the job will not change until the job is actually killed in the Other Batch System. When you kill a flow that contains Other Batch jobs, the state of the flow immediately changes, but the state of the Other Batch job does not change until the job is actually killed in the Other Batch System.
- **Suspend:** If you suspend a flow that contains Other Batch jobs, Other Batch jobs will also be suspended. When you suspend a flow that contains Other Batch jobs, the state of the flow immediately changes, but the state of the Other Batch job does not change until the job is actually suspended in the Other Batch System.
- **Resume:** If you resume a flow that contains Other Batch jobs, Other Batch System jobs will also be resumed. When you resume a flow that contains Other Batch jobs, the state of the flow immediately changes, but the state of the Other Batch job does not change until the job is actually resumed in the Other Batch System.

It is possible that jobs submitted to different batch systems have the same job ID. This is not a problem because jfd distinguishes jobs by job name.

## Dependencies

---

The dependency types coming out of an Other Batch job are:

- **Completes successfully:** The job completes with exit code 0. If a job state is Done, it is considered as "Completes Successfully", otherwise it is considered as "Fails".
- **Fails:** The job fails.
- **Ends with any exit code:** The job exits with any exit code, including 0.
- **Ends with exit code:** The job exits with a particular exit code pattern. For example, not-equal-to, equal-to.

## Failover and Other Batch jobs

---

Should jfd terminate abnormally, when it restarts it can recover running and finished Other Batch jobs and determine their status and resource usage.

If jfd is shut down normally, or if jfd terminates abnormally:

- If the Other Batch job is still running when jfd is restarted, jfd can recover the job as long as query.sh returns the correct job status from the Other Batch System.
- If the Other Batch job has finished(either normally or abnormally) when jfd is restarted, jfd can recover the exit status and resource usage of the job as long as query.sh returns the correct job status from the Other Batch System.

If eem.batch terminates abnormally, it will be restarted by jfd.

## Configuring Process Manager to work with Other Batch Systems

- **Step Summary and Configuration Checklist**

Before you begin, take a few moments to gather the information you will need for configuration.

- **1. Configure password-less SSH connections**

In order for Process Manager to operate on jobs in the Other Batch System, you need to set up password-less SSH connections for administrator and regular user accounts.

- 2. Enable another Batch System

- 3. Configure connection to the Other Batch System

- 4. Customize job submission, control, and query scripts

There are three files to map job submission, control, and query commands and options to the Other Batch System: `submit.sh`, `control.sh`, and `query.sh`. jfd modifies these scripts at runtime to set environment variables, then copies them over to the temporary directory on the Other Batch System host, where they are executed.

- **5. Restart the Process Manager Server**

Restart the Process Manager Server to make your changes take effect.

- **6.Test the Other Batch System**

## Step Summary and Configuration Checklist

Before you begin, take a few moments to gather the information you will need for configuration.

| Step Summary   | Category      | What you need                         | Description  |
|--|---------------|---------------------------------------|--|
| Prerequisites  | Prerequisites | Port number JS_PORT + 1 must be free  | For each configured batch system, an eem.batch binary is started by jfd.<br><br>jfd listens on the port number JS_PORT + 1 to receive status updates from eem.batch. The parameter JS_PORT is defined in js.conf.    |
| <a href="#">1. Configure password-less SSH connections</a> | User accounts | Process Manager administrator account | User name of the Process Manager administrator account.<br>You need this information to set up password-less SSH between the Process Manager administrator account and the Other Batch System administrator account. |
|  |               |                                       |  |

| Step Summary  | Category                        | What you need                            | Description  |
|---|---------------------------------|--|--|
|   | User accounts                   | Other Batch System administrator account | <p>User name of the Other Batch System administrator account.</p> <p>You need this information to set up password-less SSH between the Process Manager administrator account and the Other Batch System administrator account.</p> <p>This account is used to execute all query commands on the Other Batch System.</p> <p>This account must have read/write/execute access to the temporary directory on the host of the Other Batch System specified with JS_EE_HOST in your settings.conf file.</p>   |
|   | User accounts                   | Other Batch System user accounts         | <p>User names of the accounts that will be used to submit jobs to the Other Batch System. Jobs will also run under this user account.</p> <p>You need this information to set up password-less SSH between the Process Manager regular user accounts and the Other Batch System user accounts.</p> <p>These accounts must have read/write/execute access to the temporary directory on the host of the Other Batch System specified with JS_EE_HOST in your settings.conf file.</p> <p>When a user creates an Other Batch job in Flow Editor, the user is required to specify the user account under which to run the job. He can specify either to run under his own user account on the Other Batch System, or under a different user account.</p> |
| <a href="#">2. Enable another Batch System</a>                    | Configuration files and scripts | Example configuration files and scripts  | <p>There are ready-to-use configuration files and submission scripts for IBM® LoadLeveler® and Open Grid Scheduler/Grid Engine in <b>\$JS_TOP/\$JS_VERSION/examples/conf/other_batch</b>.</p> <p>You will need to copy these files to a subdirectory of \$JS_ENVDIR/other_batch.</p>   |
| <a href="#">3. Configure connection to the Other Batch System</a> | Host name                       | Host name of the Other Batch System      | <p>Host name or IP address of a host on the Other Batch System from which you can submit/control/query jobs. All job commands will be run from this host on the Other Batch System.</p> <p>You will need this information to specify JS_EE_HOST in your settings.conf file.</p>  |
|   |                                 |  |  |

| Step Summary  | Category                     | What you need   | Description  |
|---|------------------------------|---|--|
|   | Temporary directory location | Temporary directory on the host of the Other Batch System | Temporary directory location on the other host on which you will be able to store the temporary submission, control, and query scripts.<br><br>You will need this information to specify JS_EE_TMP_DIR in your settings.conf file. |
|   | User accounts                | Other Batch System administrator account                  | User name of the Other Batch System administrator account.<br><br>You need this information to specify JS_EE_USER in your settings.conf file.  |
| <a href="#">4. Customize job submission, control, and query scripts</a> |                              |   |  |
| <a href="#">5. Restart the Process Manager Server</a>                   |                              |   |  |
| <a href="#">6. Test the Other Batch System</a>                          |                              |   |  |

## 1. Configure password-less SSH connections

In order for Process Manager to operate on jobs in the Other Batch System, you need to set up password-less SSH connections for administrator and regular user accounts.

### Set up password-less SSH for administrator accounts

#### Procedure

1. Configure SSH Public and Private keys between the Process Manager administrator on the Process Manager server and the administrator of the Other Batch System on the remote host so that no passwords or passphrases are required.  
This is required because the administrator account of the Other Batch System is used to run job query commands on the Other Batch System.
2. Ensure that the Other Batch System administrator account has read/write/execute permission on the temporary directory on the Other Batch System host.

### Set up password-less SSH for regular user accounts

#### Procedure

1. Configure SSH Public and Private keys between Process Manager user accounts on the Process Manager server, and the user accounts on the Other Batch System so that no passwords or passphrases are required.  
This is required because when a job is submitted, it can run under the user account which submitted the job or as a different user account indicated in the Job Definition.

2. Ensure that the Other Batch System user accounts have read/write/execute permission on the temporary directory on the Other Batch System host.

---

## 2. Enable another Batch System

### About this task

---

In order to be able to submit jobs to a batch system other than LSF®, you need to enable it in Process Manager so it can be selected by the user in the Job Definition. All valid batch systems are listed in Flow Editor, in the Job Definition, under the General tab, Select the Batch System, Name .

There are ready-to-use configuration files and submission scripts for IBM® LoadLeveler® and Open Grid Scheduler/Grid Engine in `$JS_TOP/$JS_VERSION/examples/conf/other_batch`.

### Procedure

---

1. On the Process Manager server host, create a subdirectory under `$JS_ENVDIR/other_batch`.  
Important: The name of the subdirectory you create becomes the identifier of the batch system and will be displayed in the Job Definition and as the job type when monitoring jobs in flows.  
For example, to configure a batch system called **mybatch**, create the directory `$LSF_ENVDIR/other_batch/mybatch`.
2. If you are using IBM LoadLeveler or Open Grid Scheduler/Grid Engine, copy the example files provided for your batch system from `$JS_TOP/$JS_VERSION/examples/conf/other_batch` to your directory. If you are using a batch system other than IBM LoadLeveler or Open Grid Scheduler/Engine, copy the example files from one of the directories provided for IBM LoadLeveler or Open Grid Scheduler/Grid Engine as a starting point.  
You will need `control.sh`, `submit.sh`, `query.sh`, `settings.conf`, and `submit.conf`.

---

## 3. Configure connection to the Other Batch System

### Procedure

---

1. With a text editor, open the `settings.conf` file in the subdirectory you created under `$JS_ENVDIR/other_batch`.
2. Specify `JS_EE_HOST`, `JS_EE_USER`, and `JS_EE_TMP_DIR`.  
These are required parameters.
3. Save the file.

---

## 4. Customize job submission, control, and query scripts

There are three files to map job submission, control, and query commands and options to the Other Batch System: `submit.sh`, `control.sh`, and `query.sh`. `jfd` modifies these scripts at runtime to set environment

variables, then copies them over to the temporary directory on the Other Batch System host, where they are executed.

If you are using IBM® LoadLeveler® or Open Grid Scheduler/Grid Engine, there are ready-to-go scripts already provided in the `$JS_TOP/$JS_VERSION/examples/conf/other_batch` directory.

If you are using a different batch system than IBM LoadLeveler or Open Grid Scheduler/Grid Engine, you will need to modify these files and map job submission, control, and query commands for your specific batch system.

The scripts must be bash shell scripts.

## submit.sh

| Description   | Input   | Output  | Exit Code  |
|---|---|---|--|
| <p>Job submission command and options for the specific batch system. The script is executed on the Other Batch System host to submit jobs.</p> <p>If you require additional submission options from the ones that are provided by default, you can list additional options in the configuration file <code>submit.conf</code>.</p> <p>Runs as the user account defined in the Job Definition: the user account who submitted the job, or the user account specified in the Job Definition.</p> <p>Each submission option in the script has an associated environment variable that is passed to the submission script.</p> <p>Default job submission options exposed in the Job Definition are:</p> <ul style="list-style-type: none"><li>• Command to run the job (required)</li><li>• Job name (required)</li><li>• Environment variables to be set for the job(optional)</li></ul> <p>Note that the user is required to provide values for required options.</p> | <p>Environment variables</p> <p><code>\$JS_EE_SUBMISSION_JOB_COMMAND (required)</code></p> <p><code>\$JS_EE_SUBMISSION_JOB_NAME (required)</code></p> <p><code>\$JS_EE_SUBMISSION_JOB_ENV_VARS (optional)</code></p> <p>Additional custom environment variables specified in <code>submit.conf</code>, if used.</p> | <p>On Success: Job ID</p> <p>On Error: Error messages if applicable</p> | <p>Zero on success.</p> <p>Non-zero on failure. Error messages must be printed to standard error</p> |

## submit.conf

Customized job submission options that are displayed in Flow Editor in the Job Definition.

The options listed in this file are displayed in addition to the default job submission options.

Note that after defining the environment variables in `submit.conf`, you will need to modify `submit.sh` to map the environment variables to actual options in the `submit.sh` file. You will need to add the new options to `submit.sh`.

Only labels and text fields (text strings) are supported as customized submission options. No type checking is enforced for the input.

Each line must contain three fields: Label, Environment Variable, and Required(1 indicates required, 0 optional). Lines that start with # are ignored.

When a user is creating a Job Definition, Flow Editor checks the value of the required fields. If any required field is empty, the Job Definition is not complete and the user will not be able to submit the flow. During job submission, jfd checks the value of the required fields. If a required field value is empty, job submission fails.

Example submit.conf file:

| #Label                 | #Environment Variable       | #Required |
|------------------------|-----------------------------|-----------|
| "Submit to queue"      | JS_EE_SUBMISSION_QUEUE_NAME | 0         |
| "Run on host"          | JS_EE_SUBMISSION_HOST_NAME  | 0         |
| "Resource requirement" | JS_EE_SUBMISSION_RES_REQ    | 0         |

## control.sh

| Description  | Input   | Output                                 | Exit Code   |
|--|---|--|---|
| Job control options for the specific batch system: kill, suspend, and resume.  | Command-line arguments:                                   | On Success: Not required               | Zero on success.  |
| Runs as the user account defined in the Job Definition: the user account who submitted the job, or the user account specified in the Job Definition.   | \$1: Job control actions: KILL, SUSPEND, RESUME(required) | On Error: Error messages if applicable | Non-zero on failure. Error messages must be printed to standard error |
| The script takes two arguments: job control action and job ID:<br><br>1. Job control action. Valid actions: KILL, SUSPEND, RESUME.<br>2. ID of the job to control. A valid job ID is a positive integer. | \$2: ID of the job to control(required)                   |  |   |
| Note: The job control script does not support controlling multiple jobs at the same time.  |   |  |   |

## query.sh

| File | Description | Input | Output | Exit Code |
|------|-------------|-------|--------|-----------|
|------|-------------|-------|--------|-----------|

| File     | Description  | Input  | Output  | Exit Code  |
|----------|--|--|---|--|
| query.sh | <p>Job query script executed on the Other Batch System to retrieve the status of jobs.</p> <p>Runs as the Other Batch System administrator account to query all jobs submitted to the Other Batch System.</p> <p>The query.sh script is not required to print out the status of all jobs specified on the command-line argument. If a requested job status is not printed out, the status of that job is assumed to be unchanged by jfd.</p> <p>For example, if query.sh has an input of job IDs 1, 2, and 3, and it only prints the status records for job 1 and job 2, jfd assumes that the status of job 3 has not changed.</p> | <p>Command-line arguments:</p> <p>\$1: Space-separated job IDs(required)</p> | <p>On Success: A list of job status records</p> <p>On Error: Error messages if applicable</p> | <p>Zero on success.</p> <p>Non-zero on failure. Error messages must be printed to standard error</p> |

Job status records must be printed in the following format:

- Each job status record must start with **BEGIN**, and end with **END**, with name-value pairs in between.
- Only JOB\_ID and JOB\_STATE are required. Other names, if no values exist, do not need to be listed.
- jfd parses the output of query.sh record by record. If any name-value pair is not valid, it will be ignored. If any record does not contain the required name-value pairs, it will be ignored.

Format:

BEGIN

JOB\_ID=

JOB\_STATE=

JOB\_EXIT\_STATUS=

CPU\_TIME=

DETAIL=

EXEC\_HOST=

END

...

Valid names are:

- JOB\_ID: Required. A valid job ID is a positive integer.



- **JOB\_STATE:** Required. A valid job state is one of the following: PENDING, RUNNING, SUSPENDED, DONE and EXIT.
- **JOB\_EXIT\_STATUS:** A valid job exit status is an integer from 0 to 255. Required when JOB\_STATE is DONE or EXIT.
- **CPU\_TIME:** A valid CPU time is a non-negative floating-point number. Only read and checked by jfd when JOB\_STATE is DONE or EXIT.
- **DETAIL:** Optional. String with additional information to display in the job runtime attributes. You can use the DETAIL string to provide additional information about a job status that is specific to a batch system. For example, a job in the Other Batch System might be in a PENDING state because it is on hold. In this case, the DETAIL string can say "On Hold".
- **EXEC\_HOST:** Host on which the job is running.

Example output of a query:

```
username@tt-jj-194: /tmp/query.sh 174 172 171 176
BEGIN
JOB_ID=171
JOB_STATE=DONE
JOB_EXIT_STATUS=0
CPU_TIME=0.073
DETAIL=
EXEC_HOST=hostA
END
BEGIN
JOB_ID=172
JOB_STATE=EXIT
JOB_EXIT_STATUS=137
CPU_TIME=0.088
DETAIL=100 : after job
EXEC_HOST=hostB
END
BEGIN
JOB_ID=174
JOB_STATE=SUSPENDED
END
BEGIN
JOB_ID=176
JOB_STATE=PENDING
END
```

---

## 5. Restart the Process Manager Server

Restart the Process Manager Server to make your changes take effect.

### Procedure

---

1. Log on to the Process Manager Server as root.
2. Set your environment.
  - On csh or tcsh:

```
source JS_TOP/conf/cshrc.js
```

- On sh, ksh or bash:

```
. JS_TOP/conf/profile.js
```

3. Run jadmin to restart the Process Manager Server:

```
jadmin stop
```

```
jadmin start
```

---

## 6.Test the Other Batch System

### Procedure

---

1. In Flow Editor, create a new flow definition

You should be able to see the Other Batch System icon enabled.



2. Select the Other Batch System Job icon and drop it into your flow.

3. Right-click and choose Open Job Definition.

In the General tab, you should be able to see your configured batch system in the list under Select the batch system, Name.

4. Specify the required fields and save the job definition.

5. Submit your flow definition.

6. In Flow Manager, trigger a flow from the job definition.

Monitor the job and see it run on the Other Batch System.

---

## Configure Data Transfer to and from the Other Batch System

Before running a job on the Other Batch System, you may need to transfer data from the local system to the Other Batch System as input to the job.

### About this task

---

When the job completes, you may also need to transfer the standard output/error from the Other Batch System back to the local system.

To do this, you can configure submission options for input, output and error files, and create one local job in your flow definition that precedes the Other Batch job and another local job that follows the Other Batch job to handle input and output files.

### Procedure

---

1. In your submit.conf file, specify standard input, output, and error as customized options to the Job Definition.

For example:

| #Label        | #Environment Variable   | #Required |
|---------------|-------------------------|-----------|
| "Input File"  | JS_EE_SUBMISSION_STDIN  | 0         |
| "Output File" | JS_EE_SUBMISSION_STDOUT | 0         |
| "Error File"  | JS_EE_SUBMISSION_STDERR | 0         |

2. Edit your submit.sh script file so that jobs are submitted with the appropriate input, output, and error file options for your specific batch system.

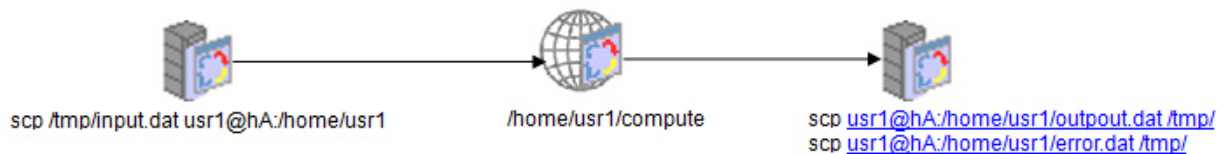
For example, -i JS\_EE\_SUBMISSION\_STDIN, -o \$JS\_EE\_SUBMISSION\_STDOUT, -e \$JS\_EE\_SUBMISSION\_ERR.

3. In Flow Editor, in your flow definition, define a local job immediately before and immediately after your Other Batch job.

The local job that directly precedes your Other Batch job copies the local data to the Other Batch System.

The local job that directly follows your Other Batch job copies the output and error data from the Other Batch System to the local system.

For example:



4. In the Job Definition of the Other Batch job, ensure the correct path to the Other Batch System is specified.

For example:

- Input file: /home/usr1/input.dat
- Output file: /home/usr1/output.dat
- Error file: /home/usr1/error.dat

## Troubleshooting Other Batch System Jobs

| Error type               | Where it is logged  |
|--------------------------|---|
| SSH connection times out | <p>The error is logged in history for the job under job submission and control actions, with a message that connection has timed out.</p> <p>You can access the history through Flow Manager, the <b>jhist</b> command, or IBM Spectrum LSF Application Center.</p> |

| Error type   | Where it is logged   |
|--|--|
| SSH connection successful, but submission and control actions fail on the Other Batch System | <p>The error is logged in history for the job.</p> <p>For job submission, the error message from the standard error output of the submit.sh script is logged in history.</p> <p>For job control, the error message from the standard error output of the query.sh script is logged in history.</p> <p>You can access the history with the <b>jhlist</b> command.</p> |
| Job status query fails from jfd to eem.batch.  | If the job status query fails for any reason, LOG_DEBUG level error messages are logged in the \$JS_LOGDIR/log/jfd.log. <i>host_name</i> file.   |
| Error and debug messages for eem.batch   | Error and debug messages for the eem.batch of each configured batch system are logged in \$JS_LOGDIR/log/eem.otherbatch.log. <i>batch_system_name</i> . The batch system name is the directory name for that batch system under \$JS_ENVDIR/other_batch.   |

---

## Mainframe support

Process Manager with IBM® z/OS® mainframe support allows you to dispatch jobs to a mainframe and monitor their progress using FTP (file transfer protocol) technology on Microsoft® Windows® or UNIX.

z/OS® is an operating system for IBM's zSeries mainframes.

For more information about z/OS, see IBM's z/OS website: <http://www-03.ibm.com/servers/eserver/zseries/zos/>.

## How does it work?

The Process Manager daemon (the jfd) supports mainframe by submitting an LSF® proxy job which controls the FTP to the mainframe host. The LSF proxy job (through FTP) submits, monitors, and retrieves the output of the mainframe job. This means that mainframe jobs specify both mainframe and LSF details.

## Requirements

- A valid z/OS mainframe user ID

## Limitations

- z/OS does not support suspending or resuming jobs
- Job arrays for mainframe jobs are not supported
- On Windows, if you want to be able to kill a mainframe job, you must submit the job to a queue set up specifically for that purpose.
- [Configure for Mainframe](#)

---

## Configure for Mainframe

## About this task

---

To use the mainframe support, you must:

## Procedure

---

1. Copy the template file z/OS\_Template.xml from JS\_TOP/10.2/examples to JS\_TOP/work/templates.
2. Edit zos.conf with your customized settings. The zos.conf file contains all the information you need to configure your settings for the FTP environment you are using.

## Results

---

The status of mainframe jobs is displayed in Flow Manager.

## Killing a job (Windows only)

---

For a user to be able to kill a job in a Windows environment, the Administrator must create a queue. For jobs to be eligible to be killed, they must be submitted by the user to that queue.

In lsb.queues in your z/OS-specific queue section, add a job control and the path to the script that kills the job.

For example,

```
Begin Queue
QUEUE_NAME= zos_queue
DESCRIPTION= Bkill for zos jobs.
JOB_CONTROLS= TERMINATE[C:\ppm\10.2\etc\zos -k]
End Queue
```

---

## Daemons

- jfd
- fod
- [jfd](#)  
Process Manager Server daemon.
- [fod](#)  
Process Manager Server failover daemon.

---

## jfd

Process Manager Server daemon.

---

## Synopsis

jfd [-2 | -3 | -4]

jfd [-V]

## Description

---

jfd is responsible for managing flow definitions and flows. When a flow definition is committed 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 management host 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 Technical Support.

-4

Specifies to run jfd as daemonized, and log debug information to the jfd.log.*hostname* log file. This option may be used for debugging purposes, and allows you to run **jfd** as a user other than `root`. Use only under the direction of Technical Support.

-V

Prints the Process Manager release version to stderr and exits.

## See also

---

**fod, jadmin**

---

## fod

Process Manager Server failover daemon.

## Synopsis

---

fod

---

## Commands

Process Manager includes a command line interface you can use to issue commands to Process Manager. You can use commands to submit flow definitions to Process Manager, trigger flows to run, monitor and control running flows, and obtain history information about many Process Manager work items.

Process Manager provides commands for various purposes: creating and editing calendars, manipulating flow definitions, monitoring and controlling active flows, and obtaining history about various work items.

You cannot use commands to create a flow definition.

## Calendar commands

---

You can use the following commands to work with Process Manager calendars:

- **caleditor**—to start the Calendar Editor graphical user interface
- **jcadd**—to create a calendar
- **jcals**—to display a list of calendars
- **jcdel**—to delete a calendar
- **jcmmod**—to edit a calendar

## Flow definition commands

---

You can use the following commands to work with flow definitions:

- **floweditor**—to start the Flow Editor graphical user interface
- **jrun**—to submit and run a flow immediately, without storing the flow definition in Process Manager
- **jsub**—alias for jcommit
- **jtrigger**—manually submits a previously committed flow definition
- **jhold**—to place a flow definition on hold, preventing automatic triggering of the flow
- **jrelease**—to release a flow definition from hold, enabling automatic triggering of the flow
- **jdefs**—to display information about flow definitions
- **jcommit**—to commit a flow definition
- **jsubmit**—alias for jtrigger
- **jexport**—exports flow definitions to a file
- **jremove**—to remove a flow definition from Process Manager
- **jsetversion**—sets the default version of a flow
- **jpublish**—to publish target flows for use by dynamic flows and flow arrays
- **junpublish**—to unpublish target flows and remove them from the list for use by dynamic flows and flow arrays

## Flow monitor and control commands

---

You can use the following commands to monitor and control flows that are in the process of running or have recently completed:

- **flowmanager**—to start the Flow Manager graphical user interface
- **jalarms**—to list open alarms
- **jcomplete**—to complete a manual job
- **jflows**—to display information about a flow
- **jjob**—to kill or run a job, or to mark a job complete
- **jkill**—to kill a flow
- **jmanuals**—to list all manual jobs waiting for completion
- **jrerun**—to rerun an exited flow
- **jresume**—to resume a suspended flow
- **jsetvars**—to change the value of a local or global variable while a flow is running
- **jstop**—to suspend a flow

## Other commands

---

- **jid**—to verify the connection between the Process Manager Client and the Process Manager Server
- **jadmin**—to control the Process Manager daemon on Unix
- **jhist**—to view the historic information about server, flow definitions, flows, and jobs.
- **jreconfigalarm**—to reload the alarm definitions.
- **jreconfigadmin**— to dynamically reconfigure and update the list of administrators.
- **[caleditor](#)**  
starts the Calendar Editor Process Manager Client.
- **[floweditor](#)**  
starts the Flow Editor Process Manager Client.
- **[flowmanager](#)**  
starts the Flow Manager Process Manager Client.
- **[jadmin](#)**  
controls the Process Manager daemon jfd on UNIX.
- **[jalarms](#)**  
lists the open alarms in Process Manager.
- **[jcadd](#)**  
creates a calendar and adds it to the set of Process Manager calendars for the user.
- **[jcal](#)**  
displays the list of calendars in Process Manager. The calendars are listed by owning user ID.
- **[jcdel](#)**  
deletes an existing calendar.
- **[jcmmod](#)**  
edits an existing calendar. Using this command, you can change the calendar expression and the description of the calendar.
- **[jcommit](#)**  
commits a flow definition to Process Manager.
- **[jcomplete](#)**  
acknowledges that a manual job is complete and specifies to continue processing the flow.
- **[jdefs](#)**  
displays information about the flow definitions stored in Process Manager for the specified user.
- **[jexport](#)**  
exports flow definitions to a file
- **[jflows](#)**  
displays information about the flows in Process Manager for the specified user. The information listed includes the current state and version of the flow.
- **[jhist](#)**  
displays historical information about Process Manager Server, calendars, flow definitions, flows, and jobs.
- **[jhold](#)**
- **[jid](#)**  
displays the host name, version number and copyright date of the current Process Manager Server.
- **[jjob](#)**  
controls a job in a running flow.
- **[jkill](#)**  
kills a flow.
- **[jlicenseupdate](#)**  
Updates the SAS license, Process Manager entitlement file, and LSF entitlement file
- **[jmanuals](#)**  
displays all manual jobs that have not yet been completed.



- [\*\*jpublish\*\*](#)  
publishes a flow to Process Manager and to other users.
- [\*\*jreconfigadmin\*\*](#)  
dynamically reconfigures and updates the list of administrators.
- [\*\*jreconfigalarm\*\*](#)  
reloads the alarm definitions.
- [\*\*jrelease\*\*](#)
- [\*\*jremove\*\*](#)
- [\*\*jrerun\*\*](#)  
reruns an exited, done, or running flow.
- [\*\*jresume\*\*](#)  
resumes a suspended flow.
- [\*\*jrun\*\*](#)  
triggers a flow definition from a file and runs the flow immediately without storing the flow definition in Process Manager.
- [\*\*jsetvars\*\*](#)  
sets values for variables during the runtime of a flow.
- [\*\*jsetversion\*\*](#)  
sets the default version of a flow.
- [\*\*jsinstall\*\*](#)  
runs **jsinstall**, the Process Manager installation and configuration script
- [\*\*jstop\*\*](#)  
suspends a running flow.
- [\*\*jsub\*\*](#)  
submits a flow definition to Process Manager. This command has been replaced with **jcommit** in version 10.1 but can still be used for compatibility. **jsub** and **jcommit** are aliases.
- [\*\*jsubmit\*\*](#)
- [\*\*jtrigger\*\*](#)
- [\*\*junpublish\*\*](#)  
unpublishes a target flow from Process Manager.
- [\*\*licenseinfo\*\*](#)  
displays information about SAS licenses, LSF licenses, or LSF entitlement files
- [\*\*ppmsetvar\*\*](#)  
sets or removes flow user variables, subflow user variables, and global user variables from a work item in a flow or subflow

---

## caleditor

starts the Calendar Editor Process Manager Client.

## 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 Process Manager Client.

### 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.

Note:

Flow Editor may not be installed if you purchased the Platform Suite for SAS. For more information, contact your sales representative.

### 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 Process Manager Client.

### 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 [-s] start
```

```
jadmin stop
```

```
jadmin [-h|-V]
```

## Description

---

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

## Options

---

`start`

Starts the Process Manager daemon on UNIX. You must be `root` to use this option.

`-s start`

Starts the Process Manager daemon on UNIX in single-user mode. You must be the primary Process Manager administrator 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 -s start
```

Starts the Process Manager daemon in single-user mode.

```
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.

`-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 as 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 9 17:06 2002

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

Example: Display all opened alarms for the last seven days

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

---

## jcadd

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

## Synopsis

---

```
jcadd [-s | -u user_name][-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. Does not support multi-line.

**-u** *user\_name*

You must be the Process Manager administrator or a Group administrator to use this option.

Sets the user account that owns the calendar. The owner of a calendar can modify and delete a calendar.

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

**-s**

Specifies that you are creating a system calendar. You must be a Process Manager administrator to create system calendars.

**-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.

**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 (\_) and a dash (-) 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  
RAN  
GE (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 **jcmmod** 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.

## 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.

## Examples

---

```
jcals -u all
```

Displays all calendars in Process Manager.

---

## jcdel

deletes an existing calendar.

## Synopsis

---

```
jcdel [-f][-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.

If you delete a calendar that is currently in use by a flow definition or flow, or another calendar, the deleted calendar will continue to be available to these existing instances, but will no longer be available to new instances.

## Options

---

-f

Specifies to force the deletion of the calendar.

-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" Runday2001
```

Deletes the calendar Runday2001 owned by the user barneyt.

## See also

---

**jcadd, jcal**

---

## jcmod

edits an existing calendar. Using this command, you can change the calendar expression and the description of the 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.

If you modify a calendar that is in use by a flow definition or flow, or another calendar, your changes will only take effect on any new instances; current instances will continue to use the previous calendar definition.

## Options

---

**-d** *description*

Specifies a description for the calendar. Specify a meaningful description for the calendar that summarizes the expression. Does not support multi-line.

**-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.

*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.

## 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

---

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

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

---

## jcommit



commits a flow definition to Process Manager.

## Synopsis

---

```
jcommit [-H] [-r [-v version]] [-d] [-m "ver_comment"] [[-T time_event] ...] [[-F "file_event"] ...] [[-p "proxy_event"] ...] [-C combination_type] flow_file_name
```

```
jcommit [-H] [-r [-v version]] [-d] [-m "ver_comment"] [-k] flow_file_name  
jcommit -h|-V
```

## Description

---

## Options

---

-H

Commits the flow definition on hold, with event triggers disabled. No automatic events can submit this definition until it has been explicitly released and event triggers enabled. 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 submitted manually, such as for testing purposes.

-r [-v *version*]

Replace. If a flow definition with the same name already exists in Process Manager, replace it with the definition being committed. Use -v to assign a version number to the flow definition being committed. If you do not assign a version number, a version number is automatically assigned incremental to the last version number.

If you do not specify -r and the flow definition already exists, commit 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 committed.

-m "*ver\_comment*"

Commit the flow with version comments. A flow version number is returned after each successful commit.

-T *time\_event*

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

```
[cal_name[@username]:]hour:minute[%duration]][#occurences][+time_zone_id]
```

Note: You can find a list of valid time zone IDs in  
JS\_HOME/JS\_VERSION/resources/timezones.properties.

*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 commit user 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"*

Overwrites file events specified in the flow definition. Specifies to automatically submit 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 Saturdays.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)*

Submit 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)*

Submit a flow if the specified file exists in the specified location, and continue to submit 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)*

Submit a flow if the specified file does not exist in the specified location, and continue to submit 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*

Submit 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*"

Overwrites proxy events specified in the flow definition. Specifies to automatically submit 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])*

Submit 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 ==.

If you are specifying exit codes, you can specify multiple exit codes when using the operators != and ==. Separate the exit codes with spaces, and specify a number from 0 to 255.

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)"

- Example: if payjob ends with any of the following exit codes: 5, 10, 12, or 14:

-p "job(exit(jdoe:myflow:testflow:payjob)==5 10 12 14)"

- Example: if payjob does NOT end with any of the following exit codes: 7, 9, 11:

```
-p "job(exit(jdoe:myflow:testflow:payjob)!=7 9 11)"
```

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

Submit 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])`

Submit 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.

**-C combination\_type**

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

**AND**

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

**OR**

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

-k

Use the triggering events defined in the flow definition. If you do not specify this option, you can overwrite triggering events defined in the flow definition with the options -T, -F, -p, -C.

*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
```

Commit the flow definition in myflow.xml, to be submitted 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
```

Commit the flow definition in backup.xml, to be submitted 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.

---

## jcomplete

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

## Synopsis

---

```
jcomplete [-d description] [-u user_name] [-e exit_code]-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.

**-e** *exit\_code*

Specifies the exit code with which to complete the manual job.

The exit code you specify determines the state of the manual job. Exit codes can be any number from 0 to 255.

If you did not define custom success exit codes in the Manual Job Definition, an exit code of 0 indicates the manual job was successful and the state is set to Done. Any other exit code indicates the manual job failed and its state is set to Exit.

If you defined custom success exit codes in the Manual Job Definition, an exit code of 0 and any of the numbers you specified in the Non-zero success exit codes field indicates the manual job was successful and the state is set to Done. Any other exit code indicates the manual job failed and its state is set to Exit.

**-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] [-v][-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
- flow version history and details

## 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 submit the flow
- any exit conditions specified in the flow definition
- the default version and the latest version of the flow

**-v**

Displays the version history of the flow.

**-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.

## Examples

---

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

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

---

## jexport

exports flow definitions to a file

## Synopsis

---

```
jexport [-o ] [-u user_name | -u all] [-s status] [flow_name ...]
```

```
jexport [-o ] [-u user_name] -v version flow_name | -v all flow_name
```

```
jexport -h | -V
```

## Description

---

By default, exports to the current directory the default version of all flow definitions owned by the user who invoked the command. Each flow definition version is saved with the name *owner\_flowname\_version.xml*

## Options

---

#### -o

Writes the name of the user who owns the flow definition on the Process Manager server into the exported file. If an owner is specified in the flow definition, that user name is written as the owner. If an owner is not specified in the flow definition, the user who submitted the flow definition to Process Manager is the owner and the user name that is written to the file.



**-u *user\_name* | -u all**

Exports the default version of all flow definitions owned by the specified user. If the keyword all is used, exports the default version of all flow definitions owned by all users.

**-s *status***

Exports the default version of all flow definitions that have the specified state Valid states are:

- ONHOLD: Flow definitions that are not eligible to be automatically triggered. These definitions have disabled event triggers.
- RELEASE: Flow definitions that are not On Hold. These definitions have enabled event triggers.

***flow\_name* ...**

Name of the flow definition to export. To specify a list of flow definitions, separate the flow definition names with a space.

**-v *version* | -v all**

Specifies which version of the flow definition to export. Use the keyword all to export all versions of the specified flow definition.

**-h**

Prints the command usage to stderr and exits.

**-V**

Prints the Process Manager release version to stderr and exits.

## Example: Export all versions of the flow definition Sample for the user who invoked the command

---

```
jexport -o -v all Sample
```

## Example: Export the default version of all flow definitions for all users

---

```
jexport -o -u all
```

## Example: Export version 1.1 of the flow definition Sample for user user1

---

```
jexport -o -u user1 -v 1.1 Sample
```

---

## jflows

displays information about the flows in Process Manager for the specified user. The information listed includes the current state and version 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, subflows, and flow arrays in the flow, and displays the currently-used version 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.

## 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,...] [-o operator_user_name|-o all] [-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.

If your Process Manager client and server are on separate hosts, the number of history records retrieved is limited to 1500 records by default. If the limit is reached, only the first (oldest) 1500 are retrieved. This limit is configurable with the parameter **JS\_HISTORY\_LIMIT** in js.conf.

Process Manager searches the directory JS\_HOME/work/history/ for history logs.

If JS\_HISTORY\_ARCHIVE\_DIR is defined in js.conf, you can view archived history logs by running **jhist** from the Process Manager server host: Process Manager searches both JS\_HOME/work/history/ and

JS\_HISTORY\_ARCHIVE\_DIR for history logs. When **jhist** is run from a host that is not the Process Manager server host, Process Manager only searches JS\_HOME/work/history/ for history logs.

## 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 the 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.

### **-o** *operator\_user\_name* | **-o all**

Only applies to the following categories: flowdef, flow, calendar.

Displays only information about flows, flow definitions, or calendars that the specified user has operated on. If you do not specify a user name, user name defaults to the user who invoked this command. If you specify **-o all**, displays information about all flow definitions, flows, or calendars that all users have operated on.

### **-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.

### **-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 as 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)

## 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,.
```

Display information for all flows associated with the flow definition myflow and that userA can operate on

```
jhist -C flowdef,flow -f myflow -o userA
```

---

## jhold

places a previously committed flow definition on hold. No automatic events can submit this definition until it has been explicitly released. Use this command when you want to temporarily interrupt automatic submission of a flow. When a flow is on hold, it can still be submitted 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 committed flow definition on hold. This prevents it from being submitted 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|-p|-g|-l [-a] "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.

-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.

-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.

-p

Specifies to put the job on hold. Only jobs in the Waiting state can be put on hold. You can put on hold LSF jobs, job submission scripts, local jobs, and job arrays.

If the selected job is in a flow array, by default the hold applies to the job in the element the job is in. You can, alternatively, apply the hold to jobs in all elements in the flow array.

When you put a job in the flow on hold, the flow pauses at that specific job. Only the branch of the flow that contains the job that is On Hold pauses. Other branches of the flow continue to run. The status of the flow is not affected.

When desired, you can then release the job that you have put on hold.

-g

Specifies to release a job that has been put on hold. You can release LSF® jobs, job submission scripts, local jobs, and job arrays that have been put on hold.

When you release a job that has been put on hold, the flow instance continues to run and the job receives the status Waiting.

-l

Specifies to view the detailed history of local and input variables that the job uses. This does not show global variables.

-a

Specifies that the job to control is a job array.

*flow\_name:subflow\_name: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**

Note:

When specifying the job name for a flow array, you must enclose the name in quotation marks ("). This is because the Linux® command line does not process parentheses characters ( ( or ) ) properly unless

you use quotation marks.

For example:

```
"myflow:print(5):prtreport"
```

-h

Prints the command usage to stderr and exits.

-V

Prints the Process Manager release version to stderr and exits.

## Examples

---

### Kill a specific flow

---

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

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

### Hold and release a job

---

- Hold a job

```
jjob -i 42 -p "myflow:myjob"
```

In flow with ID 42, flow name myflow, put the job named myjob on hold. The job receives the status On Hold and the flow stops running when it reaches that specific job.

- Release the job

```
jjob -i 42 -g "myflow:myjob"
```

In flow with ID 42, flow name myflow, release the job named myjob. The flow will resume running from that point onward in the flow.

### Hold and release a job array

---

- Hold a job array

```
jjob -i 42 -p -a  
"myflow:myarray"
```

In flow with ID 42, flow name myflow, put the job array named myarray on hold. The job array receives the status On Hold and the flow stops running when it reaches that specific job array.

- Release the job array

```
jjob -i 42 -g -a "myflow:myarray"
```

In flow with ID 42, flow name myflow, release the job array named myarray. The flow will resume running from that point onward in the flow.

### Hold and release a job in a flow array

---

- Hold a job in a flow array

```
jjob -i
45 -p "mymainflow:myflowarray(1):myjob"
```

In flow with ID 45, flow name mymainflow, flow array myflowarray hold the job named myjob in the first element only. The job receives the status On Hold and the subflow stops running when it reaches that specific job in the flow array.

- Release the job in the flow array

```
jjob
-i 45 -g "mymainflow:myflowarray(1):myjob"
```

In flow with ID 45, flow name mymainflow, flow array named myflowarray, release the job named myjob in the first element only. The job receives the status Waiting and the subflow will continue running once it reaches that job in the flow.

- Hold all jobs in all elements in the flow array

```
jjob
-i 45 -p "mymainflow:myflowarray:myjob"
```

- Release all jobs in all elements in the flow array

```
jjob
-i 45 -g "mymainflow:myflowarray:myjob"
```

## 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.

---

## jlicenseupdate

Updates the SAS license, Process Manager entitlement file, and LSF entitlement file

---

## Synopsis

```
jlicenseupdate [-p] license_file
```

```
jlicenseupdate-h |-V
```

---

## Description

Updates the SAS license, or Process Manager and LSF entitlement file with the specified license file.

---

## Options

**-p**

Updates only the Process Manager license file.

*license\_file*

Specifies the path to the license file.

Specify a relative path to indicate the file is located in the current working directory.

-V

Prints Process Manager release version to stderr and exits.

-h

Prints command usage to stderr and exits.

-V

Prints Process Manager release version to stderr and exits.

---

## 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.

## See also

---

**jcomplete**

---

## jpublish

publishes a flow to Process Manager and to other users.

## Synopsis

---

```
jpublish [-u user_name] [-f flow_name]  
jpublish [-h][[-V]]
```

## Description

---

You use the **jpublish** command to publish a target flow to Process Manager.

Subflows and flow arrays linked by reference can only refer to published target flows.

Only Process Manager administrators and control administrators can publish target flows.

## Options

---

**-u** *user\_name*

Specifies the name of the user who owns the flow.

**-f** *flow\_name*

Specifies the name of the flow. If you do not specify a flow name, all flows meeting the other criteria are published.

**-h**

Prints the command usage to stderr and exits.

**-V**

Prints the Process Manager release version to stderr and exits.

## Examples

---

```
jpublish -u userA -f flow1
```

Publishes the flow1 flow belonging to user A.

## See also

---

**junpublish**

---

# jreconfigadmin

dynamically reconfigures and updates the list of administrators.

## Synopsis

---

```
jreconfigadmin [-h][[-V]
```

## Description

---

You use the **jreconfigadmin** command to manually trigger a dynamic reconfiguration and update of the list of administrators.

Run the **jreconfigadmin** command if you changed the list of administrators (by changing the JS\_ADMINS or JS\_CONTROL\_ADMINS parameters in the js.conf file, or by changing the membership in a user group specified in the JS\_ADMINS or JS\_CONTROL\_ADMINS parameters in the js.conf file, or when JS\_ENABLE\_GROUP\_ADMIN=true by changing the membership of an LSF® user group in the LSF lsb.users file) and require this change to apply immediately rather than at the next scheduled update.

If you disabled scheduled updates of the list of administrators (by setting JS\_ADMIN\_UPDATE\_INTERVAL in js.conf to 0), you need to manually run **jsreconfigadmin** whenever you modify the JS\_ADMINS or JS\_CONTROL\_ADMINS parameters, whenever you modify any user groups specified in the JS\_ADMINS or JS\_CONTROL\_ADMINS parameters, or when JS\_ENABLE\_GROUP\_ADMIN=true whenever you modify Group administrators or group members of an LSF user group in the LSF lsb.users file.

You must be a Process Manager administrator account 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.

---

# 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.

---

## 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 committed flow definition from hold. The flow definition is now eligible to be submitted automatically by any of its triggering events. Use this command when you want to resume automatic submission 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 committed 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 committed flow definition from Process Manager. Issuing this command has no impact on any flows associated with the definition, but no further flows can be submitted 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 submission 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, done, or running 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. The flow must have a state of Exit, Done, or Running.

The flow is rerun from the first exited job or starting point, and the flow continues to process as designed.

Note that in order to rerun the flow, work items used as starting points to rerun the flow must have their dependencies met.

If the flow contains multiple branches, the flow is rerun from the first exited jobs or starting points in each branch 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=jdoe" 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 administrative 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

sets values for variables during the runtime of a flow.

## Synopsis

---

```
jsetvars -i flow_ID -l [scope]
jsetvars -i flow_ID [scope:]var=value [[scope:]var=value ...]
jsetvars -i flow_ID -r [scope:]var [[scope:]var ...]
jsetvars -l
jsetvars var=value [var=value ...]
jsetvars -r var [var ...]
jsetvars -h |-V
```

## Description

---

Use the **jsetvars** command to change the value of one or more local variables in a flow at runtime or to change the value of one or more global variables at runtime.

## Options

---

```
jsetvars -i flow_ID -l [scope]
```

List all variables for the flow with the specified ID at the main flow scope.

[*scope*]

If scope is specified, lists all variables for the flow with the specified ID in the specified scope.

Only one scope can be specified. If more than one scope is specified, only the first scope is used. For example, "jsetvars -i 59 -l F2:F1 F2" only list variables at the F2:F1 scope.

```
jsetvars -i flow_ID [scope:]var=value [[scope:]var=value ...]
```

Sets variables for the flow with the specified ID at the main flow scope.

[*scope:*]

If scope is specified, sets variables for the flow with the specified ID in the specified scope.

```
[var=value [[scope:]var=value ...]
```

Specifies the value to which to set the specified variable. Separate variables with a space.

You cannot combine variables of the same scope together. For example, "jsetvars -i 59 F2:F1:A=1 B=2" sets A=1 at the F2:F1 scope, B=2 at the main flow scope.

```
jsetvars -i flow_ID -r [scope:]var [[scope:]var ...]
```

Removes variables for the flow with the specified ID at the main flow scope.

[*scope:*]

If scope is specified, removes variables for the flow with the specified ID in the specified scope.

```
var [[scope:]var ...]
```

Specifies the names of the variables to remove. Separate variables with a space.

```
jsetvars -l
```

List all variables at the global scope.

```
jsetvars var=value [var=value ...]
```

Sets variables at the global scope. Separate variables with a space.

```
jsetvars -r var [var ...]
```

Removes the specified variables at the global scope. Separate variable 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

---

- Set the value of the priority variable to 10 for the flow with the ID 1234:

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

- Set the date global variable value to 05-09-2007:

```
jsetvars date=05-09-2007
```

If the global variable date already exists, changes the value of the `date` variable, otherwise, this adds a new global variable called `date`.

- Delete the time variable from the flow with the ID 1234:

```
jsetvars -i 1234 -r time
```

- Set variables at different scopes:

```
jsetvars -i 21 mainvar1=123 mainvar2=456 MF:SF1:myvar1=abc \
MF:SF1:myvar2=xyz MF:SF2:svar1=333 MF:SF2:svar2=555
```

For the flow with the ID 21, this command sets the *mainvar1* and *mainvar2* variables at the main flow scope level. Also sets the *myvar1* and *myvar2* variables at the subflow level (specifically, the MF:SF1 subflow), and sets the *svar2* variables at the subflow level (specifically, the MF:SF2 subflow). If these variables already exist, this command changes the value of these variables, otherwise, this command adds any new variables that do not already exist.

- Set variables for flow arrays:

```
jsetvars -i 212 MF:FA:myarrayvar=abc#{JS_FLOW_INDEX}
```

For the flow with the ID 212 and assuming MF:FA is a flow array, this command sets the *myarrayvar* variable to *abc1*, *abc2*, *abcX*, for all the different flow array elements (for example, for 212:MF:FA(1), 212:MF:FA(2), and the remaining flow array elements to 212:MF:FA(X)).

- List all variables for a flow:

```
jsetvars -i 21 -l MF:SF1
```

For the flow with the ID 21, lists all variables at the MF:SF1 subflow scope.

- Remove variables at different scopes:

```
jsetvars -i 21 -r mainvar MF:SF1:myvar1 MF:SF1:myvar2 MF:SF2:myvar3
```

For the flow with the ID 21, removes the *mainvar* variable at the main flow scope, removes *myvar1* and *myvar2* variables at the MF:SF1 subflow scope, and removes the *myvar3* variable at the MF:SF2 subflow scope.

---

## jsetversion

sets the default version of a flow.

## Synopsis

---

```
jsetversion -v default_version [-u user_name] flow_name ...
```

```
jsetversion [-h][[-V]]
```

## Description

---

You use the **jsetversion** command to set the default version of the specified flow. The default version of the flow is the version set to be effective at the current time. If you trigger this flow, Process Manager will instantiate the flow instance with the default version.

## Options

---



**-v** *default\_version*

Specifies the version of the flow that you are setting as the default version.

**-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.

*flow\_name*

Specifies the name of the flow for which you are setting the default version.

**-h**

Prints the command usage to stderr and exits.

**-V**

Prints the Process Manager release version to stderr and exits.

## Examples

---

```
jsetversion -v 1.3 flow1
```

Sets version 1.3 as the default version for the flow named `flow1`.

---

## jsinstall

runs **jsinstall**, the Process Manager installation and configuration script

## Synopsis

---

```
jsinstall [-s -y] -f install.config
```

```
jsinstall -h | -V
```

## Description

---

**jsinstall** runs the 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, and configures Process Manager Server.

To uninstall Process Manager, delete the folder in which it was installed.

## Options

---

**-s**

Performs a silent installation of Process Manager.

Creates an installation log `install.log` in the directory in which Process Manager was installed. If any errors occurred, also creates `install.err` in the same location.

`-y`

Accepts the license agreement for the silent installation.

`-f install.config`

Specify the `install.config` file you edited to perform the installation.

Installation parameters are described in the file.

`-h`

Prints the command usage to `stderr` and exits.

`-V`

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

---

## 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. This command has been replaced with jcommit in version 10.1 but can still be used for compatibility. **jsub** and **jcommit** are aliases.

---

## jsubmit

manually submits a previously committed flow definition. Alias for jtrigger.

---

## jtrigger

manually submits a previously committed flow definition.

## Synopsis

---

```
jtrigger [-u user_name] [-e version] [-v "var=value[;var1=value1;...]" [-f variable_file] flow_name flow_name...
```

## Description

---

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

If no version is specified, submits the default version of a flow definition.

If the flow definition is on hold, you can use this command to submit a flow. If the flow definition is not on hold, this command submits an additional execution of the flow. If you want to submit 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.

**-e** *version*

Specifies which version of the flow to submit. You can view versions for a flow definition with the command **jdefs -v**.

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

Specifies to pass variables and their values to the flow. 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).

- To specify a list of variables, separate the variable and value pairs with a semi-colon (;). For example: `var1=1; var2=2;`
- Variable names:
  - Can only contain alphanumeric characters and underscores.
  - Cannot start with a number, cannot contain spaces, and cannot be empty.
  - Can have leading and trailing spaces. Leading and trailing spaces are trimmed.
- Variable values can contain spaces and are kept as is.

**-f** *variable\_file*

Specifies the path to a file that contains variables to pass to the flow.

Specify a relative path to indicate the file is located in the current working directory.

If both the **-v** and **-f** options are used to define variables, variables defined by both options are passed to the flow when triggering it. If the same variable is defined with the **-v** option and with the **-f variable\_file** options, the variable specified with **-v** overrides the same variable specified in the variable file.

Example file:

```
# Example variable file
# Variables are delimited by semicolons.
# The semicolon is not required for the last variable in a line

Var1=value1;Var2=value2Var3=value3;Var4=value4;
```

```
# Leading and trailing spaces in a variable name are trimmed
Var5 = value5 ;    Var6=value6
```

Variable file format:

- To specify a list of variables, separate the variable and value pairs with a semi-colon (;). For example: var1=1; var2=2;
- Each line can contain one or more variables.
- Blank lines are ignored.
- Variable names:
  - Can only contain alphanumeric characters and underscores.
  - Cannot start with a number, cannot contain spaces, and cannot be empty.
  - Can have leading and trailing spaces. Leading and trailing spaces are trimmed.
- Variable values can contain spaces and are kept as is.
- Each comment line starts with # and ends with a line break.

*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
```

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

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

Submit the flow definition payupdt, which is owned by user01.

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

Submit 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**

---

## junpublish

unpublishes a target flow from Process Manager.

## Synopsis

---

```
junpublish [-u user_name] [-f flow_name]
```

junpublish [-h][[-V]

## Description

---

You use the **jpublish** command to unpublish a target flow from Process Manager. Unpublished target flows can no longer be referred to other flows and flow arrays.

Only Process Manager administrators and control administrators can unpublish target flows.

## Options

---

-u *user\_name*

Specifies the name of the user who owns the flow. In Windows, the user name must include the domain in the form of *domain\_name\user\_name*.

-f *flow\_name*

Specifies the name of the flow. If you do not specify a flow name, all flows meeting the other criteria are unpublished.

-h

Prints the command usage to stderr and exits.

-V

Prints the Process Manager release version to stderr and exits.

## Examples

---

```
junpublish -u userA -f flow2
```

Unpublishes the flow2 flow belonging to userA.

```
junpublish -u domainA\userA -f flow2
```

In Windows, unpublishes the flow2 flow belonging to userA, which belongs to the domainA domain.

## See also

---

**jpublish**

---

## licenseinfo

displays information about SAS licenses, LSF licenses, or LSF entitlement files

## Synopsis

---

licenseinfo [-h][-V]

## Description

---

Displays information about SAS licenses, LSF licenses, or LSF entitlement files, including expiry date.

## Options

---

- h  
Prints command usage to stderr and exits.
- V  
Prints Process Manager release version to stderr and exits.

---

## ppmsetvar

sets or removes flow user variables, subflow user variables, and global user variables from a work item in a flow or subflow

## Synopsis

---

```
ppmsetvar -f variable_name=value[ variable_name=value ...]  
ppmsetvar -p variable_name=value[ variable_name=value ...]  
ppmsetvar -g variable_name=value[ variable_name=value ...]  
ppmsetvar -f -r variable_name[ variable_name ...]  
ppmsetvar -p -r variable_name[ variable_name ...]  
ppmsetvar -g -r variable_name[ variable_name ...]  
ppmsetvar -h | -V
```

## Description

---

This command is installed with IBM Spectrum LSF.

Use the **ppmsetvar** command to set user variables or remove user variables from a work item in a flow at runtime, and to set global user variables or remove global user variables at runtime.

You can use **ppmsetvar** only to set variables for LSF® jobs, job scripts, job arrays and job script arrays. You cannot use **ppmsetvar** to set variables for local jobs. To set variables for local jobs, use variable files.

This is a blocking command.

Important: This command uses the LSF bpost command with slots 4, 5, and 6. If anyone is using **bpost** in your LSF cluster, ensure the slots 4, 5, 6 are not used as this will interfere with the **ppmsetvar** command and may lead to unexpected results.

You can use **ppmsetvar** in conjunction with other methods of setting user variables in Process Manager, such as a variable file. For example, you could set some variables using **ppmsetvar** and other variables with a variable file. All user variables will be identified by Process Manager. If you use several methods to set user variables, note that the variable file can override any variables set with **ppmsetvar** as it is read last.

If **ppmsetvar** is used multiple times, the variables will be appended. For example, if you run the following, the end result will be a=10, b=2, c=7, and d=100:

```
ppmsetvar -f a=1 b=2  
ppmsetvar -f a=10 c=7  
ppmsetvar -f d=100
```

## Options

---

- f *variable\_name=value* [*variable\_name=value* ...]  
Sets user variables for a flow. Use this option to set user variables that can only be accessed by work items within a flow. These user variables cannot be accessed from other flows. Separate multiple variables with a space.
- p *variable\_name=value* [*variable\_name=value* ...]  
Sets user variables from a subflow to be used in the parent flow. Use this option when you want to pass a user variable from a subflow to its parent flow. Separate multiple variables with a space.
- g *variable\_name=value* [*variable\_name=value* ...]  
Sets global user variables for flows. Use this option to set a global user variable that is available to all flows in the system. Separate multiple variables with a space.
- f -r *variable\_name* [*variable\_name* ...]  
Clears the specified user variable for work items within a flow. Separate multiple variables with a space.
- p -r *variable\_name* [*variable\_name* ...]  
Clears the specified user variable for the parent subflow. Separate multiple variables with a space.
- g -r *variable\_name* [*variable\_name* ...]  
Clears global user for flows. Separate multiple variables with a space.

## Return Values

---

- 0: command completed successfully.
- 1: command exited due to an invalid parameter.
- 2: command exited due to incorrect variable/value syntax.
- 3: command exited due to unknown reasons.

## Set user variables ABC and XYZ that can be accessed by all work items in a flow

---

```
ppmsetvar -f ABC=123 XYZ=456
```

## Clear flow user variables ABC and XYZ

---

```
ppmsetvar -f -r ABC XYZ
```

## Set a user variable from a subflow to a parent flow

---

For example, the subflow name is `Dynamic_Subflow1`. In this example, the job sets the user variable with the flow short name. The parent flow can access this user variable by indicating in a work item `echo #{result_Dynamic_Subflow1}`. In this case, the result of `echo #{result_Dynamic_Subflow1}` would be `xyz100`.

```
ppmsetvar -p result_#{JS_FLOW_SHORT_NAME}=xyz#{MYVAR}
```

## Clear a user variable from a subflow

---



In this example, the subflow clears the user variable with the subflow short name in its parent subflow. If the subflow name is Subflow1, this command clear the user variable other\_result\_Subflow1.

```
ppmsetvar -p -r other_result_#{JS_FLOW_SHORT_NAME}
```

---

## Set a global user variable from any work item

---

```
ppmsetvar -g MYGLOBAL=all
```

---

## Clear a global user variable from any work item

---

```
ppmsetvar -g -r MYGLOBAL
```

---

## 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.

- [File Structure](#)
- [Files created on the server host](#)
- [Process Manager history files](#)
- [Process Manager log files](#)
- [history.log](#)

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® management host, 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.

- [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.
- [js.conf](#)
- [name.alarm](#)  
When you define an alarm, you create an individual file for each alarm. Alarms are stored in the directory `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 either notify one or more email addresses, or execute a script. To enable the alarm, reload the alarm list using the command **jreconfigalarm**.

---

## 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 describes what each directory contains:

| Directory           | Contents  |
|---------------------|---|
| <version>/app       | Contains the files required to run Process Manager Client.  |
| <version>/bin       | Contains the executables for all of the Process Manager commands and the Process Manager Client applications.   |
| <version>/etc       | Contains the Process Manager messages and the data specification used by the Process Manager software when creating flows.  |
| <version>/examples  | Contains example flows you can use and customize.   |
| <version>/jre       | On Windows only, contains the Java™ runtime environment files for the client applications.  |
| <version>/install   | On UNIX only, contains the Process Manager README file and install.config and other installation-specific information.  |
| <version>/lib       | Contains the Process Manager Java™ files.   |
| <version>/resources | Contains the properties files used by Process Manager.  |
| <version>/man       | On UNIX only, contains the man pages for each of the Process Manager commands.  |
| <version>/platform  | Contains files specifically for running the Process Manager software on each platform.  |
| 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 and recovery data required by Process Manager to complete its processing, including the following directories: <ul style="list-style-type: none"><li>• alarms—contains all alarm definitions</li><li>• calendar—contains all system calendar definitions</li><li>• events—contains persisted events</li><li>• history—contains all historical data</li><li>• lock—contains lock files to prevent multiple Process Manager Servers from accessing the same working files</li><li>• storage—contains copies of active and completed flows</li><li>• system—contains system status data used by Process Manager Server during recovery</li><li>• templates—contains templates for inserting custom applications in a flow</li><li>• var_comm—contains temporary values for user variables</li><li>• variable—contains the current values of any global or local user variables</li><li>• proxy_storage—contains persisted proxy event definitions</li></ul> |

## 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 history files called history.log.<index>.

Process Manager creates a new `history.log.<index>` file when the log age has reached the number of hours specified with the parameter `JS_HISTORY_LIFETIME` in `js.conf`, or the log size has reached the maximum number of bytes indicated with the parameter `JS_HISTORY_SIZE` in `js.conf`. The `<index>` is incremented by 1 every time a new log is created. When JFD is restarted, JFD scans the `JS_HOME/work/history` directory to determine the last index, and uses that for calculating the index of the next history log file.

History log files are periodically cleaned up according to the time period specified by the parameter `JS_HISTORY_CLEAN_PERIOD` in `js.conf`. History log files older than the specified time period are cleaned up by the Process Manager server (JFD). The default is 15 days.

If you do not want history log files to be deleted, you can set the parameter `JS_HISTORY_ARCHIVE_DIR` in `js.conf` and specify a directory in which to store archived history logs. When this parameter is set, instead of deleting the old history logs, the Process Manager Server (JFD) moves them into the directory specified by `JS_HISTORY_ARCHIVE_DIR` according to the time period specified by `JS_HISTORY_CLEAN_PERIOD` in `js.conf`.

The parameter `JS_HISTORY_LIMIT` in `js.conf` specifies the maximum number of history records retrieved when the **jhist** command is used and the Process Manager Client and Process Manager Server are on different hosts. If more than the maximum number of records are available, only the oldest number of records specified in this parameter are retrieved.

---

## 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.

Tip: The Process Manager log and event files can contain personal information (for example, UNIX usernames and email addresses). OS-level access control is one level of protecting this persona data; for further protection, you can use encryption software tools to encrypt the disk partition or volume used to store the log and events files. Process Manager log and event files are located at `JS_TOP/log` and `JS_TOP/work`.

---

## history.log

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® management host, 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 management host. The second line indicates when LSF management host 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.

## 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.

Tip: To prevent users from viewing detailed information for jobs that belong to other users, you can set the `SECURE_INFODIR_USER_ACCESS` parameter to Y or G in the LSF system's `lsb.params` file. If set, also ensure that the primary Process Manager administrator is primary LSF administrator so that the Process Manager server daemon (JFD) can read the `lsb.event` file. Set this configuration within the `JS_ADMIN` parameter for the `install.config` file during installation.

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

For example, to specify users and user groups:

```
JS_ADMINS=lsfadmin engineering_group userA
```

## 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.

For example, to specify users and user groups:

```
JS_CONTROL_ADMINS=admin GroupB userA
```

## 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 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 UNIX, specify just the name of the mail server host. That is:

*JS\_MAILHOST=hostname*

Note: JS\_MAILHOST is equivalent to LSB\_MAILSERVER in LSF.

### Default

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® management host configuration files are stored. There is no default for this value.

## **EGO\_DAEMON\_CONTROL**

---

### **Syntax**

EGO\_DAEMON\_CONTROL=false | true

### **Description**

OPTIONAL

Specifies whether or not to install Process Manager as an EGO service and enable to control JFD.

### **Default**

The default is EGO\_DAEMON\_CONTROL=false.

## **EGO\_CONFDIR**

---

### **Syntax**

EGO\_CONFDIR=/path

### **Description**

REQUIRED if EGO\_DAEMON\_CONTROL=true

Specifies the directory containing the path to the EGO configuration file ego.conf.

### **Default**

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

---

# js.conf

This is the configuration file for Process Manager. Process Manager Server receives its configuration information on startup from its configuration file `js.conf`.

When you make changes in this file, restart `jfd` with the commands **jadmin start** and **jadmin stop** to make changes take effect.

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, UNIX user group names, or Windows active directory user group names.

Tip: To prevent users from viewing detailed information for jobs that belong to other users, you can set the `SECURE_INFODIR_USER_ACCESS` parameter to Y or G in the LSF system's `lsb.params` file. If set, also ensure that the primary Process Manager administrator is primary LSF administrator so that the Process Manager

server daemon (JFD) can read the lsb.event file. Set this configuration within the JS\_ADMIN parameter for the install.config file during installation.

If you change the list of administrators specified in this parameter, or change the membership in a user group specified in this parameter, these changes will be applied at the next scheduled update or by running **jreconfigadmin**.

Windows user IDs and active directory user group names must include the domain name. To specify a list, separate the names with a comma without a space. If the Windows user ID or active directory user group name contains spaces, enclose the user ID or group name in quotation marks.

For example, to specify Windows users and user groups:

```
JS_ADMINS=DOMAIN\lsfadmin,"DOMAIN\Engineering Group",DOMAIN\userA
```

## Default

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

# JS\_ADMIN\_UPDATE\_INTERVAL

---

## Syntax

JS\_ADMIN\_UPDATE\_INTERVAL=days

If set to 0, scheduled updates is disabled.

## Description

Specifies the interval between scheduled updates of the list of Process Manager administrators.

If the membership in a user group changes, the list of Process Manager administrators needs updating. This parameter specifies the interval of time between scheduled updates. You can also manually update the list of Process Manager administrators using the **jreconfigadmin** command.

If you disable scheduled updates (by setting this interval to 0), you need to manually run **jsreconfigadmin** whenever you modify the JS\_ADMINS or JS\_CONTROL\_ADMINS parameters, or whenever you modify any user groups specified in the JS\_ADMINS or JS\_CONTROL\_ADMINS parameters.

## Default

The default is one day.

## See also

JS\_ADMINS, JS\_CONTROL\_ADMINS

# JS\_ALARM\_CMD\_TIMEOUT

---

## Syntax

JS\_ALARM\_CMD\_TIMEOUT=seconds

## Description

Specifies the maximum number of seconds that an alarm script executes before Process Manager forcefully terminates it.

## Default

The default is 180 seconds.

# JS\_BSUB\_RETRY\_EXIT\_VALUES

---

## Syntax

JS\_BSUB\_RETRY\_EXIT\_VALUES=exit\_code[, exit\_code...]

## Description

Specifies **bsub** exit codes to retry job submission. Separate multiple exit codes with a comma (,).

When job submission fails and the LSF **bsub** command exits with any of the specified exit codes, Process Manager retries to submit the job again. The number of retries is specified with the parameter JS\_START\_RETRY in js.conf.

## Default

Undefined-there is no retry when job submission fails.

## See also

JS\_START\_RETRY

# JS\_CHANGE\_FLOW\_OWNER

---

## Syntax

JS\_CHANGE\_FLOW\_OWNER=false | true

## Description

Specifies whether non-administrator users can trigger flows from other users' flow definitions and own the triggered flows.

Applies only to Published flows.

When this parameter is set to false:

- Only the Process Manager administrator, the Process Manager control administrator, and the user who committed the flow definition can submit the flow. The user who committed the flow definition is the owner of the flow. In Flow Manager, the Run As field in the job definition has this user name.

When this parameter is set to true:

- Any user can submit the flow. The user who submits the flow is the owner of the flow.
- In Flow Manager, the value defined in the job definition Run As field is replaced with the user name of the user who submitted the flow.

If a flow definition has a trigger event defined, the flow owner is the user who committed the flow definition.

If a user runs a flow with the **jrun** command or through Run Now in Flow Editor, the flow owner is the user who invokes the command or the Run Now action.

## Permissions

The following table illustrates control permissions when JS\_CHANGE\_FLOW\_OWNER=true.

|  | Can submit other users' non-published flow definitions | Can submit other users' published flow definitions | Flow owner/ job owner     |
|--|--|--|---------------------------|
| Primary administrator, Control administrator | Y  | Y  | User who submit the flow. |
| Non-administrator users                      | N  | Y  | User who submit the flow. |

The following table illustrates control permissions when JS\_CHANGE\_FLOW\_OWNER=false.

| Users  | Can submit other users' non-published flow definitions | Can submit other users' published flow definitions | Flow owner/ job owner                |
|--|--|--|--------------------------------------|
| Primary administrator, Control administrator | Y  | Y  | User defined in the flow definition. |
| Non-administrator users                      | N  | N  | Not applicable.                      |

## User interface affected

In Flow Manager:

- When a user opens the a job definition dialog from the flow diagram, the Run As field always displays the actual job owner.
- Flow and job control action permissions are based on flow owner. The flow owner can:
  - Flows: kill, suspend, resume, rerun, query
  - Jobs: kill, rerun, resume, set job complete, set rerun point
  - Set variables
  - Complete dependencies
  - Complete/query manual jobs

Commands:

- `jtrigger -u user_name`

When JS\_CHANGE\_FLOW\_OWNER=false, **-u** specifies the name of the user who owns the flow definition. This is the user who committed the flow definition to Process Manager. Use this option if you have administrator authority and you are submitting the flow on behalf of another user.

When JS\_CHANGE\_FLOW\_OWNER=true, **-u** specifies the name of the user who owns the flow definition. The flow is owned by the user who submitted the flow. Jobs in the flow run under the submission user, and the submission user is able to control the flow and its jobs.

- Flow commands:

For flow-related commands such as **jflows**, **jkill**, **jmanuals**, **jrerun**, **jresume**, **jstop**, **-u** specifies the owner of the flow: the user who submitted the flow.

In the output of the **jflows** command, the **USER** field indicates the flow owner: the user who submitted the flow. In the **NAME** field, the full name of the flow definition is displayed (example: LSFAD/lsfadmin:untitled).

- Job commands:

For job-related commands such as **jcomplete**, **jjob**, **-u** specifies the owner of the job.

- For the **jhist** command, **-u** specifies the user who owns the category specified by the **-C** option. In the following example, **-u** indicates the owner of the flow definition (user who committed the flow definition):

```
jhist -C myflowdef -u user1 -f myflow
```

In the following example, **-u** specifies the owner of the flow (user who submitted the flow):

```
jhist -C flow -u user1 -f myflow
```

In the following example, **-u** specifies the owner of the job (user who submitted the flow):

```
jhist -C job -u user1 -f myflow
```

- In the history.log file, the user in the **User Name** field is the owner of the category. For example, the user name in the FLOWDEF category is the flow definition owner(user who committed the flow definition), the user name in the FLOW category is the flow owner(user who submitted the flow) and the user in the JOB category is the job owner(user who submitted the flow).

## Default

JS\_CHANGE\_FLOW\_OWNER=false

## See also

JS\_LIMIT\_USER\_VIEW

If you are using JS\_LIMIT\_USER\_VIEW to limit users from viewing other users' flows, when you set JS\_CHANGE\_FLOW\_OWNER=true:

- The user who is logged on can view and control flows that he owns. For example, if **userA** committed and published the flow definition, but **userB** triggered a flow from the flow definition, **userB** can see the flow definition because he is the owner of the flow.

# 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[,cadmin1,cadmin2,...]

## 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 commit or remove flows belonging to other users.

Any administrators specified can be user IDs, UNIX user group names, or Windows active directory user group names.

If you change the list of administrators specified in this parameter, or change the membership in a user group specified in this parameter, these changes will be applied at the next scheduled update or by running **jreconfigadmin**.

Windows user IDs and active directory user group names must include the domain name. To specify a list, separate the names with a comma without a space. If the Windows user ID or active directory group name contains spaces, enclose the user ID or group name in quotation marks.

For example, to specify Windows users and user groups:

```
JS_CONTROL_ADMINS=DOMAIN\admin,"DOMAIN\QA Group",DOMAIN\userA
```

### Default

There is no default for this parameter.

### See also

JS\_ADMINS

## JS\_CREATE\_WORKING\_DIR

---

### Syntax

```
JS_CREATE_WORKING_DIR=false | true
```

### Description

Controls whether Process Manager automatically creates the work directory for a work item. This is useful in cases in which Process Manager does not have access to local directories on server hosts.

When JS\_CREATE\_WORKING\_DIR=true(default value), if a work directory is specified for a work item in its definition, Process Manager creates the specified directory if it does not exist.

When JS\_CREATE\_WORKING\_DIR=false, if a work directory is specified for a work item in its definition, Process Manager assumes the directory already exists in the specified location. Ensure the directory exists or the work item will fail.

### Default

```
JS_CREATE_WORKING_DIR=true
```

## 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\_DEFAULT\_FLOW\_WORKING\_DIR

---

## Syntax

JS\_DEFAULT\_FLOW\_WORKING\_DIR=path

## Description

Specifies the default working directory that is used by flows when no working directory is defined in the flow definition or passed to Process Manager with the variable JS\_FLOW\_WORKING\_DIR.

Process Manager creates the default working directory and any specified subdirectories if the directories do not exist.

To specify subdirectories, you can use the built-in variables:

- %u for user name
- %t for time stamp
- #{JS\_FLOW\_NAME}
- #{JS\_FLOW\_ID}

Work items inherit the flow working directory unless users override the working directory in individual work items.

The override order for working directories is (in order of highest precedence):

1. The working directory defined at the job level, in the Job Definition.
2. The working directory defined at the subflow level, in the subflow's Flow Attributes.
3. The working directory defined at the flow level, in the Flow Attributes.
4. The working directory specified with the variable JS\_FLOW\_WORKING\_DIR when you trigger a flow.
5. The working directory defined with JS\_DEFAULT\_FLOW\_WORKING\_DIR in js.conf.
6. The execution user's home directory:
  - Linux®: \$HOME
  - Windows: %HOMEDRIVE%%HOMEPATH%

Requirements for the working directory:



- The directory and parent directories to the working directory must be shared and accessible to the Process Manager server and all LSF® execution hosts.  
For example, if you specify JS\_DEFAULT\_FLOW\_WORKING\_DIR=/home/lfsadmin/{JS\_FLOW\_NAME}\_%t, the directory /home/lfsadmin must exist and must be shared.
- The shared directory must be writable by the user who submits the flow, and by execution users of individual work items(Run As user).
- If the root directory cannot be shared, the directory:
  - Must exist on all LSF execution hosts
  - Must have the same path on all LSF execution hosts
  - Must be writable by the user who submits the flow definition and by execution users of work items in the flow and any subflows
- Windows: The working directory must be set to a physical drive on the machine such as C:\ . If you need to refer to a network drive, create a symbolic link inside your C:\ drive.

Automatically created directories have the following permissions:

- Owner is the execution user
- Owning group is the execution user's group
- Read and write execute permissions are set for the owner
- Linux: read and write execute permissions are set for all
- Windows: new folders have full permissions set for the execution user

## Default

Undefined. When no working directory is specified for the flow in the flow definition or with JS\_FLOW\_WORKING\_DIR, the location that is used for the working directory is the execution user's home directory:

- Linux: \$HOME
- Windows: %HOMEDRIVE%%HOMEPATH%

## Examples

Create a separate directory for each flow that is triggered from the flow definition in the user's home directory, with a time stamp at the end:

- Linux: JS\_DEFAULT\_FLOW\_WORKING\_DIR=/home/%u/#{JS\_FLOW\_NAME}\_%t  
If the user is user1, the flow name myflow, and the flow ID 123, the directory that is created is:  
  
/home/user1/123:user1:myflow\_1366648793
- Windows: JS\_DEFAULT\_FLOW\_WORKING\_DIR=C:\shared\%u\#{JS\_FLOW\_NAME}\_%t  
If the user is user1, the flow name myflow, and the flow ID 123, the directory that is created is:  
  
c:\shared\123\_user1\_myflow\_1366648793

## JS\_DEFAULT\_USER\_VARIABLE\_VALUE\_IS\_EMPTY

### Syntax

JS\_DEFAULT\_USER\_VARIABLE\_VALUE\_IS\_EMPTY=false | true

### Description

Defines how a user variable whose value is not set is interpreted by Process Manager.

When set to false, when a flow runs and a user variable is specified in the flow and its value is not set, Process Manager interprets the variable as specified. For example: if in the job definition you specified the command `ls`

`#{MYDIR}`, when the flow runs and MYDIR is not set, Process Manager interprets the command as: `ls #{MYDIR}`.

When set to true, when a flow runs and a user variable is specified in the flow and its value is not set, Process Manager interprets the variable as being set to empty. For example: if in the job definition you specified the command `ls`

`#{MYDIR}`, when the flow runs and MYDIR is not set, Process Manager interprets `MYDIR=" "`, and the command as: `ls`.

### Default

The default is false.

## JS\_DTD\_DIR

---

### Syntax

`JS_DTD_DIR=JS_HOME/9.1.0.0/etc`

### Description

DO NOT CHANGE THIS VALUE.

Specifies the directory containing the DTD files required by Process Manager.

### Default

The default is `JS_HOME/10.2/etc`

## JS\_ENABLE\_DOUBLE\_QUOTE

---

### Syntax

`JS_ENABLE_DOUBLE_QUOTE=true | false`

### Description

Applies only to Linux/UNIX.

Applies to LSF jobs. Configures whether job commands are to be quoted with single quotation marks or double quotation marks.

When this parameter is set to false, the LSF **bsub** job command is always quoted with single quotation marks.

When this parameter is set to true:

- If the LSF **bsub** job command does not contain single quotation marks, the job command is quoted with single quotation marks.
- If the LSF **bsub** job command contains single quotation marks, it is quoted with double quotation marks.

Adding double quotation marks means that the command will be interpreted before it is sent to LSF.

For example, if the command is `ls -l | awk {print '$2}'`, it becomes: `bsub "ls -l | awk {print '$2}'"`.

### Default

False. The LSF **bsub** job command is always quoted with single quotation marks.

## JS\_ENABLE\_GROUP\_ADMIN

---

### Syntax

JS\_ENABLE\_GROUP\_ADMIN=true | false

### Description

When set to true, users that are specified in the GROUP\_ADMIN column for an LSF user group in the configuration file `lsb.users` are considered Process Manager Group administrators. In addition, the Owner field is displayed in the Flow Attributes in Flow Editor, and in the Calendar description in Calendar Editor.

For flows:

- Group administrators can operate on flows that are owned by accounts that are listed in the column GROUP\_MEMBER in `lsb.users` for their group.

Tip: If you want a Group administrator to be able to submit, trigger, and control flows that are owned by another Group administrator, specify the other Group administrator account in the column GROUP\_MEMBER in `lsb.users`.

For calendars:

- Group administrators can modify and delete calendars that are owned by accounts that are listed in the column GROUP\_MEMBER in `lsb.users` for their group.

Tip: If you want a Group administrator to be able to modify and delete calendars that are owned by another Group administrator, specify the other Group administrator account in the column GROUP\_MEMBER in `lsb.users`.

### Default

The default is false. Group administrators that are defined in `lsb.users` have no special privileges in Process Manager.

## JS\_ENCRYPTION

---

### Syntax

JS\_ENCRYPTION=true | false

### Description

Specifies whether to encrypt communication between Process Manager Server and Process Manager Client.

### Default

The default is false—do not encrypt communication.

## JS\_EXTERNAL\_EXECUTION

---

### Syntax

JS\_EXTERNAL\_EXECUTION=false | true

### Description

UNIX only.

Specifies that the external execution daemon (EED) is to be enabled. This allows the Process Manager daemon (JFD) to delegate any command execution to the EED so that the JFD does not need to use the **fork()** function to execute commands. This provides a significant performance enhancement if the JFD's memory footprint is large (usually greater than 1 GB).

JFD communicates with EEDs through full-duplex pipes. JFD passes the commands to execute to the EEDs and reads the output of the commands from the EEDs. The EEDs collect the exit status of the commands.

JFD maintains the connection between itself and the EEDs, and restarts any EED that shuts down. If JFD is shut down, the EED will exit in 15 seconds.

### Default

The default is JS\_EXTERNAL\_EXECUTION=false.

## 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.

### See also

JS\_FAILOVER\_HOST, JS\_FOD\_PORT

## JS\_FAILOVER\_HOST

---

### Syntax

JS\_FAILOVER\_HOST=host\_name

### Description

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

Specifies the fully-qualified host name 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 host name as that specified for Process Manager Server.

### **See also**

JS\_FAILOVER, JS\_FOD\_PORT

## **JS\_FILE\_AGE\_EVENT\_REPEATABLE**

---

### **Syntax**

JS\_FILE\_AGE\_EVENT\_REPEATABLE=true | false

### **Description**

Specifies whether Process Manager can repeatedly trigger a flow with a file event and the condition of last modified (age).

When JS\_FILE\_AGE\_EVENT\_REPEATABLE=true, Process Manager can repeatedly trigger a flow with a file event and the condition last modified (age), if the event condition is met.

### **Default**

JS\_FILE\_AGE\_EVENT\_REPEATABLE=false

## **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\_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\_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\_ARCHIVE\_DIR**

---

## Syntax

JS\_HISTORY\_ARCHIVE\_DIR=/path

## Description

Path and name to the directory in which history logs are archived. If the directory does not exist, it is created by Process Manager.

When JS\_HISTORY\_ARCHIVE\_DIR is not defined, any history log files older than the time period specified by JS\_HISTORY\_CLEAN\_PERIOD are deleted by Process Manager.

When JS\_HISTORY\_ARCHIVE\_DIR is defined, any history log files older than the time period specified by JS\_HISTORY\_CLEAN\_PERIOD are moved to the directory specified by JS\_HISTORY\_ARCHIVE\_DIR.

The directory specified by JS\_HISTORY\_ARCHIVE\_DIR must have the same owner and permission as JS\_HOME/work/history/. The directory must be owned and must be writable by the Process Manager administrator, and must be readable by everyone. JFD checks permissions and reports error messages in jfd.log.xxx if it does not have permission to move the history logs into the specified directory.

If failover is configured, the directory specified by JS\_HISTORY\_ARCHIVE\_DIR must be on a shared file system and accessible by both the primary Process Manager server and the failover hosts.

If after setting JS\_HISTORY\_ARCHIVE\_DIR you need to change the location, manually move existing archived history logs to the new location.

You can use the command **jhlist -t** to view archived history logs by running **jhlist** from the Process Manager server host. When **jhlist** is run from the Process Manager server host, Process Manager searches both JS\_HOME/work/history/ and JS\_HISTORY\_ARCHIVE\_DIR for history logs. When **jhlist** is run from a host that is not the Process Manager server host, Process Manager only searches JS\_HOME/work/history/ for history logs.

## Default

Undefined. History log files are deleted according to the time period defined by JS\_HISTORY\_CLEAN\_PERIOD.

# JS\_HISTORY\_CLEAN\_PERIOD

---

## Syntax

JS\_HISTORY\_CLEAN\_PERIOD=days

## Description

Specifies the time period in days for which history log files are stored. When JS\_HISTORY\_ARCHIVE\_DIR is not defined, any history log files older than the specified time period are deleted by Process Manager. If JS\_HISTORY\_ARCHIVE\_DIR is defined, any history log files older than the specified time period are moved to the directory specified by JS\_HISTORY\_ARCHIVE\_DIR.

## Default

The default is 15 days.

# 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\_LIMIT

---

### Syntax

JS\_HISTORY\_LIMIT=number\_of\_records

### Description

Specifies the maximum number of history records retrieved when the **jhist** command is used and your Process Manager Client and Process Manager Server are on different hosts. If more than the maximum number of records are available, only the oldest number of records you specify in this parameter are retrieved.

### Default

The default is 1500 history records.

## 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=host\_name

### **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 completed flow from the working set. Also controls the amount of information saved to the cache file.

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 completed flows per flow definition 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 completed flows per flow definition.

## See also

JS\_IM\_ACTIVEPOLICY, JS\_IM\_POLICY\_LIFETIME, JS\_IM\_POLICY\_CHECKING\_INTERVAL

## JS\_JOB\_SUBMISSION\_RETRY

---

### Syntax

JS\_JOB\_SUBMISSION\_RETRY=true | false

### Description

Deprecated. Use JS\_BSUB\_RETRY\_EXIT\_VALUES instead.

Specifies whether to retry job submission after the job fails.

If JS\_JOB\_SUBMISSION\_RETRY=true and JS\_BSUB\_RETRY\_EXIT\_VALUES is not defined, job submission is retried when the LSF bsub exit code is 1, 255, 127, -1, 128.

If JS\_BSUB\_RETRY\_EXIT\_VALUES is defined, JS\_JOB\_SUBMISSION\_RETRY is ignored and considered deprecated, and Process Manager retries to submit the job again when LSF bsub exits with the exit codes specified in JS\_BSUB\_RETRY\_EXIT\_VALUES.

If JS\_BSUB\_RETRY\_EXIT\_VALUES and JS\_JOB\_SUBMISSION\_RETRY are not defined, there is no retry when job submission fails.

### Default

False. There is no retry when job submission fails.

## JS\_JOB\_SUBMISSION\_TIMEOUT

---

### Syntax

JS\_JOB\_SUBMISSION\_TIMEOUT=seconds

### Description

Applies to job scripts.

Maximum number of seconds that the job script can take to submit jobs to LSF before the Process Manager daemon (jfd) terminates the script.

Specify 0 to set the maximum time to unlimited.

### Default

300 seconds

## JS\_JOB\_SUBMISSION\_SCRIPT\_TIME\_OUT

---

### Syntax

JS\_JOB\_SUBMISSION\_SCRIPT\_TIME\_OUT=seconds

## Description

Specifies the length of time for which the **job submission script** can run before the Process Manager daemon (JFD) kills the script.

## Default

The default is 300 seconds.

# JS\_JOB\_SUBMIT\_NOTICE\_THRESHOLD

---

## Syntax

JS\_JOB\_SUBMIT\_NOTICE\_THRESHOLD=number

## Description

Specifies when job queue size is logged. When the job queue reaches the size specified by JS\_JOB\_SUBMIT\_NOTICE\_THRESHOLD and every multiple of that number, the job queue size is logged in `$JS_TOP/log/jfd.log.host_name`. It is logged at LOG\_NOTICE level.

## Default

100 entries

# JS\_KRB\_KEYTAB\_FILE

---

## Syntax

JS\_KRB\_KEYTAB\_FILE=/path/filename

## Description

Used for Kerberos integration with Process Manager, when the parameter JS\_KRB\_USE\_KEYTAB=true.

Path to the Kerberos keytab file on the Process Manager server host.

## Default

If this parameter is not specified, the default value is `/etc/krb5.keytab` on the Process Manager Server host.

## See also

JS\_KRB\_USE\_KEYTAB

# JS\_KRB\_USE\_KEYTAB

---

## Syntax

JS\_KRB\_USE\_KEYTAB=true | false

## Description

Used for Kerberos integration with Process Manager. When set to true, this parameter specifies to Process Manager to use the Kerberos keytab file to generate user TGTs before reaching the maximum renewal lifetime

defined by the parameter `LSB_KRB_RENEW_MARGIN` in `lsf.conf`. This is useful when you run flows repeatedly over a long period of time (monthly, annually) or when you have flows that run for a very long time so that user TGTs are renewed before the maximum renewal lifetime period is reached.

When set to false, Process Manager automatically renews user TGTs but will be unable to renew them once the maximum renewal lifetime period has been reached. To prevent jobs from failing due to lack of credentials, users with accounts used to run jobs will need to log into Process Manager at least once during the maximum renewal lifetime period so that Process Manager can generate a new user TGT before the maximum renewal lifetime period is reached.

### Default

`JS_KRB_USE_KEYTAB=false`

### See also

`JS_KRB_KEYTAB_FILE`

## JS\_KRB\_USE\_KINIT

---

### Syntax

`JS_KRB_USE_KINIT=true | false`

### Description

Used for Kerberos integration with Process Manager. This parameter works only when `JS_LOGIN_REQUIRED=true`.

When `JS_LOGIN_REQUIRED=false`, this parameter is ignored.

When `JS_KRB_USE_KINIT=false`, Process Manager uses the Pluggable Authentication Module(PAM) on the Process Manager server to generate and renew a user TGT.

Note: You must configure the Pluggable Authentication Module(PAM) on the Process Manager server. Refer to *Administering IBM Spectrum LSF Process Manager* for instructions.

Set `JS_KRB_USE_KINIT=true` if for some reason the system does not allow Process Manager to generate a user TGT or you are unable to configure the Pluggable Authentication Module(PAM) on the Process Manager server. When `JS_KRB_USE_KINIT=true`, each time the user logs in to Process Manager server, Process Manager generates a TGT for the user even if the TGT did not expire.

Note: To generate a TGT, Process Manager uses the Kerberos command **kinit**. Ensure the Kerberos command **kinit** is accessible and executable by the execution user on the Process Manager Server. For example, the execution user can log on to the Process Manager server host and run **kinit** to generate a Kerberos ticket.

### Default

`JS_KRB_USE_KINIT=false`

### See also

`JS_LOGIN_REQUIRED`

## JS\_LARGE\_FLOW\_SAVE

---

## Syntax

JS\_LARGE\_FLOW\_SAVE=y | n

## Description

Used to improve client performance in Flow Editor and Flow Manager when a user opens or submits a flow that is larger than 12 MB.

When set to y, a temporary file is saved in the system temporary directory (Windows: user's temporary directory, Unix/Linux: /var/tmp or /tmp) when a user opens or submits a large flow. This improves client performance as the file is not saved in memory, but temporarily on disk. Once the flow is open or submitted, the file is automatically deleted.

When set to n, when a user opens or submits a flow, the flow is saved in memory.

## Default

Undefined, same as n: the flow is saved in memory when it is opened or submitted.

# 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.

If Process Manager is unable to find the license in the location specified by JS\_LICENSE\_FILE, or the specified license is not valid, Process Manager uses the license file indicated by LSF\_LICENSE\_FILE in lsf.conf. This makes it easier for administration, as the administrator only needs to update the LSF license, and Process Manager will automatically retrieve an updated license.

## Default

The default is *JS\_HOME/conf*.

# JS\_LIMIT\_FLOW\_CHART\_VIEW

---

## Syntax

JS\_LIMIT\_FLOW\_CHART\_VIEW=true | false

## Description

Specifies whether users can see the chart view of a flow and flow definition.

When this parameter is set to false, users who can view a flow or flow definition, can see everything about the flow: flow chart, general information, subflows and jobs, flow data, and flow history. These users can also perform job and subflow-specific actions.

When this parameter is set to true, there are restrictions on which users can see the flow chart of a flow and flow definition and associated actions the user can take on components of the flow.

## Permissions

The following table illustrates permissions when JS\_LIMIT\_FLOW\_CHART\_VIEW=true.

|   | <b>Can see the flow chart of a flow definition</b> | <b>Can see the flow chart of a flow</b>   |
|---|--|---|
| Process Manager administrator (as defined by JS_ADMINS) | Y  | Y   |
| Control administrator (as defined by JS_CONTROL_ADMINS) | Y, if the user is the flow definition owner.       | Y, if the user is both the flow definition owner and flow owner.<br>For example, if a user submitted a flow from another user's published flow definition, he will not be able to view the flow chart. He is the flow owner, but not the flow definition owner. |
| Non-administrator users                                 | Y, if the user is the flow definition owner.       | Y, if the user is both the flow definition owner and flow owner.<br>For example, if a user submitted a flow from another user's published flow definition, he will not be able to view the flow chart. He is the flow owner, but not the flow definition owner. |

The following table illustrates permissions when JS\_LIMIT\_FLOW\_CHART\_VIEW=false.

|   | <b>Can see the flow chart of a flow definition</b> | <b>Can see the flow chart of a flow</b> |
|---|--|---|
| Process Manager administrator (as defined by JS_ADMINS) | Y  | Y                                       |
| Control administrator (as defined by JS_CONTROL_ADMINS) | Y, if the user can see the flow definition.        | Y, if the user can see the flow.        |
| Non-administrator users                                 | Y, if the user can see the flow definition.        | Y, if the user can see the flow.        |

## User interface affected

In Flow Manager:

- If the user does not have permission to see the flow chart: the Open and Open in New Frame on the right-click menu and top drop-down menu will be disabled.

## Default

The default is false.

## See also

JS\_ADMINS, JS\_CONTROL\_ADMINS, JS\_LIMIT\_USER\_VIEW, JS\_CHANGE\_FLOW\_OWNER

# 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. For a guest user, limits the access so that no flows are viewable.

When this parameter is set to true and JS\_CHANGE\_FLOW\_OWNER is set to true:

- The user who is logged on can view and control flow definitions that he owns
- If the flow definition was not created by the user who is logged on, operations on the flow definition are disabled.
- The user who is logged on can view and control flows that he owns.

## Default

The default is false.

## See also

JS\_CHANGE\_FLOW\_OWNER

# JS\_LIMIT\_MODIFY\_GLOBALVAR

---

## Syntax

JS\_LIMIT\_MODIFY\_GLOBALVAR=true | false

## Description

Specifies whether to allow or deny users the privilege of controlling global variables through jsetvars or flow manager. When set to true, only administrators can modify global variables. When set to false, users and administrators can modify global variables.

## Default

The default is true.

# JS\_LOCAL\_EXECUTION\_TIMEOUT

---

## Syntax

JS\_LOCAL\_EXECUTION\_TIMEOUT=seconds

## Description

Specifies the amount of time, in seconds, that each local job is allowed to run before Process Manager forcefully terminates the job. If you set this to be zero or less, Process Manager uses the default value.

## Default

Linux and UNIX: no timeout on the job. There is no limit on how long the local job can run.

Windows: 180 seconds.



## JS\_LOCAL\_JOBS\_LIMIT

---

### Syntax

JS\_LOCAL\_JOBS\_LIMIT=number\_of\_jobs

### Description

Specifies the maximum number of local jobs that can be run in parallel on the Process Manager Server.

When this parameter is set to 0, local jobs are disabled:

- If any existing flows contain local jobs, the local jobs are not run and exit with an exit code of 1.
- In Flow Editor, local jobs cannot be inserted in the flow definition, and any flow definitions that contain local jobs cannot be submitted.
- In Flow Manager, flow definitions that contain local jobs cannot be triggered, released, or published.

### Default

The larger number between 1, and the number of cores on the Process Manager host - 2. For example, if the Process Manager host has 4 cores, the maximum number of local jobs that can be run in parallel is 2.

## 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.

If you set this parameter to true on the Process Manager server, set JS\_LOGIN\_REQUIRED=true in the js.conf file of all Process Manager clients. An error is displayed to the user when the value of the JS\_LOGIN\_REQUIRED parameter on the client does not match that of the server.

If you set this parameter to false on the Process Manager server, you can set JS\_LOGIN\_REQUIRED to either true or false on Process Manager clients. When set to false, users are not required to specify a user name and password to use Process Manager.

## Default

## See Also

JS\_KRB\_USE\_KINIT

# JS\_LOGON\_RETRY

---

## Syntax

JS\_LOGON\_RETRY=number

## Description

Specifies the number of times Process Manager should resubmit the same job to LSF when logon fails.

## Default

The default is 0.

# JS\_LOGON\_RETRY\_DELAY

---

## Syntax

JS\_LOGON\_RETRY\_DELAY=seconds

## Description

Specifies the number of seconds to wait in between each try to resubmit the same job to LSF when logon fails.

## Default

The default is 10 seconds.

# JS\_LOG\_MASK

---

## Syntax

JS\_LOG\_MASK=value

## Description

Specifies the error logging level used. Change this value only as directed by IBM 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\_LSF\_COMMAND\_TIMEOUT

### Syntax

JS\_LSF\_COMMAND\_TIMEOUT=seconds

### Description

Maximum number of seconds that any LSF command can take to execute before the Process Manager daemon (jfd) terminates it. This is used when the Process Manager daemon (jfd) calls any LSF command. If there are problems with command execution, the Process Manager daemon will terminate the process after the specified timeout value.

### Default

300 seconds

## JS\_MAILHOST

---

### Syntax

For Windows: JS\_MAILHOST=[SMTP: | EXCHANGE:]hostname

For Unix: 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 a Microsoft Exchange mail host, specify EXCHANGE:hostname. That is:

```
JS_MAILHOST=[SMTP: | EXCHANGE:]hostname
```

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

```
JS_MAILHOST=hostname
```

Note: JS\_MAILHOST is equivalent to LSB\_MAILSERVER in LSF.

### 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\_MAILPROG

---

### Syntax

`JS_MAILPROG=file_name`

### Description

Path and file name of the mail program used by Process Manager to send email. It affects all emails sent, such as the sending of messages from the Flow Attribute, from alarms, and from manual jobs. Equivalent to `LSB_MAILPROG` in LSF.

You can write your own custom mail program and set `JS_MAILPROG` to the path where this program is located.

The program:

- Can be a shell script, a binary executable, or, a .bat file on Windows. Any program or shell script that accepts the arguments and input, and delivers the mail correctly, can be used.
- Must read the body of the mail message from standard input. The end of the message is marked by end-of-file.
- Must be executable by any user.
- Must follow the same protocol as `sendmail`. For example:

```
/usr/mymail.sh -oi -F "Subject" -f "JFD" usera@ibm.com </dev/stdin
```

Process Manager calls `JS_MAILPROG` with three arguments: one argument gives the full name of the subject - `F "Subject"`, the other argument gives the address of the sender - `-f`, and the third argument the email address to which to send the message.

If you change your mail program, restart `jfd` with the commands **`jadmin start`** and **`jadmin stop`** to make changes take effect.

In a mixed cluster, you can specify different programs for Windows and UNIX. You can set this parameter during installation on Windows.

For your convenience, Process Manager provides the `lsmail` program for Unix and the `lsmail.exe` mail program, which supports SMTP and Microsoft Exchange Server protocols on Windows. If **`lsmail`** is specified, the parameter `LSB_MAILSERVER` must also be specified in LSF. On Windows, `lsmail.exe` can be configured directly. On Unix, the full path to the `lsmail` binary is required for configuration.

### Examples

```
JS_MAILPROG=/serverA/tools/lsf/bin/unixhost.exe
```

### Default

By default, this parameter is undefined and the following default mail programs are used:

- UNIX: **`/usr/lib/sendmail`**
- Windows: `lsmail.exe`

### See also

JS\_MAILHOST to specify the name of the mail server host.

JS\_MAILSENDER to specify the email address of the sender.

## JS\_MAILSENDER

---

### Syntax

JS\_MAILSENDER=emailaddress@emaildomain

### Description

OPTIONAL.

Specifies the email address that is used to send the job notification email. This email address is the sender address of any job notification or alarm emails.

### Valid values

Any valid email address. There cannot be any spaces in the email address.

### Default

The default name of the email sender is JFD.

## 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

## JS\_PAC\_SERVER\_URL

---

### Syntax

JS\_PAC\_SERVER\_URL=http://host\_name:port | https://host\_name:port

### Description

Used for Flow Editor in IBM Spectrum LSF Application Center. Required when submission forms are included in flow definitions.

Specifies the URL to connect to IBM Spectrum LSF Application Center. Process Manager daemon(jfd) contacts LSF Application Center server to submit workload with the submission forms included in flows.

### Default

http://localhost:8080

### Examples

JS\_PAC\_SERVER\_URL=http://host1.example.com:8080

JS\_PAC\_SERVER\_URL=https://host1.example.com:8443

### See also

JS\_SUBMIT\_PAC\_FORM\_TIMEOUT

## JS\_PORT

---

### Syntax

JS\_PORT=number

### Description

REQUIRED.

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

### Default

The default port number is 1966.

### See also

JS\_HOST

## JS\_POSIX\_TZ

---

### Syntax

JS\_POSIX\_TZ=time\_zone

## Description

Use only if your Process Manager server is running on AIX® 6.1, and Olson time zone is set in the /etc/environment file or through the TZ environment variable.

Specifies a time zone according to the POSIX time zone specification. The set time zone must be the equivalent of the Olson time zone set for the system.

This time zone setting does not affect the operating system setting. This setting is used by the Process Manager Server to work around a known issue in AIX 6.1 that ignores the set Olson time zone and uses instead Coordinated Universal Time(UTC)/Greenwich Mean Time(GMT).

The *time\_zone* must be indicated according to the POSIX specification:

**std offset dst**  
**[offset] , start[/time] , end[/time]**

where:

- [ ] indicate optional parameters
- std offset specifies the standard time when the time zone is not in dst
- dst [offset] specifies the time during dst for the time zone
- start[/time] specifies the start time of dst
- end[/time] specifies the end time of dst
- start and end is in the format, Mm.w.d:
- m is the month (number between 1 - 12, and 1 is January)
- w is the week (number between 1 - 5, 1 is first week, 5 is last week of the month)
- d is the day (number between 0 - 6, 0 is Sunday)
- [/time] is in regular time format. For example: 3:00, or simplified to 3. If no time is specified the default is 02:00 or 2.

For additional information on the POSIX time zone, refer to:

[http://www.gnu.org/software/libc/manual/html\\_node/TZ-Variable.html](http://www.gnu.org/software/libc/manual/html_node/TZ-Variable.html).

## Examples

---

| Olson time zone  | Equivalent POSIX time zone |
|------------------|----------------------------|
| America/New_York | EST5EDT, M3.2.0,M11.1.0    |
| Europe/Paris     | CET-1CEST,M3.5.0,M10.5.0/3 |
| Europe/Brussels  | CET-1CEST,M3.5.0,M10.5.0/3 |

---

## Default

Undefined, the time zone used is the time zone set on the operating system of the Process Manager Server.

---

## JS\_PROXY\_DURATION

### Syntax

JS\_PROXY\_DURATION=minutes

### Description

Specifies the length of time within which proxy events should remain valid after becoming true.

A value of 0 indicates that the proxy event will always remain valid and will never expire after it becomes true.

### Default

The default is 0.

## JS\_SERVICE\_STOP\_PEND\_WAIT

---

### Syntax

JS\_SERVICE\_STOP\_PEND\_WAIT=milliseconds

### Description

Windows only.

Specifies the amount of time that the Process Manager daemon (JDF) instructs the Windows service controller to wait before killing the service during a system reboot or shutdown.

When a host is being rebooted or shut down, the Process Manager daemon (JFD) sends a STOP\_PEND message together with a waitHint to the Windows service controller to wait for this amount of time before allowing the system to kill the service.

The system registry key HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\WaitToKillServiceTimeout normally specifies the amount of time that Windows waits before killing all services. JS\_SERVICE\_STOP\_PEND\_WAIT must be less than or equal to this value; otherwise the Windows service controller kills the service in the amount of time as specified in this registry key, before this parameter can take effect.

### Default

The default is specified in the system registry key HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\WaitToKillServiceTimeout. The default value for this system registry key is 20000 milliseconds (20 seconds).

## JS\_SKIP\_HOST\_CHECK

---

### Syntax

JS\_SKIP\_HOST\_CHECK=false | true

### Description

OPTIONAL.

Affects on which host the Process Manager server (jfd) can be started.

When this parameter is undefined or set to false, the Process Manager server (jfd) can only be started on the hosts specified with the parameters JS\_HOST and JS\_FAILOVER\_HOST in js.conf.

When this parameter is set to true, the mechanism to check and ensure that the Process Manager server (jfd) is started on either JS\_HOST or JS\_FAILOVER\_HOST will be skipped. You can start the Process Manager server on any server host in the LSF cluster.



## Default

The default is undefined: the host check is not skipped and the Process Manager server (jfd) can only be started on the hosts specified with the parameters JS\_HOST and JS\_FAILOVER\_HOST in js.conf.

## See also

JS\_FAILOVER, JS\_FAILOVER\_HOST, JS\_HOST in js.conf

# JS\_START\_RETRY

---

## Syntax

JS\_START\_RETRY=retries

## Description

Specifies the maximum number of times Process Manager tries again to submit a job or job array, or start a job or job array before raising a Start Failed exception.

## Default

The default is 20 times.

## See also

JS\_BSUB\_RETRY\_EXIT\_VALUES

# JS\_SUBMISSION\_FORM\_EXEC\_AGENT\_THREADS

---

## Syntax

JS\_SUBMISSION\_FORM\_EXEC\_AGENT\_THREADS=1 | 2 | 3 | 4 | 5

## Description

Used when Process Manager is used in IBM Spectrum LSF Application Center. Maximum number of submission forms that can be simultaneously submitted. This parameter is useful to improve performance when submission forms are included in flow definitions. When submission forms are used by many concurrent users, specify a larger number for this parameter to improve system response.

## Default

2

# JS\_SUBMIT\_PAC\_FORM\_TIMEOUT

---

## Syntax

JS\_SUBMIT\_PAC\_FORM\_TIMEOUT=seconds

## Description

Maximum number of seconds that the Process Manager daemon(jfd) waits to communicate with IBM Spectrum LSF Application Center. This is used when submission forms are included in flow definitions.

## Default

300 seconds

## See also

JS\_PAC\_SERVER\_URL

# JS\_SU\_COMMAND

---

## Syntax

JS\_SU\_COMMAND="command %u"

## Description

Used by Process Manager server to impersonate users for job submission within flows. The Process Manager server runs as root and uses the command **/bin/su** to impersonate other users.

In some cases, you may want Process Manager server to use a different command to impersonate users. Specify the exact command required by the root user to impersonate another user (%u).

When JS\_SU\_COMMAND is set to a value, the parameter JS\_SU\_NEW\_LOGIN is ignored.

## Examples

```
JS_SU_COMMAND="/bin/su %u"
JS_SU_COMMAND="op %u"
```

Note:

If you specify the **op** command, configure **op** in op.conf so that the root user can execute "**op user\_name bash\_script\_to\_bsub**". For example:

```
op lsfadmin /tmp/bsubSubmit.sh
```

An example of op.conf:

```
lsfadmin /bin/bash $* ; uid=lsfadmin
```

## Default

Unset. The default value of JS\_SU\_COMMAND depends on the setting of JS\_SU\_NEW\_LOGIN. If JS\_SU\_NEW\_LOGIN=false(default value), the default value of JS\_SU\_COMMAND is **"/bin/su - %u"**. If JS\_SU\_NEW\_LOGIN=true, the default value of JS\_SU\_COMMAND is **"/bin/su %u"**.

# JS\_SU\_NEW\_LOGIN

---

## Syntax

JS\_SU\_NEW\_LOGIN=true | false

## Description

This parameter is ignored if JS\_SU\_COMMAND is set. Specifies whether or not to start a new login shell when Process Manager server submits jobs to LSF. When this parameter is set to true, a new login shell is started when a job is submitted to LSF.

### Default

JS\_SU\_NEW\_LOGIN=true

## JS\_TIME\_EVENT\_OFFSET

---

### Syntax

JS\_TIME\_EVENT\_OFFSET=integer

### Description

Specifies the time event offset to adjust the LSF Process Manager server time. The offset is added to all the time events of a flow so that they are triggered according to the adjusted server time. The time event calculation compares the difference between the current server time and the time event. The resulting difference is added to the offset defined in JS\_TIME\_EVENT\_OFFSET so that all time events of the flow are triggered according to the adjusted calculation.

The valid range is -180 to 180 minutes.

If a country temporarily changes their time zone (for example, delaying Daylight Savings Time (DST) by a week), or the latest time zone data from <https://github.com/unicode-org/icu-data/tree/master/tzdata/icunew> does not contain the required time zone changes, it may be necessary to use the JS\_TIME\_EVENT\_OFFSET parameter as a temporary measure.

When the time zone offset is no longer required, JS\_TIME\_EVENT\_OFFSET can be disabled (set to 0) and LSF Process Manager restarted so that time events are scheduled at the current time zone.

### Example

Setting JS\_TIME\_EVENT\_OFFSET=90 will trigger all time events 1.5 hours later. If a flow or job within the flow is normally triggered at 2:00, when JS\_TIME\_EVENT\_OFFSET=90, it will be triggered at 3:30.

Setting JS\_TIME\_EVENT\_OFFSET=-90 will trigger all time events 1.5 hours earlier.

### Default

0

## JS\_TIME\_ZONE

---

### Syntax

JS\_TIME\_ZONE=client | server | 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 as 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 client.

## **JS\_UNICODE\_CMD\_UPLUS**

---

### **Syntax**

JS\_UNICODE\_CMD\_UPLUS=true | false

### **Description**

Specifies whether or not to enable double-byte character set on job command. When enabled, the job command runs in native encoding when a job is submitted to the unicodecmd queue.

### **Default**

The default is false—native encoding on job command is not supported.

## **JS\_UNICODE\_ESCAPE\_CONVERT**

---

### **Syntax**

JS\_UNICODE\_ESCAPE\_CONVERT=true | false

### **Description**

Specifies whether Process Manager translates double-byte character sets to the Unicode character escape sequence.

When JS\_UNICODE\_ESCAPE\_CONVERT=true, Process Manager supports double-byte character sets. For example, if a job name contains Chinese characters, Process Manager translates it to the Unicode character escape sequence such as \u1234.

In some cases, you may already have Unicode escape sequences in user names, job names, and so on. You do not want Process Manager to translate to the Unicode character set. You want Process Manager to use the text without converting it into Unicode format. In such cases, set JS\_UNICODE\_ESCAPE\_CONVERT=false. Note, however, that setting JS\_UNICODE\_CONVERT=false disables double-byte character support and as a result, you may see garbled characters.

### **Default**

JS\_UNICODE\_ESCAPE\_CONVERT=true: Process Manager supports double-byte character sets and translates to the Unicode escape sequence.

## JS\_VARIABLE\_CLEANUP\_PERIOD

---

### Syntax

JS\_VARIABLE\_CLEANUP\_PERIOD=hours

### Description

Specifies the cleanup frequency of variable log files. At the specified cleanup period, the JFD Process Manager daemon rewrites the variable.log file to reduce its size. This helps to reduce the startup time next time JFD restarts.

### Default

The default cleanup period is set to 24 hours: JS\_VARIABLE\_CLEANUP\_PERIOD=24

## JS\_WORK\_DIR

---

### Syntax

JS\_WORK\_DIR=/path

### Description

Specifies the name of the directory containing work data.

### Default

The default is *JS\_HOME/work*.

## 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. Alarms are stored in the directory *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 either notify one or more email addresses, or execute a script. To enable the alarm, reload the alarm list using the command **jreconfigalarm**.

## Format

---

- Email notification

```
NOTIFICATION=Email[user_name ...]
```

Specify the "**Email**" command, followed by a space-delimited list of email addresses to notify regarding the exception. Specify the complete email address, or just 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.

- Script execution

```
NOTIFICATION=CMD[/file_path/script_file user_variable ...]
```

Specify the "**CMD**" command, followed by the path to the script file and any user variables (such as the error code). For example:

```
NOTIFICATION=CMD[/home/admin/pageadmin.sh #{ERRORCODE}]
```

Variable values cannot contain the backquote character (`).

## Example

---

The following example shows a database failure alarm definition. The alarm is called DBMSfail.alarm. Its contents are:

```
DESCRIPTION=Send DBA a message indicating DBMS failure
NOTIFICATION=Email[bsmith ajones]
```