Administering Platform Process Manager

Platform Process Manager
Version 8.0
December 2010

Platform Computing

Contents

CHAPTER 1

About Platform Process Manager

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

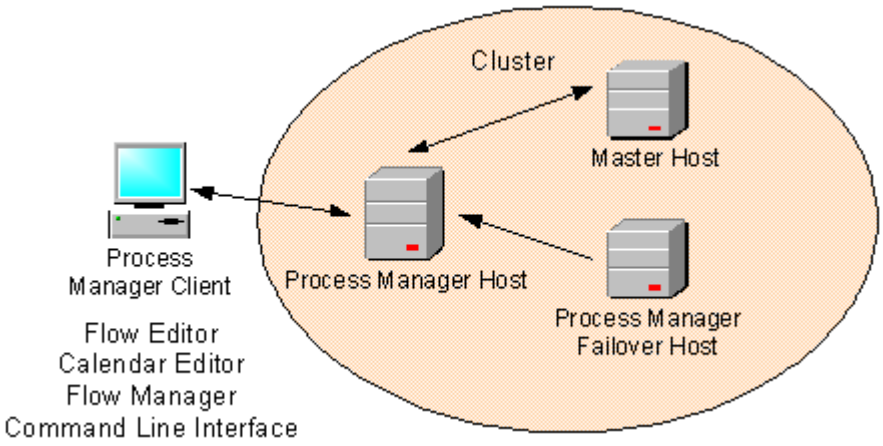
Overview

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

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

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

Components



The system is made up of the following components:

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

Platform Process Manager Server

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

Running multiple Platform Process Manager servers and daemons

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

To support running multiple instances of `jfd`, set **JS_MULTI_INSTANCE=true** in `js.conf`.

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

Platform Process Manager licenses

Platform Process Manager software is licensed per core, not per host or per cluster, so hosts with multicore processors require multiple licenses.

For example, if you run Platform Process Manager on an eight-core host, you will require at least eight licenses.

To support running multiple instances of `jfd` in your Platform LSF cluster, you need either a number of licenses equal to the total number of cores on all Platform Process Manager servers, or the maximum number of `jfd` instances that you are required to run, whichever is greater.

For example, if you run Platform Process Manager on an eight-core host and a four-core host, you will require at least 12 licenses. If you intend to run a total of 16 instances of `jfd` on the two hosts, you will require 16 licenses, rather than 12.

The Platform Process Manager Server failover host

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

Master host

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

LSF master host

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

Platform Process Manager Client

The Platform Process Manager Client contains the graphical client applications that work with Platform Process Manager: the Platform Process Manager Designer and the Flow Manager.

Platform Process Manager Designer

The Platform Process Manager Designer allows users to edit Platform 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 Platform Process Manager.

Calendar Editor

Users use the Calendar Editor to define calendars, which Platform 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. Platform 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 Platform Process Manager administrator can create calendars that can be used by any user of Platform Process Manager. These are referred to as *system* calendars. Platform 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

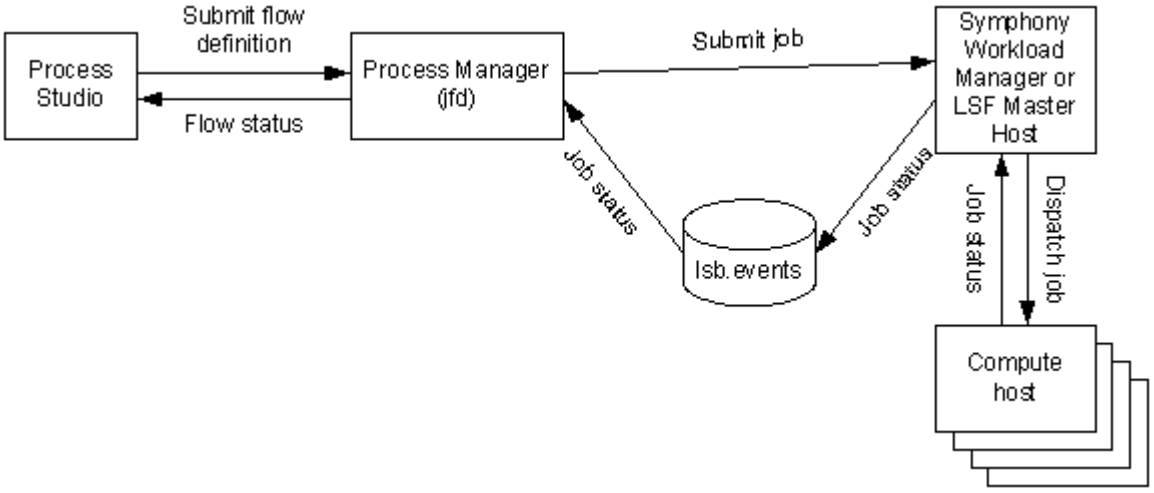
Users use the Flow Manager to trigger, monitor and control running flows, and to obtain history information about completed flows.

The command line interface

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

Data flow

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



1. The user uses the Flow Editor to create a flow definition and submits it to the Platform Process Manager Server.
2. Platform Process Manager Server stores the flow definition in its working directory.
3. When the flow is triggered, Platform Process Manager Server manages the dependencies within the flow. When a job is ready to be run, Platform Process Manager Server submits it to LSF master host.
4. The LSF master host manages any resource dependencies the job may have, and dispatches the job to an appropriate compute host.
5. When the job runs, the compute host sends the status of the job to the LSF master host, which writes the job status to `lsb.events`.
6. Platform Process Manager Server reads `lsb.events` periodically to obtain the status of the jobs it submitted.
7. Platform Process Manager Server uses the status of the job to determine the next appropriate action in the flow.
8. On request from the user, Platform Process Manager Server presents flow status to the Flow Manager.

Security

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

- User authentication
- Role-based access control

User authentication

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

If `JS_LOGIN_REQUIRED=false` , no log in 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 Platform 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 Platform Process Manager Server tries to verify the user name from the domain.

Communications are encrypted using a CAST Cipher with a 64-bit private key.

LDAP

Platform 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, Platform Process Manager uses role-based access control to secure certain types of objects. Any user of Platform Process Manager can create and submit their own flow definitions, and monitor and control their own flows within the Platform Process Manager system, provided that their user ID is recognized by LSF. In addition, all users can view calendars and flows submitted by another user. However, special permissions are required to install and configure Platform Process Manager, or to modify Platform Process Manager items on behalf of another user.

Platform Process Manager recognizes the following roles:

- Primary Platform Process Manager administrator—required to install a Platform Process Manager server and change permissions. It is also the user under which the Platform Process Manager server runs, and is the minimum authority required to stop the Platform Process Manager server.
- Platform Process Manager administrator—can create, delete, modify flows on behalf of another user.
- Platform Process Manager control administrator—can control existing Platform Process Manager items on behalf of another user. This user cannot submit or remove flows belonging to another user.
- Platform Process Manager user—can view calendars and flows owned by another user, but cannot modify them.

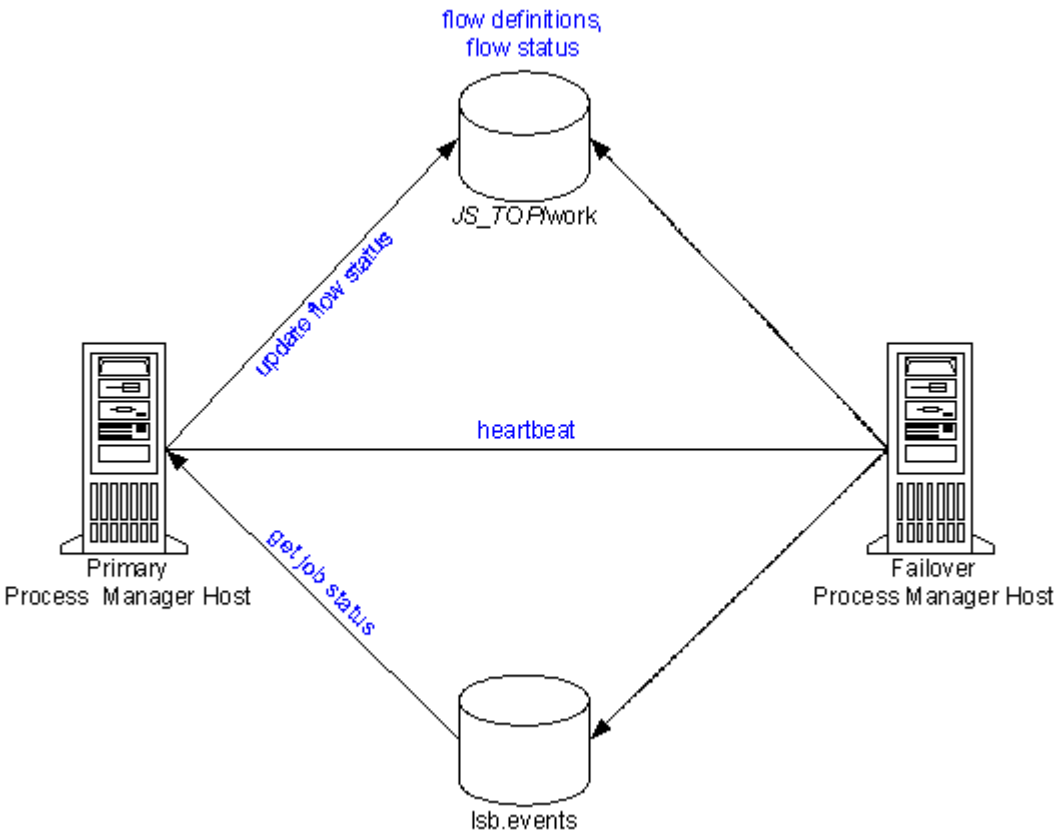
Encrypted communications

You can enable encrypted communications between Platform Process Manager Server and its clients, to further secure the Platform Process Manager network by installing the strong encryption package for your platform. If you want to use this feature, encryption must be enabled on all clients, as well as on the server.

About Failover

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



1. Platform 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. Platform Process Manager Server is now running on the failover host.
3. The `fod` or EGO 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 Platform Process Manager working directory *JS_TOP*`/work`, and the events file `lsb.events`.

About Calendars

Platform 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.

Platform 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 Platform Process Manager administrator. These calendars are owned by the virtual user `Sys` and can be used by any user.

Platform 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 Platform 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 Platform Process Manager Clients from a different geographic time zone than the Platform 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 Platform 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 Platform 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).

Default change

In Platform Process Manager 3.0, the default for JS_TIME_ZONE was server. The default is now client.

Built-in system calendars

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

Types of Calendars	Calendar Names
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 Platform 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

Platform 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. Platform 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.

Platform Process Manager monitors for the following exceptions:

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

Misschedule

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

Overrun

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

Underrun

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

Start Failed

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

Cannot Run

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

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	Overrun	The flow continues to run after the exception occurs. The run time is calculated from when the flow is first triggered until its status changes from Running to Exit or Done, or until the Overrun time is reached, whichever comes first.
	Underrun	The time is calculated from when the flow is first triggered until its status changes from Running to Exit or Done.
Subflow	Misschedule	The subflow is not run.
	Overrun	The subflow continues to run after the exception occurs. The run time is calculated from when the subflow is first triggered until its status changes from Running to Exit or Done, or until the Overrun time is reached, whichever comes first.
	Underrun	The time is calculated from when the subflow first starts running until its status changes from running to Exit or Done.
Job	Misschedule	The job is not run.
	Cannot Run	The job is not run.
	Start Failed	The job is still waiting. Submission of the job is retried until the configured number of retry times. If the job still cannot run, a Cannot Run exception is raised. The default number of retry times is 20.
	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.

When a…	Experiences this exception…	This happens…
Job array	Misschedule	The job array is not run.
	Cannot Run	The job array is not run.
	Start Failed	The job array is still waiting. Submission of the job array is retried the configured number of retry times. If the job array still cannot be started, a Cannot Run exception is raised. The default number of retry times is 20.
	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

Platform 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.

For other types of exceptions where alarms are not available as exception handlers, you can create an alarm directly in the Flow Editor.

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 Platform Process Manager administrator.

Behavior when built-in exception handlers are used

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

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

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

User-defined exception handlers

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

- Running a recovery job
- Triggering another flow

Recovery job

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

Recovery flow

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

IPv6 support

The Platform 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.

  Since Windows XP and Windows Server 2003 do not have this feature, Platform Process Manager creates two sockets for IPv4 and IPv6 on a dual-stack host to handle separate connections from IPv4 and IPv6. This allows all operating systems to handle an IPv4/IPv6 dual-stack host, including supported Windows operating systems.

New features

New method of setting user variables

- You can now use external files to set user variables for the following work items:
  - Jobs and job arrays
  - Job submission scripts and job submission script arrays
  - Local jobs

  These external files are specified by the JS_FLOW_VARIABLE_FILE environment variable for flow variables, by the JS_GLOBAL_VARIABLE_FILE environment variable for global variables, and by the JS_PARENT_FLOW_VARIABLE_FILE environment variable for parent flow variables.

  This external file contains one variable-value pair on each line. The values in the external file must not contain semicolons (;) or control characters.

  Platform Process Manager will not initially create these files — the jobs need to create these files. This approach does not require a job starter, and the job command is not required to use shell scripts. Any binary or script will work — they just need to write to the file.

  The original method of setting flow and global variables (JS_FLOW_VARIABLE_LIST and JS_GLOBAL_VARIABLE_LIST, respectively) still work, but are now deprecated and Platform Process Manager uses this original method internally to obtain the variable-value pairs.
- When setting parent flow variables, you can now use the JS_FLOW_SHORT_NAME built-in variable 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.

  For example, there are two dynamic subflows (`DSF1` and `DSF2`) in a main flow (`11:usr1:F1`), that both refer to the same target flow (`TF`). If the target flow sets a parent flow variable `myvar`, both dynamic subflows will overwrite each other's value of the `myvar` variable.

  To prevent this issue, for all subflows and flow arrays in a flow instance, use the JS_FLOW_SHORT_NAME variable to indicate the name of the subflow.

  For example,

  - For a subflow named `11:usr1:F1:SF1:SSF1`, this variable is set to `SSF1`.
  - For a main flow named `11:usr1:F1`, this variable is set to `F1`.
  - For a flow array element named `11:usr1:F1:FA(1)`, this variable is set to `FA`. Note that this does not include the array index. If you need to differentiate between array elements, you must use the `JS_FLOW_INDEX` built-in user variable.

Using the variable evaluator for decision branch variables

- Using the flow editor, you can now add a new work item to a job flow called the Variable Evaluator (VE). This new work item allows jobs to depend on the evaluation of variable expressions instead of the traditional job exit status, time events, file events, and proxy events. This new work item contains no actual jobs to run, and its purpose is to serve as an intermediate step between jobs and the validation of variable decision branches.

In a typical work flow, the predecessors of a variable evaluator assign values to user variables. When all the variables are set, the variable evaluator will then evaluate all of its variable expression branches and determine which sucessors should start executing.

Flow arrays

- Using the flow editor, you can now add flow arrays as part of a job flow. A flow array works in a similar fashion to a job array, but at a subflow level. A flow array follows the same convention as a subflow and requires a user to select a flow definition to form the array elements. When Platform Process Manager instantiates the flow array, a specified number of flow elements start to run in parallel.

Local jobs

- Using the flow editor, you can now add local jobs as part of a job flow. Local jobs will execute immediately on the Platform Process Manager host without going through LSF.

  Since a local job is short, killing a local job is disabled in Platform Process Manager to simplify the functionality of a local job.

Dynamic subflows and flow arrays

- Using the flow editor, you can now add an existing flow definition as a dynamic subflow within a flow diagram.

  A dynamic subflow, also called a subflow by reference, refers to a target flow that has already been submitted to Platform Process Manager. Only flows that have been submitted to Platform Process Manager and published are eligible target flows for dynamic subflows. When you modify the definition of the target flow, all the mainflows that reference this target flow can obtain the latest version.
- Using the flow editor, you can now add an existing flow array definition as a dynamic flow array within a flow diagram.

  A dynamic flow array, also called a flow array by reference, refers to a target flow that has already been submitted to Platform Process Manager. Only flows that have been submitted to Platform Process Manager and published are eligible target flows for dynamic flow arrays. The target flow is essentially converted into a dynamic flow array in which each array element is equivalent to the target flow.
- To distinguish between the dynamic subflows or flow arrays and the original (non-dynamic) subflows or flow arrays, the original (non-dynamic) subflows are now referred to as static subflows, while the original (non-dynamic) flow arrays are now referred to as static flow arrays.

Flow versioning

- You can now submit a flow with comments to provide a description of each flow version. This makes it easier to track different versions of the flow.
- Using the flow manager, you can now view the version history of the flow to see the different versions of the flow that are submitted. In the flow manager, right-click a flow definition and select View Version, then click View History to view the version history list.
- Using the flow manager, you can now set the default version of the flow. The default version of the flow is the version set to be effective at the current time. If you trigger this flow, Platform Process Manager will instantiate the flow instance with the default version.

  In a dynamic subflow that is automatically updated, the currently-used version is the same as the default version. In a dynamic subflow that is manually updated, the currently-used version is the default version of the target flow at the main flow submission time, or the default version at the time that you last manually updated the dynamic subflow.

- You can now set the default version of the specified flow using the `jsetversion` command.
- The `jflows -l` command now shows the currently-used version of the specified flow (specifically, **jflows -l -f *flow_name***).
- The `jdefs -l` command now shows the default version and the latest version of the specified flow (specifically, **jdefs -l -f *flow_name***).
- You can now view the version history of a flow by using the `jdefs -v` command.

Defining and viewing input variables in flows and jobs

- You can now specify user variables (in addition to environment variables) when specifying flow attributes to allow a flow to use these variables. Previously, you could only define environment variables, but now you can define these variables as input variables.
- You can now view the name and values of input and local variables when viewing the Runtime Attributes of a flow, or by using the `-l` option of the `jjob` command.
- Using the flow manager, you can now set user variables when triggering flows by using the Trigger > With Variables menu option.
- Input variables are now available to static and dynamic subflows, and flow arrays.
- Input variables for jobs are now referred to as environment variables.

Viewing and setting flow variables

- Using the flow manager, you can now view and set flow variables for subflows and flow arrays in addition to main flows.
- In the flow manager, the Set Variables menu item is now Set Flow Variables to reflect the fact that this menu item is now available to subflows and flow arrays (in addition to flows). Navigate to this menu item by by right-clicking on a flow instance that has a status of Exit, Running, or Suspended.
- The `jsetvars` command now supports the `-s`, `-r`, and `-l` options for use with `-i` *flow_ID*. This means that you can now add, edit, remove, or view flow variables (in addition to global variables by using the `-g` option).
- The `jsetvars` command now allows you to specify the scope for the `-s`, `-r`, and `-l` options. This means that you can now specify a specific scope (such as for a specific subflow) when adding, editing, removing, or viewing flow variables.

Setting additional starting points to rerun flows

- Using the flow manager, you can set specific work items in the flow from which to rerun the flow. This allows you to have more flexibility in correcting errors in a flow by rerunning jobs other than the last exited job in the flow.

  Exited jobs in the flow automatically become rerun starting points in addition to any rerun starting points that you have set.

Displaying jobs that are pending in LSF

- Platform Process Manager now displays a brown border around jobs that are pending in LSF in a flow, which means that the job is waiting in a queue for scheduling and dispatch. Previously, Platform Process Manager treated these jobs as Running and displayed a green border around these jobs. Tooltips and job runtime attributes show these jobs as Pending in LSF, while `jflows -l` output shows these jobs as `Pending`.

  Unless otherwise specified, any operation that applies to Running jobs also applies to jobs that are Pending in LSF.

Flow working directory

- You can now specify the working directory for flows, subflows, and flow arrays (in addition to jobs and job arrays). All valid inner work items (subflows, jobs, and job arrays) in the flow will use this directory as the working directory unless you further specify a working directory for the inner work item. In this case, the working directory setting for the inner work item will override the setting for this flow.

  As for job and job array working directories, you can use user variables when specifying the flow working directory.

Multiple Platform Process Manager servers

- You can now run multiple Platform Process Manager servers in a single Platform LSF cluster. This is useful, for example, if you have different Platform Process Manager environments running in one cluster.

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

  You can only run multiple Platform Process Manager servers if they are on UNIX hosts.

New version of JRE

- JRE has changed from version 1.4 to 1.6.

Deprecated support for Solaris 7

- Platform Process Manager no longer supports Solaris 7.

User priority

- Platform Process Manager no longer allows you to set the user priority for jobs if your Platform LSF cluster does not have user priority enabled (that is, if `MAX_USER_PRIORITY` is undefined in `lsb.params`). Previously, if you defined user priority under these circumstances, your job submission would fail.

  Define the user priority for your job in the Processing tab in the Edit Job dialog.

LDAP support

- Platform Process Manager now supports LDAP authentication through PAM on UNIX hosts.

  PAM (Pluggable Authentical Modules) is a third-party tool that you can configure to use the `pam_ldap` module from the `libpam-ldap` package to log into the LDAP server for password authentication.

Silent installation

- You can now install Platform Process Manager silently on Windows hosts without requiring user input. To accomplish this, run the installer once to record your responses in a response file, then run subsequent installations using the response file to automate your installation responses

Installing a Windows host to a UNIX cluster

- You can now install Platform Process Manager on a Windows host and have that host join a UNIX cluster. This results in a mixed cluster of UNIX and Windows hosts.

  When running the Platform Process Manager installer on a Windows host, select Join an existing UNIX cluster in the Installation Options dialog to add the Windows host to a UNIX cluster.

CHAPTER

2

Maintaining Platform Process Manager

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

Configure a failover host (managed by EGO)

For EGO failover to function correctly, Platform Process Manager must have its `conf` and `work` directories installed in a shared location.

When you install Platform Process Manager as an EGO service, Platform Process Manager benefits from the failover features of Platform EGO. If the server running Platform Process Manager fails, Platform EGO relocates and restarts Platform Process Manager on another host.

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: `%LSFENVDIR%\ego\`*`cluster_name`*`\eservice\esc\conf\services\processmanager.xml`
   - UNIX: `$LSFENVDIR/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 this line to the following:

   ```
   <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')</ego:ResourceRequirement>
```

5. Save and close the file.
6. Restart EGO to apply your changes.

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

Note:

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

When you install Platform Process Manager Server, the failover daemon `fod` is automatically installed. You only need to license and configure the failover host. It is recommended that you do this prior to installing a large number of Platform 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 Platform Process Manager Server on the failover host, and start the failover host.

Configure the primary host

1. Log on to the Platform Process Manager Server host as `root` or as the primary Platform 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 Platform Process Manager Server and make your changes take effect.

Prepare the installation files on the failover host

1. Make sure that you have access to the Platform Process Manager distribution files.
   a) Copy the installer to the Platform Process Manager directory.
   b) Untar the package (for example, ppm8.0_pinstall.tar.Z).

      ```
      % zcat ppm8.0_pinstall.tar.Z|tar xvf -
      ```

      This creates a directory called `ppm8.0_pinstall`. For example:

      ```
      % ls /usr/share/pmanager/ppm8.0_pinstall/
      ```

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

Prepare the configuration on the failover host

1. Log on to the failover host as `root` or as the primary Platform Process Manager administrator.
2. Make the Platform Process Manager directory current. For example:

```
# cd /usr/share/pmanager/ppm8.0_pinstall
```

3. Copy `install.config` from the Platform 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

1. Run `jsinstall` to start the installation:

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

   Logging installation sequence in `/usr/share/pmanager/ppm8.0_pinstall/ppm8.0_pinstall/Install.log`

2. Select the Platform Process Manager Server. For example:

```
Searching for Platform Process Manager tar files in /usr/share/pmanager/
ppm8.0_pinstall please wait ...
1) Linux 2.6-glibc2.3-x86 Server
2) Linux 2.6-glibc2.3-x86 Flow Editor and Calendar Editor Client
3) Linux 2.6-glibc2.3-x86 Flow Manager 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 Platform Process Manager environment:

   - On `csh` or `tcsh`:

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

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

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

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

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

```
# jadmin start
```

Add a UNIX client

1. Copy the client tar file for the operating system Platform Process Manager Client will run on to the UNIX host on which you want to install Platform Process Manager.

   For example, ppm8.0_pinstall.tar.Z.
2. Untar ppm8.0_pinstall.tar.Z as follows:

   ```
   % zcat ppm8.0_pinstall.tar.Z|tar xvf -
   ```

   This creates a directory called `ppm8.0_pinstall`.
3. In `ppm8.0_pinstall`, edit section 1 of the file `install.config` to define your configuration.

   Remove the comment symbol (#) and set values for the following parameters:

   - For JS_TOP, specify the full path to the top-level Platform Process Manager installation directory. The installation script will create the directory you specify.
   - For JS_HOST, specify the fully qualified hostname of the host on which the Platform Process Manager daemon will run. You can specify only one host, as each host requires its own configuration files.
   - For JS_PORT, specify the port number through which the clients will access the Platform Process Manager Server. The default is 1966.
   - For JS_TARDIR, specify the full path to the directory containing the Platform Process Manager distribution tar files. The default is the parent directory of the current working directory where `jsinstall` is running.

Add a Windows client

1. Copy `ppm8.0_pinstall_win.exe` to the desktop or a shared file location from which you can run it.
2. Run `ppm8.0_pinstall_win.exe` to start the installation.
3. In the Welcome dialog, click Next
4. In the Choose Destination Location dialog, click Next to use to the default location; or click Browse... to select a different directory. Click Next.
5. In the Select Components dialog, select the components to install and click **Next**.

   - Flow Editor and Calendar Editor
   - Flow Manager

   Click Next to continue.
6. In the Client Configuration dialog:
   a) In the Host name field, specify the name of the Platform Process Manager host the desktop will connect to.
   b) In the Port field, specify the port number of the Platform Process Manager host. If you used the default port number for the Server, leave the value at 1966.
   c) Click Next.
7. Verify that the settings are correct, and click Next to complete the installation.
8. Click Finish.
9. When the installation is complete, from the Start menu, select Platform Computing and Platform Process Manager, and the appropriate application: Flow Editor, Flow Manager, or Calendar Editor.

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

Run the Platform Process Manager server on system startup

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

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

About Platform Process Manager variables

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

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

Platform Process Manager users can set a value for a single variable within a script, or set values for a list of variables, and make all of the values available to the flow or to the Platform 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

Platform Process Manager supports three types of variables:

- Built-in variables
- User variables
- Environment variables

Built-in variables

Built-in variables are those defined by Platform Process Manager, where the value is obtained automatically by Platform Process Manager and made available for use by a flow. No special setup is required to use Platform 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, where the value is set at runtime within a UNIX script or Windows `.bat` file, and made available to Platform Process Manager. To use a user variable, you must first create a job that sets a runtime value for the variable and exports it to Platform Process Manager. You submit that job to a special queue that is configured to set variables. See your Platform Process Manager administrator for the queue name. 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 Platform Process Manager users can set:

- Local variables—those whose values are available only to jobs, job arrays, subflows or events within the current flow. These variables are set in JS_FLOW_VARIABLE_LIST or in a file specified by JS_FLOW_VARIABLE_FILE.
  - Parent variables are local variables whose values 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 in 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. For more details, refer to *Using Platform Platform Process Manager*.
- Global variables—those whose values are available to all the flows within the Platform Process Manager Server. These variables are set in JS_GLOBAL_VARIABLE_LIST or in a file specified by JS_GLOBAL_VARIABLE_FILE.

User variables can also be used inside environment variables.

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

The variables set by the job have similar scope to variables in any programming language (C, for example). If the job sets the variable in JS_FLOW_VARIABLE_LIST (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 in the file specified by JS_PARENT_FLOW_VARIABLE_FILE within a subflow, the user variable is passed to the parent flow.

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

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

How variables are set

How user variables are set

User variables are set using the following methods:

- Job starter
- External file

Job starter

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

External file

Platform Process Manager can set user variables by writing to an external file. This method does not require a job starter, and the job command is not required to use shell scripts. Any binary or script will work, as long as it can write to the file. Platform 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.

Therefore, 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. Platform Process Manager will not initially create these files — the files need to be created by the jobs.

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

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

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:\

Dedicate the Platform Process Manager Server Host

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

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

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

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

Configure LDAP authentication

On UNIX systems, Platform Process Manager supports LDAP (Lightweight Directory Access Protocol) authentication through PAM (Pluggable Authentication Modules). PAM is a third-party tool that you can configure to use the `pam_ldap` module from the `libpam-ldap` package to log into the LDAP server for password checking.

Configure LDAP authentication in the Platform Process Manager server as follows:

1. Edit the js.conf file and set the following parameter:

   JS_LOGIN_REQUIRED=true

2. Modify the PAM configuration in your system to add a new service name (`eauth_userpass`) for the `auth` module type.

   - On Linux, create a new file `/etc/pam.d/eauth_userpass` and add the following line to the new file:

```
auth  required  /lib/security/$ISA/pam_ldap.so
```

   - On Solaris, edit the `/etc/pam.conf` file and add the following lines:

```
eauth_userpass  auth requisite /usr/lib/security/64/pam_authtok_get.so.1
eauth_userpass  auth required  /usr/lib/security/64/pam_dhkeys.so.1
eauth_userpass  auth required  /usr/lib/security/64/pam_unix_cred.so.1
eauth_userpass  auth binding   /usr/lib/security/64/pam_passwd_auth.so.1 server_policy
eauth_userpass  auth required  /usr/lib/security/64/pam_ldap.so.1
```

   - On AIX, edit the `/etc/pam.conf` file and add the following line:

```
eauth_userpass auth required /usr/lib/security/64/pam_aix
```

   Note:

   The absolute file path for the `pam_ldap` module may be different on your host. You must ensure that you specify the 64-bit pam_ldap module on 64-bit operating systems.

3. Restart the Platform Process Manager Server.

Configure an alarm

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

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

The alarm file contains the following parameters:

```
DESCRIPTION=<description>
NOTIFICATION=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.

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

   ```
   DBError.alarm
   ```

   The name you specify will appear in the Flow Editor in the list of available alarms.
2. Optional. Specify a meaningful description for the alarm. For example:

   ```
   DESCRIPTION=Send DBA a message indicating DBMS failure
   ```
3. Required. Specify 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

Configure to support user variables

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

1. If the Platform Process Manager Server runs on UNIX, and users will be setting variables in jobs that run on UNIX hosts, go to *Configure variables for UNIX hosts* on page 47.
2. If the Platform Process Manager Server runs on Windows, and users will be setting variables in jobs that run on Windows hosts, go to *Configure variables for Windows hosts* on page 48.
3. If the Platform Process Manager Server runs on UNIX and users will be setting variables from both UNIX and Windows hosts, go to you need to follow both sets of instructions.
4. If your users will be using many variables in any job definition field, you may need to increase the number of variables that can be substituted at a time per field. Go to *Increase the number of variables that can be substituted* on page 51 for instructions.

Configure variables for UNIX hosts

1. Configure one or more UNIX-specific queues to accept jobs that set variables. See *Configure a queue to support setting user variables* on page 50 for instructions.
2. Ensure that the korn shell (ksh) is available on the host, as the korn shell is required to export variables on UNIX.
3. Ensure that the *JS_TOP* directory is accessible by all LSF hosts that will run jobs that set variables—on a shared file system.

Configure variables for Windows hosts

1. Configure one or more Windows-specific queues to accept jobs that set variables. See *Configure a queue to support setting user variables* on page 50 for instructions.
2. Ensure that the *JS_TOP* directory is accessible by all LSF hosts that will run jobs that set variables—on a shared file system.

Configure variables for both UNIX and Windows hosts

1. Configure at least one Windows-specific queue and at least one Linux-specific queue to accept jobs that set variables. See *Configure a queue to support setting user variables* on page 50 for instructions.
2. On the UNIX LSF hosts, ensure that the korn shell (ksh) is available, as the korn shell is required to export variables on UNIX.
3. Log on to the Platform Process Manager Server host as root or as the primary Platform Process Manager administrator.
4. Configure the Server host as follows:
   a) Copy `ppm8.0_writevar_w2k.tar.Z` to the directory containing the Platform Process Manager distribution files.
   b) Run `jsinstall` to start the installation:

      ```
      # ./jsinstall -f install.config
      ```
   c) Select Windows 2000 Variables from the list of components to install.
   d) Press Enter to complete the installation.
5. Edit `jsstarter.bat`
6. Set a value for *JS_TOP*. For example:

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

Configure a queue to support setting user variables

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

1. Create a new queue in the LSF queues file `lsb.queues`. If users will be setting variables in both UNIX and Windows jobs, you will need a separate queue for each.
2. Add the variable JOB_STARTER in the queue configuration to point to the starter script shipped with Platform Process Manager. Starter scripts are available in `JS_TOP/8.0/bin`.

   For example, for a UNIX queue:

   JOB_STARTER=*JS_TOP*/8.0/bin/jsstarter

   For example, for a Windows queue:

   JOB_STARTER=*JS_TOP*\8.0\bin\jsstarter.bat

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

Increase the number of variables that can be substituted

1. Stop the Platform Process Manager Server and edit `js.conf`.
2. Add a line that specifies the maximum number of variable substitutions that can be performed in a single job definition field by specifying a value for JS_MAX_VAR_SUBSTITUTIONS For example:

   ```
   JS_MAX_VAR_SUBSTITUTIONS=20
   ```

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

Control the Platform Process Manager Server

Starting and stopping the Server on UNIX

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

Start the Platform Process Manager daemon

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

Stop the Platform Process Manager daemon

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

Start and stop the Server on Windows

On Windows, the Platform 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 Platform Process Manager service

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

Stop the Platform Process Manager service

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

Forcing a system snapshot

Periodically, Platform 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 Platform 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, Platform 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 Platform 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`.

1. Log on to the Platform Process Manager Server host as `root` or as the primary Platform 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.
   ```

Change the Configuration

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

Change a configuration value on UNIX

1. Log on to the Platform Process Manager Server host as `root` or as the primary Platform 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 Platform Process Manager Server and make your changes take effect.

Change a configuration value on Windows

1. Stop the Platform Process Manager Server service.
2. Edit *JS_TOP*`/conf/js.conf.`
3. Make your changes.
4. Save `js.conf`.
5. Start the Platform Process Manager Server service to make your changes take effect.

Add an administrator

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

Platform Process Manager recognizes the following kinds of administrators:

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

1. Stop the Platform Process Manager Server and edit `js.conf`.
2. To add a Platform Process Manager administrator, for the JS_ADMINS 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_ADMINS=DOMAIN\lsfadmin,"DOMAIN\Engineering Group",DOMAIN\userA
3. For JS_CONTROL_ADMINS, specify one or more user IDs or UNIX user group names.

   To specify a list, separate the names with a comma. 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 starting the Platform Process Manager Server.

Sign on as a guest

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
- jcals

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.

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.

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

Create system calendars

Platform Process Manager uses system calendars to share scheduling expressions that are commonly used. System calendars are created by the Platform 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

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

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

  calendar_name@Sys

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

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

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

Update the Holidays@Sys calendar

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

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

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

Delete a calendar

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

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 Platform Process Manager Server reinstalls the system calendars that come with Platform Process Manager.

1. Stop Platform 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 Platform Process Manager to have the change take effect.

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.

Specify the mail host

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

1. Stop the Platform 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 Platform Process Manager Server.

Change the job start retry value

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

1. Stop the Platform 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 Platform Process Manager Server.

Change the history setting

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

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

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

   and change them as follows:

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

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

Troubleshooting

Platform Process Manager daemon cannot restart—port is in use

The problem:

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

The solution:

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

Overrun exception triggers at incorrect time

The problem

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

The solution

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

After deleting a calendar, user cannot find flow

The problem

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

The solution

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

Unable to run GUI on linux 2.2 through XTERM

The problem

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

The solution

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

limit stacksize 2048

For bash:

ulimit -s 2048

Not all user variables are replaced

The problem

The user specified more than the configured maximum number of user variables that can be substituted in a single field.

The solution

Increase the value for `JS_MAX_VAR_SUBSTITUTIONS` in `js.conf`.

User is unable to trigger their own flow

The problem

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

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

The solution

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

- jdoe
- Jdoe
- JDOE

CHAPTER

3

Mainframe support

Platform 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 Platform 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

To use the mainframe support, you must:

1. Copy the template file z/OS_Template.xml from JS_TOP/8.0/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.

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\8.0\etc\zos -k]
End Queue
```

CHAPTER

4

Daemons

- jfd
- fod

jfd

Platform 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 submitted to Platform 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 master 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 Platform 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 Platform Technical Support.

-V

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

See also

`fod`, `jadmin`

fod

Platform Process Manager Server failover daemon.

Synopsis

fod

Description

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

Description

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

See also

`jfd`, `jadmin`

CHAPTER

5

Commands

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

Platform 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 Platform 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
- `jcmod`—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 Platform Process Manager
- `jsub`—to submit a flow definition to Platform Process Manager
- `jtrigger`—to trigger the creation of a flow
- `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
- `jremove`—to remove a flow definition from Platform Process Manager

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
- `jpublish`—to publish target flows for use by dynamic flows and flow arrays
- `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
- `junpublish`—to unpublish target flows and remove them from the list for use by dynamic flows and flow arrays

Other commands

- `jid`—to verify the connection between the Platform Process Manager Client and the Platform Process Manager Server
- `jadmin`—to control the Platform Process Manager daemon on Unix
- `jhist`—to view the historic information about server, flow definitions, flows, and jobs.
- `jreconfigalarm`—to reload the alarm definitions.

caleditor

starts the Calendar Editor.

Synopsis

caleditor

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

Examples

caleditor

opens the Calendar Editor.

floweditor

starts the Flow Editor.

Synopsis

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

Description

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

Options

file_name

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

Examples

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

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

floweditor

opens the Flow Editor with no files opened.

flowmanager

starts the Flow Manager.

Synopsis

flowmanager

Description

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

Example

flowmanager

opens the Flow Manager.

jadmin

controls the Platform 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 Platform Process Manager daemon. You must be either `root` or the primary Platform Process Manager administrator to stop the Platform Process Manager daemon.

Options

start

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

-s start

Starts the Platform Process Manager daemon on UNIX in single-user mode. Ensure Platform Process Manager is up and running before you start the Platform Process Manager daemon. You must be the primary Platform Process Manager administrator to use this option.

stop

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

-h

Prints the command usage to stderr and exits.

-V

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

Examples

jadmin start

Starts the Platform Process Manager daemon.

jadmin -s start

Starts the Platform Process Manager daemon in single-user mode.

jadmin stop

Stops the Platform Process Manager daemon.

See also

`jfd`, `js.conf`

jalarms

lists the open alarms in Platform 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 Platform Process Manager release version to `stderr` and exits.

Time interval format

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

Specify the time interval is follows:

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

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

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

Where:

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

start_time*,*end_time

Specifies both the start and end times of the interval.

***start_time*,**

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

,*end_time*

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

start_time

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

Absolute time examples

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

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

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

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

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

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

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

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

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

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

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

Relative time examples

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

,.-2/ = the time of the first occurrence to Mar 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

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

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

jcadd

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

Synopsis

jcadd [**-d** *description*] [**-s**] **-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.

-s

Specifies that you are creating a system calendar. You must be a Platform 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 (_) in the calendar name.

-h

Prints the command usage to `stderr` and exits.

-V

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

Limitations

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

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

Some examples that follow this format are:

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

where ABC, DEF, HIJ are predefined calendars.

Creating calendar expressions

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

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

To create absolute dates:

Specify the date in the following standard format:

```
(yyyy/mm/dd)
```

For example:

```
(2001/12/31)
```

Specify multiple dates separated by commas. For example:

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

To create schedules that recur daily:

Specify the expression in the following format:

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

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

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

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

To create schedules that recur weekly:

Specify the expression in one of the following formats:

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

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

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

or

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

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

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

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

To create schedules that recur monthly:

Specify the expression in one of the following formats:

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

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

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

or

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

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

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

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

or

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

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

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

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

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

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

Examples

```
jcals -u all
```

Displays all calendars in Platform 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 Platform 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 Platform Process Manager release version to `stderr` and exits.

Examples

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

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

See also

`jcadd`, `jcals`

jcmod

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

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 Platform 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.

-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 Platform 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

Platform 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 Platform Process Manager. The following are the reserved words:

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

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

EXAMPLES

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

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

jcomplete

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

Synopsis

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

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

Description

You use the `jcomplete` command to mark a manual job complete, to tell Platform 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 Platform Process Manager administrator to complete a manual job.

Options

-d *description*

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

-i *flow_id*

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

flow_name*:*subflow_name*:*manual_job_name

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

```
myflow:prtcheck:prtpage
```

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

-u *user_name*

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

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Platform 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 Platform Process Manager for the specified user.

Synopsis

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

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 trigger the flow
- any exit conditions specified in the flow definition
- the default version and the latest version 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.

-v

Displays the version history of the flow.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Platform 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.

jflows

displays information about the flows in Platform 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 Platform 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 Platform Process Manager Server, calendars, flow definitions, flows, and jobs.

Synopsis

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

jhist [**-h**|**-V**]

Description

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

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

If your Platform Process Manager Client and Platform Process Manager 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 variable `JS_HISTORY_LIMIT` in `js.conf`.

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 Platform Process Manager Server
- flowdef-displays historical information about one or more flow definitions
- flow-displays historical information about one or more flows
- job-displays historical information about one or more jobs or job arrays

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

-u *user_name*

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

-t *start_time*,*end_time*

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

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

The time interval can be specified in many ways.

-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 Platform Process Manager release version to `stderr` and exits.

Usage

-C alarm

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

-C calendar

Displays the times when calendars are added or deleted.

-C daemon

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

-C flowdef

Displays information about when a flow definition state is:

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

-C flow

Displays information about when a flow state is:

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

-C job

Displays information about when a job or job array is:

- Started
- Killed
- Suspended
- Resumed
- Finished

Time interval format

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

Specify the time interval is follows:

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

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

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

Where:

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

start_time*,*end_time

Specifies both the start and end times of the interval.

***start_time*,**

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

,end_time

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

start_time

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

Absolute time examples

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

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

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

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

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

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

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

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

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

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

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

Relative time examples

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

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

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

Examples

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

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

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

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

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

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

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

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

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

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

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

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

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

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

jhold

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

Synopsis

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

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

Description

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

Synopsis

jid [**-h**|**-V**]

Description

You use the `jid` command to verify the connection between Platform Process Manager Client and Platform Process Manager Server. If the command returns the host name of Platform 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 Platform 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** | **-l** *flow_name*[**:***subflow_name*]**:***job_name*

Flow arrays in UNIX:

jjob [**-u** *user_name*] **-i** *flow_id* **-c** | **-k** | **-r** | **-l "***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 Platform 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.

-l

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

flow_name*:*subflow_name*:*manual_job_name

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

```
myflow:print:prtreport
```

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

Examples

jjob -i 42 -k payprt:report

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

See Also

`jmanuals`

jkill

kills a flow.

Synopsis

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

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

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

Description

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

Options

-u *user_name*

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

-f *flow_name*

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

flow_id

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

0

Specifies to kill all flows.

-h

Prints the command usage to `stderr` and exits.

-V

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

Examples

jkill -f myflow

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

jmanuals

displays all manual jobs that have not yet been completed.

Synopsis

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

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

Description

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

Options

-i *flow_ID*

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

-u *user_name*

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

-f *flow_definition*

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

-r yes

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

-r no

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

-h

Prints the command usage to `stderr` and exits.

-V

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

See also

`jcomplete`

jpublish

publishes a target flow to Platform Process Manager.

Synopsis

jpublish [**-u** *user_name*] [**-f** *flow_name*] **jpublish** [**-h**]|[**-V**]

Description

You use the `jpublish` command to publish a target flow to Platform Process Manager. Dynamic subflows and flow arrays can only refer to published target flows.

Only Platform 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 Platform 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 (either 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) 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, or whenever you modify any user groups specified in the JS_ADMINS or JS_CONTROL_ADMINS parameters.

You must be a Platform Process Manager administrator account to use this command.

Options

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Platform 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 Platform Process Manager Server. You must be a Platform Process Manager administrator to use this command.

Options

-h

Prints the command usage to `stderr` and exits.

-V

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

Options

-u *user_name*

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

flow_name

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

-h

Prints the command usage to `stderr` and exits.

-V

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

Examples

jrelease myflow

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

jrelease -u "user01" payupdt

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

See also

`jhold`

jremove

removes a previously submitted flow definition from Platform Process Manager.

Synopsis

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

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

Description

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

Options

-u *user_name*

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

-f

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

flow_name

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

-h

Prints the command usage to `stderr` and exits.

-V

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

Examples

jremove myflow

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

jremove -u "user01" payupdt

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

See also

`jsub, jhold`

jrerun

reruns an exited flow.

Synopsis

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

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

Description

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

jsetvars

sets values for variables during the runtime of a flow.

Synopsis

jsetvars -i *flow_ID* **-s** [*scope_1*]**:***variable_1a***=***value_1a* [**;***variable_1b***=***value_1b* ...] [[*scope_2*]**:***variable_2a***=***value_2a* [**;***variable_2b***=***value_2b* ...] ...] **jsetvars -i** *flow_ID* **-r** [*scope_1*]**:***variable_1a* [*variable_1b* ...] [[*scope_2*]**:***variable_2a* [*variable_2b* ...] ...] **jsetvars -i** *flow_ID* **-l** [*scope_1*[**;***scope_2* ...]] **jsetvars** [**-g**] **-s** [*scope_1*]**:***variable_1a***=***value_1a* [**;***variable_1b***=***value_1b* ...] [[*scope_2*]**:***variable_2a***=***value_2a* [**;***variable_2b***=***value_2b* ...] ...] **jsetvars** [**-g**] **-r** [*scope_1*]**:***variable_1a* [*variable_1b* ...] [[*scope_2*]**:***variable_2a* [*variable_2b* ...] ...] **jsetvars** [**-g**] **-l** [*scope_1*[**;***scope_2* ...]] **jsetvars** [**-h**]|[**-V**]

Description

You 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

-i *flow_ID*

Specifies the ID of the flow in which to take action.

-g

Specifies that the action is to take place on global variables. The `-g` option is assumed if **-i *flow_ID*** is not specified,

scope_n

Specifies the name of the flow indicating the scope for the following variables. If unspecified, this defaults to the main flow scope. You can combine variables of the same scope together and specify multiple scope levels.

variable_nx

Specifies the name of the variable you are setting.

value_nx

Specifies the value to which you will set the specified variable.

-s

Adds new or edits existing variables

-r

Removes existing variables

-l

Lists all variables.

-h

Prints the command usage to `stderr` and exits.

-V

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

Examples

jsetvars -i 1234 priority=10

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

jsetvars -g -s date=05-09-2007

Sets the `date` global variable value to 05-09-2007. If the `date` variable already exists, this changes the value of the `date` variable, otherwise, this adds a new variable called `date`).

jsetvars -i 1234 -r time

Deletes the `time` variable from the flow with the ID 1234.

jsetvars -i 21 -s mainvar1=123;mainvar2=456 mainvarX=zzz MF:SF1:myvar1=abc;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, sets the `myvar1` and `myvar2` variables at the subflow level (specifically, the `MF:SF1` subflow), and sets the `svar2` variable 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.

jsetvars -i 212 -s 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)`).

jsetvars -i 21 -l MF:SF1

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

jsetvars -i 21 -r mainvar MF:SF1:myvar1;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 thjis 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 Platform 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 Platform Process Manager installation and configuration script

Synopsis

jsinstall -f install.config

jsinstall -h

Description

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

Before installing and configuring Platform 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 Platform Process Manager.

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

jstop

suspends a running flow.

Synopsis

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

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

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

Description

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

Options

-u *user_name*

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

-f *flow_name*

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

flow_id

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

0

Specifies to suspend all flows.

-h

Prints the command usage to `stderr` and exits.

-V

Prints the Platform 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 Platform Process Manager.

Synopsis

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

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

Description

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

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

Options

-H

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

-r

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

-d

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

-m "*ver_comment*"

Submit the flow with version comments. `jsub` returns a flow version number after each successful submission.

-T *time_event*

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

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

cal_name

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

hour*:*minute

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

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

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

%duration

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

-F "*file_event*"

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

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

Note:

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

Specify the file event in one of the following formats:

arrival(*file_location*)

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

```
absolute_directory/filename
```

exist(*file_location*)

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

```
absolute_directory/filename
```

! exist(*file_location*)

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

```
absolute_directory/filename
```

size(*file_location*) *operator size*

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

```
absolute_directory/filename
```

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

Note:

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

Specify the size in bytes.

age(*file_location*) *operator age*

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

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

```
absolute_directory/filename
```

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

Note:

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

Specify the age in minutes.

-p "*proxy_event*"

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

Specify the proxy event in one the following formats:

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

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

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

Note:

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

Example: on successful completion of J1:

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

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

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

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

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

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

Example: on successful completion of all jobs in Array1:

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

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

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

Example: if more than 3 jobs in A1 exit:

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

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

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

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

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

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

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

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

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

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

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

-f "*flow_event*"

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

Specify the flow event in one of the following formats:

done(*flow_definition_name*)

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

user_name:*flow_definition*

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

end(*flow_definition_name*)

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

user_name:*flow_definition*

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

numdone(*flow_definition_name*) *operator nn*

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

user_name:*flow_definition*

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

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

For example:

numdone(jdoe:payflow)>=5

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

numstart(*flow_definition_name*) *operator nn*

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

user_name:*flow_definition*

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

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

numexit(*flow_definition_name*) *operator nn*

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

user_name:*flow_definition*

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

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

For example:

numexit(jdoe:payflow)>=3

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

exit(*flow_definition_name*) *operator nn*

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

user_name:flow_definition

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

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

For example:

exit(jdoe:payflow)>=2

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

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

-C *combination_type*

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

AND

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

OR

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

flow_file_name

Specifies the name of the file containing the flow definition.

-h

Prints the command usage to `stderr` and exits.

-V

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

Examples

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

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

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

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

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

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

- The size of `/data/tmp.log` exceeds 3.5 MB
- The `file /tmp/1.dat` arrives

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

jtrigger

manually triggers a previously submitted flow definition.

Synopsis

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

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

Description

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

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

Options

-u *user_name*

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

-v *var=value*

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

flow_name

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

-h

Prints the command usage to `stderr` and exits.

-V

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

Examples

```
jtrigger myflow
```

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

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

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

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

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

See also

`jrun`

junpublish

unpublishes a target flow from Platform 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 Platform Process Manager. Unplublished target flows can no longer be referred to by dynamic subflows and flow arrays.

Only Platform 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 Platform 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`

CHAPTER

6

Files

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

File Structure

When Platform 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 Platform Process Manager server is started, rather than immediately after installation.

Files created on the server host

The directories on the left are those that exist on UNIX after the Platform Process Manager server has been started. The directories on the right are those that exist on a Windows server after installation is complete:

The following describes what each directory contains:

Directory	Contents
<version>/app	Contains the files required to run Platform Process Manager Client.
<version>/bin	Contains the executables for all of the Platform Process Manager commands and the Platform Process Manager Client applications.
<version>/etc	Contains the Platform Process Manager messages and the data specification used by the Platform 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 Platform Process Manager README file and `install.config` and other installation-specific information.
`<version>`/lib	Contains the Platform Process Manager Java files.
`<version>`/resources	Contains the properties files used by Platform Process Manager.
<version>/man	On UNIX only, contains the man pages for each of the Platform Process Manager commands.
<version>/*platform*	Contains files specifically for running the Platform Process Manager software on each platform. In the above example, the files are for running the Platform Process Manager software on Solaris 7 and 8.
conf	Contains the configuration files used by the install script to define the Platform 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 Platform Process Manager to store Platform Process Manager Server and failover error logs. Platform Process Manager creates a log file called `jfd.log.`*hostname*, which contains the error logs.

Directory	Contents
work	Contains working information required by Platform Process Manager to complete its processing, including the following directories: • `alarms`—contains all alarm definitions • `calendar`—contains all system calendar definitions • `events`—contains persisted flow events and manual jobs • `history`—contains all historical data • `lock`—contains lock files to prevent multiple Platform Process Manager Servers from accessing the same working files • `storage`—contains copies of active and completed flows • `system`—contains system status data used byPlatform Process Manager Server during recovery • `templates`—contains templates for inserting custom applications in a flow • `var_comm`—contains temporary values for user variables • `variable`—contains the current values of any global or local user variables • `proxy_storage`—contains persisted proxy event definitions

Platform Process Manager history files

The log files containing Platform Process Manager audit data are located in *JS_TOP*`/work/history`. Platform Process Manager writes audit data to a history file called `history.log.1`. When the file reaches the maximum size specified in the configuration file `js.conf` (the default is 500 KB), a new file is created, and the suffix is incremented by 1. Periodically, you may want to manually archive or delete these files.

Platform Process Manager log files

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

history.log

Platform 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 Platform Process Manager system, is submitted to LSF master 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 master host. The second line indicates when LSF master 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

Platform Process Manager configuration file for installation on UNIX or Linux. Run `jsinstall -f install.config` to install Platform 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 Platform Process Manager installation script tar file. Edit the file to specify the options for your Platform 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 Platform Process Manager. The first entry is the primary Platform 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.

To specify a list, separate the names with a space. 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_ADMINS=DOMAIN\lsfadmin,"DOMAIN\Engineering Group",DOMAIN\userA

Default

There is no default for this parameter. A value for the primary Platform 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 Platform 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, UNIX user group names, or Windows active driectory user group names.

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

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 Platform 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 Platform Process Manager Server will run if the primary Platform Process Manager Server host is unavailable.

Default

The default is the same hostname as that specified for Platform 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 Platform 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 Platform 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 Platform Process Manager makes of the `license.dat` file.

Default

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

JS_MAILHOST

Syntax

JS_MAILHOST=*hostname*

Description

OPTIONAL.

Specifies the name of the mail server host.

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

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

Default

If Platform Process Manager Server is installed on Windows, the default is Exchange:*localhostname*. If Platform 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 Platform Process Manager Client to connect with Platform 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 Platform 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 master 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 Platform 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 Platform Process Manager. Platform Process Manager Server receives its configuration information on startup from its configuration file `js.conf`. The file `js.conf` is created automatically during the installation of Platform Process Manager. The values in `js.conf` are set automatically when you install Platform 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 Platform 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 Platform Process Manager. The first entry is the primary Platform 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.

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 Platform 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 Platform Process Manager administrators.

If the membership in a user group changes, the list of Platform Process Manager administrators needs updating. This parameter specifies the interval of time between scheduled updates. You can also manually update the list of Platform 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 Platform Process Manager forcefully terminates it.

Default

The default is 180 seconds.

JS_CONN_TIMEOUT

Syntax

JS_CONN_TIMEOUT=*seconds*

Description

Specifies the maximum number of seconds a Platform Process Manager Client waits for a response from Platform 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 Platform Process Manager, regardless of who the owner is. These administrators cannot submit or remove flows belonging to other users.

Any administrators specified can be user IDs, 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_DATACAPTURE_TIME

Syntax

JS_DATACAPTURE_TIME="*cal_name***@***user_name***:***hour*[**:***minute*]**"**

Description

Periodically, Platform Process Manager Server interrupts its processing to take an image of the workload in Platform Process Manager, and saves it for recovery purposes. Depending on the amount of workload that passes through your server, recovery of Platform 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, Platform Process Manager Server does not submit new work. Ideally, schedule this activity at a time when Platform Process Manager is least busy. You may need to adjust this schedule to find the balance between frequency and duration of the process, to ensure server productivity.

Default

The default is Daily@Sys:0:0 (daily at midnight).

JS_DTD_DIR

Syntax

JS_DTD_DIR=*JS_HOME***/8.0/etc**

Description

DO NOT CHANGE THIS VALUE.

Specifies the directory containing the DTD files required by Platform Process Manager.

Default

The default is *JS_HOME*/`8.0/etc`

JS_ENCRYPTION

Syntax

JS_ENCRYPTION=true | **false**

Description

Specifies whether to encrypt communication between Platform Process Manager Server and Platform Process Manager Client. If you set this value to true, ensure that the strong encryption package is installed.

Default

The default is false—do not encrypt communication.

JS_EVENTS_LIFETIME

Syntax

JS_EVENTS_LIFETIME=*hours*

Description

Specifies the time period in hours for which event data is collected before a new event log file is created. If the size of the log file exceeds the file size specified in JS_EVENTS_SIZE, a new log file is created, regardless of how many hours of data it contains.

Default

The default is 168 hours (7 days).

See also

JS_EVENTS_DEFAULT_SIZE

JS_EVENTS_DEFAULT_SIZE

Syntax

JS_EVENTS_DEFAULT_SIZE=*bytes*

Description

Specifies the maximum number of bytes an event 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_EVENTS_LIFETIME, a new log file is created, regardless of its size.

Default

The default is 1000000 bytes (1 MB).

See also

JS_EVENTS_LIFETIME

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 Platform 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 Platform 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 Platform Process Manager Server will run if the primary Platform Process Manager Server host is unavailable.

Default

The default is the same host name as that specified for Platform Process Manager Server.

See also

JS_FAILOVER, JS_FOD_PORT

JS_FILEAGENT_SENSITIVITY

Syntax

JS_FILEAGENT_SENSITIVITY=*seconds*

Description

Specifies the time interval in seconds at which Platform 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 Platform 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_CLEAN_PERIOD

Syntax

JS_HISTORY_CLEAN_PERIOD=*days*

Description

Specifies the time period in days for which history log files are stored. Any history log files older than the specified time period is cleaned up by Platform Process Manager.

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 Platform Process Manager Client and Platform 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 Platform 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 Platform Process Manager to determine when to delete a copy of 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 Platform 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 Platform Process Manager working set.

This value takes effect when JS_IM_ACTIVEPOLICY = JF_IM_TPolicy. The oldest occurrence is deleted first.

Default

The default is 5 days.

See also

JS_IM_ACTIVEPOLICY, JS_IM_POLICY_CHECKING_INTERVAL, JS_IM_POLICY_NOOFFLOWS

JS_IM_POLICY_NOOFFLOWS

Syntax

JS_IM_POLICY_NOOFFLOWS=*number*

Description

Specifies the number of copies of a completed flow that are retained within the Platform Process Manager working set. Specify a number greater than 0.

This value takes effect when JS_IM_ACTIVEPOLICY = JF_IM_IPolicy. The oldest occurrence is deleted first.

Default

The default is 36 copies.

See also

JS_IM_ACTIVEPOLICY, JS_IM_POLICY_LIFETIME, JS_IM_POLICY_CHECKING_INTERVAL

JS_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 Platform 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_LICENSE_FILE

Syntax

JS_LICENSE_FILE=/*path/filename*

Description

DO NOT CHANGE THIS VALUE.

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

Default

The default is *JS_HOME*`/conf`.

JS_LIMIT_USER_VIEW

Syntax

JS_LIMIT_USER_VIEW=true | **false**

Description

Specifies whether a user's view of flows is limited to their own flows, or includes all flows in Platform Process Manager. For a guest user, limits the access so that no flows are viewable.

Default

The default is false.

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 administratos can modify global variables.

Default

The default is true.

JS_LOCAL_EXECUTION_THREADS

Syntax

JS_LOCAL_EXECUTION_THREADS=*number_of_jobs*

Description

Specifies the maximum number of local jobs that may be run in parallel independent of flows. If you set this to be lower than the default value, Platform Process Manager uses the default value.

Default

The default is one job.

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 Platform Process Manager forcefully terminates the job. If you set this to be zero or less, Platform Process Manager uses the default value.

Default

The default is 180 seconds.

JS_LOGDIR

Syntax

JS_LOGDIR=/*path*

Description

Specifies the name of the directory containing the `jfd.log` file, the error log file for the Platform 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 Platform Process Manager. Set as true if you want to require users to log in before using Platform Process Manager.

Default

The default is false; users do not have to log in to use Platform Process Manager.

JS_LOGON_RETRY

Syntax

JS_LOGON_RETRY=*number*

Description

Specifies the number of times Platform 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 Platform Technical Support. Valid values from highest to lowest are:

- LOG_EMERG
- LOG_ALERT
- LOG_CRIT
- LOG_ERR
- LOG_WARNING
- LOG_NOTICE
- LOG_INFO
- LOG_DEBUG
- LOG_DEBUG1
- LOG_DEBUG2
- LOG_DEBUG3

The level specified by the log mask determines which messages are recorded and which are discarded. All messages logged at the specified level or higher are recorded, while lower level messages are discarded.

For debugging purposes, the level LOG_DEBUG contains the fewest number of debugging messages and is used for basic debugging. The level LOG_DEBUG3 records all debugging messages, and can cause log files to grow very large; it is not often used. Most debugging is done at the level LOG_DEBUG2.

Default

The default is JS_LOG_MASK=LOG_NOTICE.

JS_MAILHOST

Syntax

JS_MAILHOST=[**SMTP:** | **Exchange:**]*hostname*

Description

OPTIONAL.

Specifies the name of the mail server host.

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

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

Default

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

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

10 substitutions

JS_MAX_VAR_SUBSTITUTIONS

Syntax

JS_MAX_VAR_SUBSTITUTIONS=*number*

Description

OPTIONAL.

Specifies the maximum number of variable substitutions that can be performed in a single job definition field.

Default

10 substitutions

JS_MULTI_INSTANCE

Syntax

JS_MULTI_INSTANCE=true | **false**

Description

Specifies whether there will be multiple instances of the Platform Process Manager daemon (`jfd`) running in Platform LSF clusters.

If set to false, the Platform Process Manager daemon checks out the number of licenses equal to the number of cores in the Platform Process Manager server.

If set to true, the Platform Process Manager daemon checks out one licenses for each instance. This allows the Platform Process Manager servers to run multiple instances of the Platform Process Manager daemon, up to the number of cores in the Platform Process Manager server.

Default

The default is false.

JS_PORT

Syntax

JS_PORT=*number*

Description

REQUIRED.

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

Default

The default port number is 1966.

See also

JS_HOST

JS_PROXY_DURATION

Syntax

JS_PROXY_DURATION=*minutes*

Description

Specifies the length of time for which to publish events that occur in Platform Process Manager, keeping the event information available for flows that contain proxies looking for that event. This is required if the event can occur before the flow looking for it requires it.

Default

The default is 0.

JS_REALTIME_UPDATE

Syntax

JS_REALTIME_UPDATE=true | **false**

Description

Specifies whether or not to enable real-time updates to the data displayed in the Flow Manager. When enabled, the status of work items in the Flow Manager updates automatically as a change occurs. Users can choose real-time updates, automatic refreshes at a specified time interval, or manual refreshes. If you disable this option, and a user has selected real-time updates, the client updates automatically at the specified refresh interval instead.

Default

The default is false.

JS_REALTIME_OBJECT_URL

Syntax

JS_REALTIME_OBJECT_URL=*url*

Description

Required when JS_REALTIME_UPDATE is set to true. Specifies the url to the JMS (Java Message Service), used by Platform Process Manager Server when obtaining status updates. This url must match the url specified when configuring the JMS broker—IMQ_JNDI_URL.

Default

There is no default for this parameter.

JS_SERVICE_STOP_PEND_WAIT

Syntax

JS_SERVICE_STOP_PEND_WAIT=*milliseconds*

Description

Windows only.

Specifies the amount of time that the Platform 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 Platform 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_START_RETRY

Syntax

JS_START_RETRY=*retries*

Description

Specifies the maximum number of times Platform Process Manager tries again to start a job or job array before raising a Start Failed exception.

Default

The default is 20 times.

JS_SU_NEW_LOGIN

Syntax

JS_SU_NEW_LOGIN=true | **false**

Description

Specifies whether or not to start a new login shell when Platform 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

The default is true.

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_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 Platform 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. The file name is the name of the alarm and the file type is alarm.

Format

Each alarm file has the following format:

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

Example

The following example shows a database failure alarm definition. The alarm is called `DBMSfail.alarm`. It's contents are:

```
DESCRIPTION=Send DBA a message indicating DBMS failure
NOTIFICATION=Email[bsmith ajones]
```

Index

G

H

I

J

L

M

N

O

P

R

S

U

V

W