
Using Platform Process Manager

Platform Process Manager
Version 8.0.2
November 2011



Copyright

© 1994-2011 Platform Computing Corporation.

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

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

We'd like to hear from you

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

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

Document redistribution and translation

This document is protected by copyright and you may not redistribute or translate it into another language, in part or in whole.

Internal redistribution

You may only redistribute this document internally within your organization (for example, on an intranet) provided that you continue to check the Platform Web site for updates and update your version of the documentation. You may not make it available to your organization over the Internet.

Trademarks

LSF is a registered trademark of Platform Computing Corporation in the United States and in other jurisdictions.

ACCELERATING INTELLIGENCE, PLATFORM COMPUTING, PLATFORM SYMPHONY, PLATFORM JOB SCHEDULER, PLATFORM ISF, PLATFORM ENTERPRISE GRID ORCHESTRATOR, PLATFORM EGO, and the PLATFORM and PLATFORM LSF logos are trademarks of Platform Computing Corporation in the United States and in other jurisdictions.

UNIX is a registered trademark of The Open Group in the United States and in other jurisdictions.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft is either a registered trademark or a trademark of Microsoft Corporation in the United States and/or other countries.

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

Intel, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other products or services mentioned in this document are identified by the trademarks or service marks of their respective owners.

Third-party license agreements

<http://www.platform.com/Company/third.part.license.htm>

Contents

1	New Features in Platform Process Manager 8.0.2	7
	General new features	8
	New features in Flow Manager	10
	New features in Flow Editor	12
	Flow-related new features available only in Platform Application Center	16
2	Introduction to Platform Process Manager	19
	About Platform Process Manager	20
	About Platform Process Manager terms	22
	Change your server	27
	About flow definitions and flows	29
3	Platform Process Manager Calendars	33
	About the calendar editor	35
	Create a calendar with specific dates	37
	Create a calendar using an expression	38
	Create a calendar with a complex expression	41
	Calendar examples	43
	Edit an existing calendar	45
	Delete a calendar	46
4	Define your flow	47
	Ways to create a flow definition	48
	Using the example flows	49
	Create a flow diagram	50
	Include a job array in the flow diagram	52
	Include a job submission script in the flow diagram	55
	Include a job array submission script in the flow diagram	56
	Include a static subflow in the flow diagram	58
	Include a static flow array in the flow diagram	59
	Include a dynamic subflow in the flow diagram	61
	Include a dynamic flow array in the flow diagram	62
	Include a manual job in the flow diagram	64
	Specifying custom exit codes for successful job completion	66
	Include a local job in the flow diagram	67
	Variables in Platform Process Manager	69
	Job dependencies	83
	Specify dependency on the start or submission of specific jobs	86

	Specify a dependency on a file	90
	Change the label displayed for an event	93
	Dependency on a date and time	94
	Specify dependencies on a job array	96
	Specify dependencies on a subflow	98
	Specify dependencies on an unconnected work item	100
	Specifying multiple dependencies	105
	Details of a job	106
	About flow completion attributes	120
	Specify flow completion attributes	122
	Configuring flow exit codes	126
	Specify exception handling for a flow	128
	Flow attributes	129
	Specify flow attributes	130
	Save the flow definition	132
	Loop a flow or subflow	133
5	About Platform Process Manager exceptions	135
	About exception handling	139
	Handling exceptions	143
	Alarms	148
6	Run your flow	153
	Create a flow definition to be triggered manually	155
	Schedule your flow	157
	Run a flow when another flow...	162
	Run your flow once	166
	Submit your flow definition	167
7	Control a Flow	169
	About the Flow Manager	170
	Real-time data	173
	Print data	174
	Filter the data displayed in the tree view	175
	Trigger a flow	178
	View a flow definition and specify versioning options	179
	View inter-flow relationships	181
	Determine the status of jobs in a flow	185
	Manually complete a dependency	187
	Kill a running job	188
	Stop a flow at a specific point by putting a job on hold	189
	Run or rerun a single job	190
	Mark a job complete	191
	Work with manual jobs	192
	Completing manual jobs with exit codes	194
	Work with proxies	195

	Kill a running flow	197
	Suspend a running flow	198
	Resume a suspended flow	199
	Rerun an exited flow	200
	Rerun a flow while a job is still running	202
	Rerun an exited job array	203
	Hold a flow definition	204
	Releasing a flow definition from hold	205
	Remove a flow definition	206
8	Mainframe support	207
	Using mainframe	208
	Exit codes	211
9	Commands	213
	caeditor	215
	floweditor	216
	flowmanager	217
	jadmin	218
	jalarms	220
	jcadd	223
	jcal	228
	jcdel	229
	jcm	230
	jcomplete	234
	jdefs	236
	jflows	238
	jhist	240
	jhold	245
	jid	246
	jjob	247
	jkill	250
	jmanuals	252
	jpublish	253
	jreconfigalarm	254
	jrelease	255
	jremove	256
	jrerun	258
	jresume	259
	jrun	261
	jsetvars	262
	jsetversion	264
	jsinstall	265
	jstop	266
	jsub	268
	jtrigger	275

junpublish 277

New Features in Platform Process Manager

8.0.2

This chapter provides a summary of new features available in this version. Some new features available in Process Manager are also available or visible in Platform Application Center.

General new features

Enhancements to local jobs on Linux and UNIX

This feature is available in:

- Platform Process Manager

Description:

These enhancements to local jobs only apply to Linux and UNIX.

A local job is a job that will execute immediately on the Platform Process Manager host without going through LSF. A local job is usually a short and small job.

Enhancements that have been made:

- Local jobs are now non-blocking. This means that multiple local jobs can run at the same time.
- You can now kill a local job. If a local job is killed outside of Process Manager, Process Manager can identify the local job's exit status and resource usage.
- Local jobs are now suspended and resumed when you suspend or resume the flow that contains them.
- In the job's runtime attributes, you can now view the exit status and CPU usage of a local job after the job completes. The process ID identifies the local job and you can view CPU usage for the job. You can also view the process ID of the job and CPU usage information with `jobs -l flow_id` and `jobs -C job`.
- To avoid overloading the Platform Process Manager host with too many local jobs, there is a new parameter `JS_LOCAL_JOBS_LIMIT` in `js.conf` to control the maximum number of local jobs that can simultaneously run on the Process Manager host.
- By default, local jobs now have no timeout. The default value of `JS_LOCAL_EXECUTION_TIMEOUT` in `js.conf` has been changed to unlimited.
- The parameter `JS_LOCAL_EXECUTION_THREADS` in `js.conf` is now obsolete. Its value is now fixed at 1 and cannot be changed, as local jobs are now non-blocking.
- Should `jfd` terminate abnormally, when it restarts it can recover running and finished local jobs and determine their status and resource usage.
- A new binary is installed in `JS_SERVERDIR:eem.local`. It is started by `jfd` and handles job submission, control, and status checking for local jobs and reports back to `jfd`.
- Two additional port numbers are now used by `jfd` and `eem.local`: `JS_PORT + 1` and `JS_PORT + 2`.

New built-in user variable JS_FLOW_FULL_NAME

This feature is available in: Platform Process Manager and Platform Application Center.

Description:

You use the built-in user variable `JS_FLOW_FULL_NAME` when you need to use the long version of a subflow name.

For example:

- For a subflow named `11:usr1:F1:SF1:SSF1`, this variable is set to `11:usr1:F1:SF1:SSF1`.
- For a main flow named `11:usr1:F1`, this variable is set to `11:usr1:F1`.

Use a custom mail program to send email

This feature is available in:

- Platform Process Manager: Set `JS_MAILPROG` in `js.conf` to your custom mail program. After setting your custom mail program, you will need to restart `jfd` with the commands `jadmin start` and `jadmin stop` to make changes take effect.

Description:

By default, Process Manager sends email through `/usr/lib/sendmail` on UNIX or `lsmail.exe` on Windows.

You can now specify a custom mail program to send emails. Your custom mail program can be a shell script, a binary executable, or, a `.bat` file on Windows. Your custom mail program must follow the same protocol as `sendmail`.

Restrict who can see the flow chart view

This feature is available in:

- Platform Process Manager: You set the parameter `JS_LIMIT_FLOW_CHART_VIEW` in `js.conf` and affects display of the flow chart and associated actions in Flow Manager and Platform Application Center.
- Platform Application Center: Flow Chart view is restricted along with associated actions based on the parameter set in Platform Process Manager.

Description:

There is a new parameter in `js.conf`, `JS_LIMIT_FLOW_CHART_VIEW`. This parameter allows you to restrict viewing the chart view of a flow and flow definition to only the Process Manager administrator and users who are both the flow definition owner and flow owner.

When this parameter is set to `false`, users who can view a flow or flow definition, can see everything about the flow: flow chart, general information, subflows and jobs, flow data, and flow history. These users can also perform job and subflow-specific actions.

When this parameter is set to `true`, there are restrictions on which users can see the flow chart of a flow and flow definition and associated actions the user can take on components of the flow.

New features in Flow Manager

Hold and release for jobs

This feature is available in:

- **Flow Manager:** You can hold and release jobs through Flow Manager By State tab, display a flow, and select the job in the Waiting state, right-click and choose Hold, or the new options in the `jj ob` command, `- p` for hold, and `- g` for release.
- **Platform Application Center:** Go to Jobs > Jobs > By State > Running, select the flow, select the Flow Chart tab, select the job in the Waiting state, right-click, and choose Hold.

Description:

In some cases, you may want to stop a flow at a specific point so that you can fix problems. You can do this by putting a job in the Waiting state in the flow on hold.

Only the branch of the flow that contains the job that is On Hold pauses. Other branches of the flow continue to run.

You can put on hold LSF jobs, job submission scripts, local jobs and job arrays.

Allow users to trigger other users' flows

This feature is available in:

- **Platform Process Manager:** Set `JS_CHANGE_FLOW_OWNER` in `js.conf`. There is also now one more tab in Flow Manager, the By Definition tab. This tab displays flow definitions organized by the user who submitted them. In addition, what is displayed in the tree view has been enhanced for all tabs to indicate the flow owner and flow submitter.
- **Platform Application Center:** The parameter setting in Process Manager also controls who can trigger other users' flows in Platform Application Center. Select Jobs > Flow Definitions > By User to trigger flows.

Description:

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

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

With the new parameter `JS_CHANGE_FLOW_OWNER=true` in `js.conf`, non-administrator users can trigger other users' flows. In this way, one user can submit flow definitions, and another user can trigger the flow from the flow definition, own the flow, and control it. The user who submitted the flow definition is the owner of the flow definition, the user who triggered the flow is the owner of the flow.

Rerun a flow while a job is still running

This feature is available in:

- **Platform Process Manager:** In Flow Manager, the Rerun Now and Rerun with variables menu items have been replaced with Rerun, and a window is displayed in which you can choose what to rerun in the flow.
- **Platform Application Center:** Select Jobs > Jobs > By State, select the flow, and click the Rerun button to display a window in which you can choose what to rerun in the flow.

Description:

In previous versions, you could only rerun flows that were in an Exited state. You can now rerun flows when the flow state is Running, Exited, or Done.

This is useful for flows that have several branches. When one branch fails, you can rerun the branch without waiting for other branches of the flow to complete.

You can:

- Set or unset starting points when there are still jobs running in the flow.
- Choose whether to rerun the flow from:
 - Starting points and exited jobs. The flow will rerun from any starting points, exited jobs, and, from the item following any manually completed jobs provided dependencies are met.
 - Starting points only. The flow will rerun only from starting points.

Note that you can only rerun a running flow if the part of the flow to be rerun does not overlap with items that are currently running.

Exit codes for manual jobs

This feature is available in:

- Platform Process Manager: In Flow Manager, you can now specify exit codes when completing a manual job, or by using the new option in the `j complete` command, `-e exit_code`. Manual jobs can now fail. In Flow Editor, you can now specify in the manual Job Event Definition the dependencies Fails, Ends with any exit code, and Ends with exit code....
- Platform Application Center: You can complete a manual job and specify an exit code through Jobs > Jobs > By State > Pending User Input, select the manual job, click the Complete Manual Job button.

New features in Flow Editor

New Other Options field for additional LSF job submission options

This feature is available in:

- Flow Editor: Job Definition or Job Array Definition dialog, Advanced tab, Other Options field.
- Platform Application Center: In the Job Definition or Job Array Definition dialog, Advanced tab, Other Options field. To access the Job Definition, in the Jobs tab, select Jobs, select a Flow, select the Flow Chart tab, right-click a job to display the Job Definition.

Description:

This allows you to use options that are not available from the job definition dialog. The options you specify are added to the bsub command when you submit the job or job array.

For example:

```
-w "done("#{JS_FLOW_FULL_NAME}:JobArray1")"
```

You can also specify user variables in the Other Options field.

Configure custom exit codes for successful jobs

This feature is available in:

- Platform Process Manager: In Flow Editor, open the Job Definition dialog, Job Script Definition dialog, Manual Job Definition dialog, or Local Job Definition dialog, and configure the new field Non-zero success exit codes.
- Platform Application Center: You can view settings for the field Non-zero success exit codes in the Job Definition, but you cannot change them.

Description:

By default, for a job to complete successfully, the exit code must be 0. Any other exit code indicates the job failed.

In some cases, however, you may want to use exit codes to pass information to subsequent work items and may want to use numbers other than 0 to indicate success.

You can now do so by specifying a space-separated list of exit codes in the Job Definition dialog, Job Script Definition dialog, Manual Job Definition dialog, or Local Job Definition dialog, with the new Non-zero success exit codes field.

Configure how to calculate flow exit codes

This feature is available in:

- Platform Process Manager: In Flow Editor, select Action > Specify Flow Completion Attributes, new section Determine the flow exit code from
- Platform Application Center: You can view Flow Completion Attributes but you cannot change them.

Description:

By default, a Done flow or subflow has an exit code of 0, since the default way that Process Manager determines the flow exit code is through the sum of all exit codes of all work items in the flow.

However, it is possible to specify custom success exit codes for LSF jobs, job scripts, local jobs, and manual jobs. For this case, you can configure the flow to inherit the exit code of the last item that was successfully completed or that failed in the Flow Completion Attributes dialog.

New dependencies

This feature is available in:

- Platform Process Manager: In Flow Editor, new dependencies have been added for subflows, flow arrays, and jobs.

Flow Event Definition, for subflows:

- The flow completes successfully with exit code...
- The flow fails with exit code...
- The flow fails

Flow Array Event Definition:

- Any flow fails

Job Event Definition:

- Fails
- Is Submitted

Job Array Event Definition:

- Any job fails
- Platform Application Center: You can view dependency settings but you cannot change them.

User variables in more fields when defining jobs and job arrays

This feature is available in:

- Platform Process Manager: In Flow Editor, Job Definition and Job Array Definition dialogs.
- Platform Application Center: User variables are displayed, but cannot be specified.

Description:

You can now use user variables in more fields in the Job Definition and Job Array Definition dialogs. When you select a field and hover, the help that displays indicates whether you can use a user variable or not in the field.

User variables for job parameters are resolved at runtime, just before the job is submitted.

The following fields now support user variables:

Tab	Field
Processing tab	Number of Processors for Parallel Jobs, Minimum
	Number of Processors for Parallel Jobs, Maximum
	Before Execution, Run command
	User Group, Associate job with user group

Tab	Field
Limits tab	All fields under Job Limits
	Host Limits, Maximum run time
	Host Limits, Maximum CPU time

Submit a dependent job after selected jobs start running or are submitted

This feature is available in:

- Platform Process Manager

Description:

- In Flow Editor, Advanced tab, Pre-submit section, you can now select jobs upon the current job depends. This now applies not only to jobs and job scripts, but also to job arrays, job array scripts, and template jobs.
- You can now specify either Starts or Submitted as the dependency. In this way, you can identify that the current job is to be submitted right after the selected jobs have started to run in LSF, or that the current job is to be submitted right after the selected jobs have been submitted to LSF.
- Create proxy events for jobs with the new Starts or Is Submitted events
- Create proxy events for job arrays with the new Number of jobs started is..., and The job array is submitted events.

Static and dynamic flow arrays can now run sequentially

This feature is available in:

- Platform Process Manager: In Flow Editor, Flow Array Attributes dialog.

Description:

In Flow Editor, there is now an option in the Flow Array Attributes to run in parallel or sequentially. As a result, you now have the choice of running static or dynamic flow array elements in parallel, or sequentially. In previous versions, flow arrays always ran in parallel.

Determining success or failure based on specific exit codes in the dependency condition

This feature is available in:

- Platform Process Manager: In Flow Editor, Job Event Definition, Proxy Event Definition, and Exception Handler Definition with the events Ends with exit code equal to and Ends with Exit code not equal to.

Description:

You can now define dependencies to take action if any of the specified exit codes are encountered.

You can specify a list of exit codes in:

- Dependencies between jobs, job scripts, template jobs, local jobs, and manual jobs.

- Proxy event definitions for a proxy job, proxy template job, proxy job script, and proxy local job. For proxy dependencies, you can also use `j sub -p` and specify a list of exit codes.
- Exception Handler Definition for a job, job script, template job.

Command to run field can now display multiple lines

This feature is available in:

- Platform Process Manager: Flow Editor, in the definition of a job, job array, or local job.
- Platform Application Center: You can enter the command to run in Jobs > Submission Forms > Flow Forms. You can view a command that spans multiple lines in Jobs > By State, by selecting the state, selecting a flow, selecting the Flow Chart tab, right-clicking and selecting Open Definition for a job, job array, or local job.

Flow-related new features available only in Platform Application Center

Jobs and Flows can now be monitored in the same window

This feature is available in: Platform Application Center.

Description:

Jobs and flows are now in the same window, accessible through Jobs > Jobs > By State. There is now a Type column by which you can sort.

Possible types are:

- Job
- Flow
- Array

Completion attributes now visible for subflows and flow arrays in Flow Chart tab

This feature is available in: Platform Application Center.

Description:

You can now view completion attributes for static and dynamic subflows, and flow attributes and completion attributes for static and dynamic flow arrays.

For subflows, select Jobs > Submission Forms > Flow Forms, select a flow, select the Flow Chart tab, select a subflow, right-click and choose Completion Attributes.

For flow arrays, select Jobs > Submission Forms > Flow Forms, select a flow, select the Flow Chart tab, select the flow array, right-click and select Expand. When the new page is displayed, right-click on the page, and select the Attributes or Completion Attributes menu items.

Reorganization of pages for flow definitions

This feature is available in:

- Platform Application Center: Pages related to flow definitions have been reorganized.

Description:

- Resources > Submission Templates > Flow Definitions: view flow definitions as a list or graphically and perform actions on the flow definitions: Hold, Release, Remove, Publish, Unpublish.
- Settings > System Services > Flow Manager Service: View the Process Manager server name, port, and statistics about the number of flows and flow definitions in each state, and set global variables for all flows.
- Jobs > Submission Forms > Flow Forms by User: Trigger a flow from a flow definition. Non-administrator users can see their own submitted flow definitions and all published flow definitions.

Process Manager administrators and control administrators can see all submitted flow definitions and flows.

- Jobs > By State > Pending User Input: View and complete manual jobs.
- Jobs > Job Alerts: View open alarms in the system.

Introduction to Platform Process Manager

This section describes each of the component applications that make up the Platform Process Manager software, and introduces each of the work items used to define and schedule your workload.

About Platform Process Manager

Platform Process Manager comprises three client applications and a server application. The client applications are:

- Platform Process Manager Designer:
 - The Flow Editor
 - The Calendar Editor
- The Flow Manager

The Platform Process Manager Server is the scheduling interface between the client applications and the execution agent, Platform Process Manager.

The Flow Editor

You use the Flow Editor to define your 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. You also use the Flow Editor to submit your flow definitions—this places them under the control of Platform Process Manager.

You can submit a flow definition in three ways:

- By submitting it to be triggered when one or more events occur
- By submitting it to be triggered manually
- By running it immediately

After a flow definition is submitted, a copy of the definition resides in Platform Process Manager. If the flow definition is to be triggered by an event, Platform Process Manager triggers it automatically when that event occurs, creating a flow. If the flow definition is to be triggered manually, the flow definition waits in Platform Process Manager until you trigger it, creating a flow. If the flow definition is run immediately, Platform Process Manager does not store a copy of the definition—just the flow.

Using the Flow Editor, you can work with existing flow definitions, easily modifying them to create new ones. You can also create reusable flow definitions that can be shared by many users, or reused over and over again. These flow definitions can be easily incorporated into a new definition as subflows. These techniques allow you to create intricate work flows quickly, with fewer errors.

You start the Flow Editor from the Windows start menu, by selecting Platform Computing > Platform Process Manager > Flow Editor, or by running `fl owedi t` or on UNIX.

The Calendar Editor

You use the Calendar Editor to define calendars, which Platform Process Manager uses to calculate the dates 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. 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.

Once a calendar is defined, you associate a job or a flow definition with the calendar using a *time event*.

You start the Calendar Editor from the Windows start menu, by selecting Platform Computing > Platform Process Manager > Calendar Editor, or by running `cal edi t` or on UNIX.

The Flow Manager

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

Using the Flow Manager, you can view the status of, suspend, or kill a flow. While working within the Flow Manager, you can review the flow definition, while comparing it to the running flow.

You start the Flow Manager from the Windows start menu, by selecting Platform Computing > Platform Process Manager > Flow Manager, or by running `flowmanager` on UNIX.

About Platform Process Manager terms

Jobs



A *job* is a program or command that is scheduled to run in a specific environment. A job can have many attributes specifying its scheduling and execution requirements. You specify the attributes of the job when you define the job in the Flow Editor. Platform Process Manager schedules and manages jobs that run on Platform Process Manager hosts. Platform Process Manager uses job attributes, system resource information, and configuration settings to decide when, where, and how to run jobs. While each job is assigned a unique job name by the system, you can associate your own job names to make referencing easier.

Dependencies



A *dependency* describes the order in which something happens within a flow: a job (or job array or subflow) can depend on the completion of a job, job array, subflow, or event before it can run.

A dependency is shown in the Flow Editor and Flow Manager as a line with an arrow. The job at the tip of the arrow cannot run until the work item at the other end of the arrow reaches a particular condition.

A dependency is used to indicate relationships between jobs, events, alarms, and so on.

Job dependencies

A *job dependency* is a dependency that a job (or job array or subflow) has on the completion of a predecessor job. You can define a dependency that controls a job's execution upon the completion, failure, or startup of other jobs. You can also start a job when the predecessor fails with a specific exit code, or experiences a specific exception.

Job dependencies are shown in the Flow Editor and Flow Manager as a line with an arrow. The job at the tip of the arrow cannot run until its predecessor at the other end of the arrow reaches a particular condition. The default type of job dependency is on the successful completion of the predecessor.

Job arrays



A *job array* is a group of homogeneous jobs—jobs that share the same executable and resource requirements, but have different input files, for example `input 1`, `input 2`, `input 3` and so on. You can use a job array to submit, control and monitor all of the jobs as a single unit. Each job submitted from a job array shares the same job ID as the job array and is uniquely referenced using an array index. The dimension and structure of a job array is defined when the job array is created.

Job submission script



A *job submission script* is a shell script or a batch file, which you can define to submit a job. You can define and submit customized job array submission script with `bsub` command and options. You can monitor and control the jobs that have been submitted through the customized job submission scripts. You specify the attributes of the script when you define the job submission script in the Flow Editor. Platform Process Manager schedules and manages job submission scripts that run on the Platform Process Manager hosts.

Job array submission script



A *job array submission script* is a group of submission scripts—that share the same executable and resource requirements, but have different input files, for example `script 1`, `script 2`, `script 3`, and so on. You can use a customized job array submission script to control and monitor all the jobs and job arrays. Each job submitted from a job array submission script shares the same job ID as the job array submission script and is uniquely referenced using an array index. The dimension and structure of a job array submission script is defined when the job array submission script is created.

Manual jobs



A *manual job* is a place-holder in a flow—it marks the place in a process where some manual activity must take place before the flow can continue. Successors of a manual job cannot run until the manual job is explicitly completed.

Flow definitions

A *flow definition* is a container for a group of related jobs. The flow definition describes both the jobs and their relationships to each other, as well as any dependencies the jobs have on files or dates and times. Using a flow definition, you can create a complex schedule involving many jobs, and manipulate it as a single entity. You can also use a flow definition to group jobs together that form a particular function, and imbed the flow definition as a subflow within a larger flow definition. This allows you to share and reuse common functions.

Flow definitions can be stored locally on your own machine, or within a shared file system. You can see and import flow definitions created by another user, but you cannot control running flows owned by another user unless you have administrative authority.

Flows

A *flow* is the particular occurrence of a flow definition that is created when the flow definition is triggered. When Platform Process Manager creates a flow from the flow definition, it assigns each occurrence of the flow a unique ID called the *flow ID*.

Adhoc flows

An *adhoc flow* is a flow that is run directly by the Platform Process Manager Server without the server saving a copy of the flow definition. An adhoc flow is run directly from the Flow Editor.

An adhoc flow is displayed in the tree view of the Flow Manager in each of the following ways:

- In the By Flow User view, under the user ID of the user who runs it, in a folder called adhoc
- In the By Event view, in a folder called adhoc

Subflows



A *subflow* is simply a flow definition that has been imbedded within another flow definition. Using subflows within a flow is a simple method to share and reuse common routines.

Events

An *event* is a change or occurrence in the system (such as the creation of a specific file, a prior job completing with a particular exit code, or simply the arrival of a file at a particular date and time) that can be used to trigger a flow or one or more jobs within a flow. Platform Process Manager responds to the following types of events:

- Time events—points of time (defined by calendars and time expressions) that can be used to trigger the scheduling of jobs
- File events—changes in a file's status
- Proxy events—events used to represent another flow or a work item that runs within another flow
- Link events—events used to consolidate the output of other events

Time events



You use *time events* in Platform Process Manager to make something happen at a specific time. You can use a time event to specify the frequency at which a repetitive job repeats, to prevent a job from running until a particular time, or to specify when to start running a flow.

You cannot create a time event without referencing a calendar, which provides the date or dates on which the time event is valid, allowing it to trigger.

You create time events using the Flow Editor.

File events



You use *file events* to make something happen when a file reaches a particular state. You can use a file event to trigger a flow, job or subflow: when a file arrives; when a file reaches a certain size; if a certain file exists; or any combination of these conditions.

You create file events using the Flow Editor.

Proxy events



You use *proxy events* to represent work items that run within another flow, or to represent another flow. You can create a dependency on the success or failure of a proxy event. You can use a proxy event to trigger a flow, or to trigger a work item within a flow.

Link events



You use *link events* to combine multiple dependencies into a single point in a flow diagram. You can use link events to run a job when multiple jobs complete, or you can use them to run a subflow when one of a group of jobs complete. For example, you can use an *AND* link event to trigger a job when all of a group of conditions are met, or you can use an *OR* link event to trigger a job when any one or more of a group of conditions is met.

You create link events using the Flow Editor.

Calendars

A *calendar* consists of a sequence of days on which the calendar is considered valid. A job is scheduled when the calendar is valid and a time of day specification is met. Calendars are defined and manipulated independently of jobs so that multiple jobs and flows can share the same calendar. Each user can maintain a private set of calendars, or use the calendars defined as system calendars. If a calendar is changed, any jobs associated with the calendar will automatically run according to the new definition. Calendars are stored within Platform Process Manager's private storage, and cannot be stored locally or edited outside of the Calendar Editor.

Exceptions

An *exception* is a specific error condition that is detected when a job does not process as expected. Platform Process Manager detects several of these conditions.

Exception Handlers

An *exception handler* is a function used to respond when an exception occurs. You can use jobs or flows as exception handlers, or you can use Platform Process Manager's built-in exception handlers:

- Kill
- Rerun
- Alarms

Kill

You can automatically kill a job, flow, or subflow if it experiences the specified exception.

Rerun

You can automatically rerun a job, flow, or subflow if it experiences the specified exception.

Alarms



An alarm is a type of built-in exception handler, used to send an email notification to key personnel or execute a script to show that an error has occurred that requires intervention.

Variables

You can use Platform Process Manager to pass variables to and from scripts. Platform Process Manager supports three kinds of variables:

- Local variables, that allow you to set a value of a variable and have the value available within a flow;
- Global variables, that allow you to set a value of a variable and have the value available anywhere within the Platform Process Manager Server.
- Environment variables, that allow you to submit a job with an environment variable, including a user or local variable.

File naming conventions

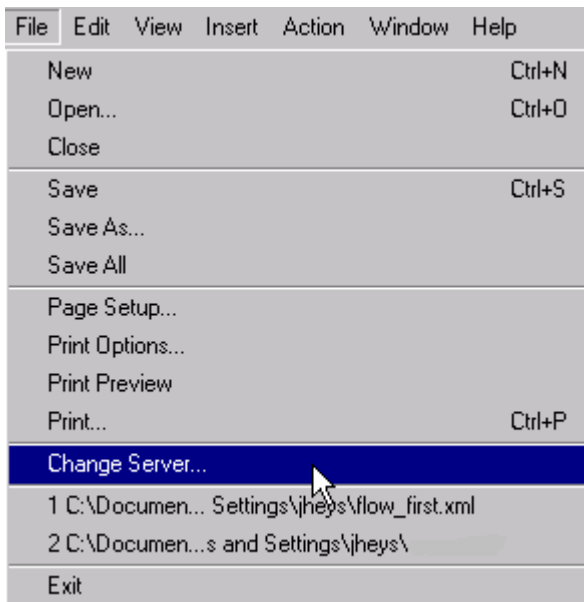
This guide uses UNIX file naming conventions to illustrate file names. However, if the file you are referencing is on a Windows file system, use Windows file naming conventions where applicable.

Change your server

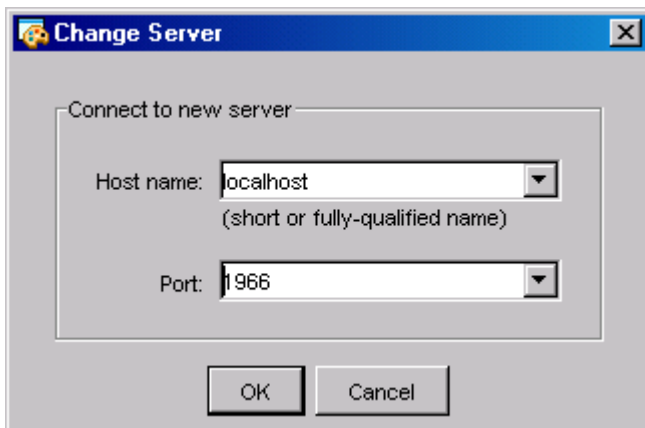
You can change the server you are using in any of the three client applications (Flow Editor, Flow Manager, and Calendar Editor).

Changing the server affects only the current session of the client application. The next time you open the client application, the server defined in `js.conf` is still used.

1. In any of the client applications, select File > Change Server.



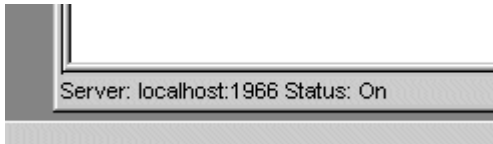
The Change Server dialog opens.



2. Specify the host name and port for the new server. Use the drop down list to select from any host names or ports that you have specified before.
3. Click OK.

The client application connects to the new server. You may be asked for your username and password. If the connection fails, the previous server is used instead.

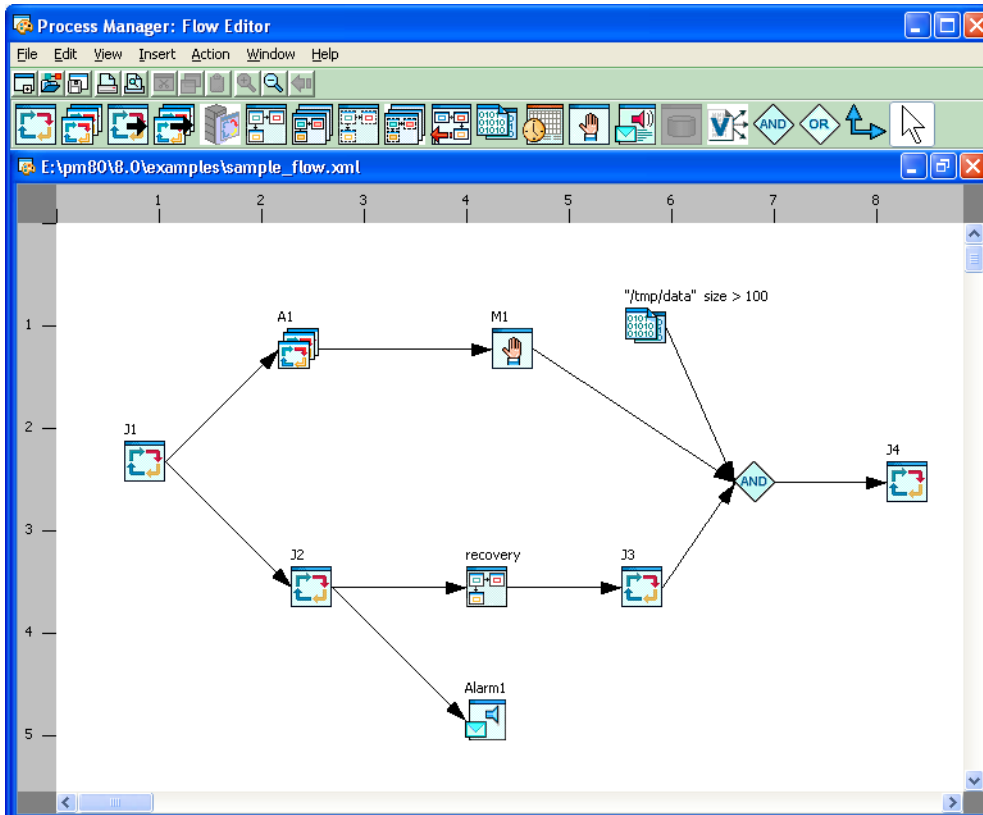
4. Check you server host name and port number in the lower left hand corner of your client application. In Flow Manager, status is also available.



You have changed your server.

About flow definitions and flows

A flow definition is a collection of Platform Process Manager work items (jobs, job arrays and subflows) and their relationships. These Platform Process Manager work items are defined graphically in the Flow Editor, where the relationships between the work items—any dependencies they may have on each other—are also represented graphically. The following picture illustrates a flow definition that consists of two jobs (J1 and J2) a job array (A1) and an imbedded subflow (recovery):



In the above flow definition, J1 must complete before J2 and A1 can run. If J2 fails, the subflow recovery runs.

Actions you can perform against flow definitions

The following terms have specific meanings within the Platform Process Manager context:

Publish

The term publish is used to describe the act of enabling a target flow to become available to dynamic subflows and flow arrays.

Unpublish

The term unpublish is used to describe the act of removing a target flow from the list of target flows available to dynamic subflows and flow arrays. The target flow is no longer available to dynamic subflows and flow arrays.

Hold

The term hold is used to describe the act of preventing Platform Process Manager from running a flow, even though the flow definition has been submitted to Platform Process Manager and is recognized by Platform Process Manager. Holding a flow definition essentially causes Platform Process Manager to ignore that definition until such time as it is released or explicitly triggered.

Release

The term release is used to describe the act of requesting that Platform Process Manager once again manage a flow definition—to release a flow definition that is on hold.

Trigger

The term trigger is used to describe the act of initiating the running of a flow. When a flow definition is triggered, a flow is created.

Remove

The term remove is used to describe the act of removing a flow definition from the Platform Process Manager system. After a flow definition is removed, Platform Process Manager no longer knows about it and cannot schedule any new occurrences of the flow.

What can I do with a flow definition?

- Submit and run the flow immediately, where the definition of the flow is not stored in the Platform Process Manager system. Platform Process Manager is only aware of the specific, adhoc occurrence of the flow.
- Submit a flow definition to be triggered manually at a later time.
- Submit a flow definition to run on a recurring basis, on a particular schedule.
- Submit a flow definition to run when a file reaches a particular state.
- Define specific routines as individual flow definitions, so that each can be reused like a subroutine within other flow definitions.
- Set exit conditions on a flow definition that contains multiple branches, so that completion of any single branch constitutes completion of the flow, or require that all branches complete before the flow is complete.
- Imbed a flow definition as a subflow within another flow definition.
- Use a flow to handle an exception in another flow.

What can I do with a flow?

- Kill, suspend, or resume an entire flow
- Rerun a failed flow, starting at the first job that failed in each path through the flow or from any rerun starting points that you set in the flow
- Rerun a flow while a job is still running, or rerun a flow that is done, starting at the first job that failed in each path through the flow or from any rerun starting points that you set in the flow

What can I do with a job?

- Kill a running job
- Hold a waiting job in a running flow
- Rerun a job in a completed flow

- Force a job complete in a completed flow

Where do I store my flow definitions?

You can store your flow definitions locally on your own computer, or you can store them on a shared file system. If other users will be creating similar flow definitions, you can create flow definitions that perform common routines so that you can share them with other users.

What makes a flow Done?

Unless you specify otherwise by defining an exit condition for the flow, Platform Process Manager follows this default behavior:

- A flow is considered successful with a status of Done only when all jobs in the flow complete successfully.
- A flow is considered to have failed with a status of Exit if any job in the flow fails.

What happens if a job exits?

Under the default behavior, if a job in a flow exits, no additional jobs in the flow are dispatched, although any currently running jobs will continue running until they complete. If you do not want the flow to exit if a job fails, you must specify an exit condition for the flow and handle the exit condition explicitly.

How does Platform Process Manager know when my flow is complete?

A large flow may diverge into multiple branches, depending on the design of your workflow. For example, job 1 may release multiple jobs, each of which has a string of successors. In some cases, you may want every work item in the flow to complete successfully before the flow is considered complete, and each branch of the flow must complete. This is the default behavior.

In other cases, your flow may include error recovery routines that only run under certain conditions. In those cases, you do not expect every job or path in the flow to complete. Platform Process Manager allows you to specify a completion attribute, which defines what constitutes completion of the flow. For example, you can specify that only one of many paths must complete.

Platform Process Manager Calendars

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

Platform Process Manager uses calendars to create time events to initiate an action at a particular date and time. The time event consists of the date and time to trigger the event, and the duration in which the event is valid. The calendar provides the date specification for the time event.

You create Platform Process Manager calendars using the Calendar Editor.

About calendars

Platform Process Manager uses three types of calendars:

- Those that consist of one or more specific dates
- Those that consist of an expression that resolves to a series of dates
- Those that combine other calendars to create complex expressions that resolve to a series of dates. These calendars can use logical operations within the calendar definition.

Each type of calendar definition can resolve to one or more dates.

About system calendars

System calendars are calendars that are predefined or created by your Platform Process Manager administrator. These calendars are owned by the virtual user “Sys” and can be referenced by any user. Only the Platform Process Manager administrator can create or delete system calendars.

Platform Process Manager includes a number of predefined system calendars that you can use without having to define them. In addition to the following list, your Platform Process Manager administrator may define other system calendars for your use. The following is a list of the system calendars that are ready for your use:

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

*The Holidays calendar is predefined with Platform Process Manager. However, it must be edited by a Platform Process Manager administrator to update the holidays for your company for each year.

About the calendar editor

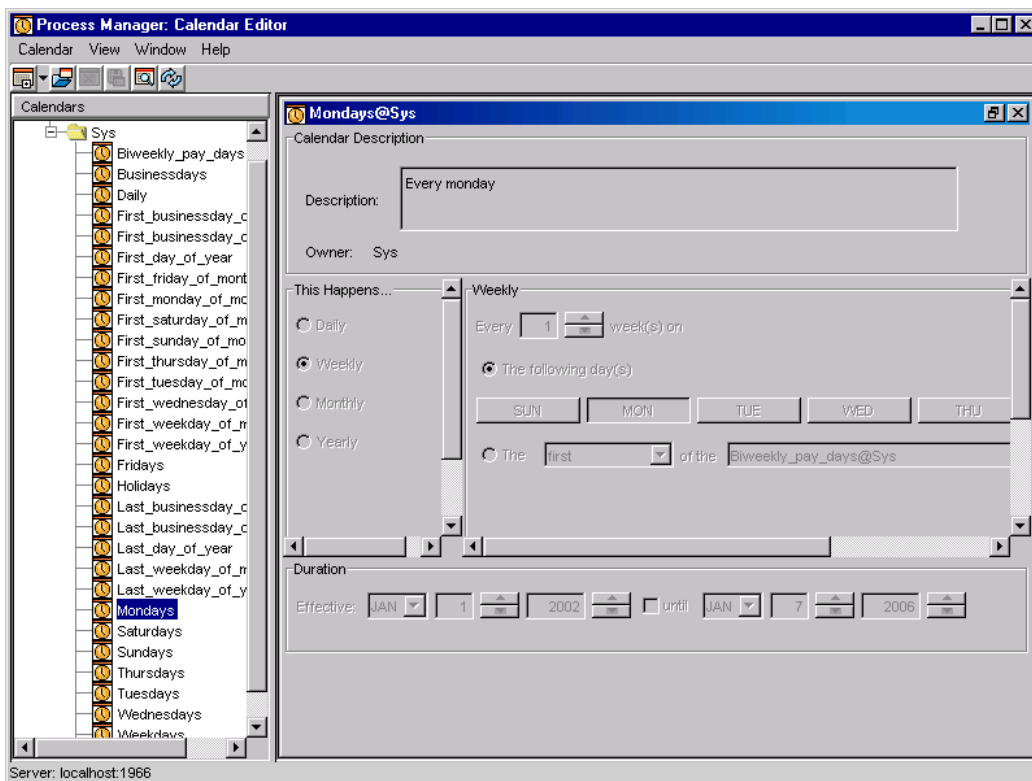
You use the Calendar Editor to create calendars that define the dates on which you want some action to take place. Calendars are required to create time events to trigger flows or dispatch jobs at a particular time. The 6 Server must be running before you can use the Calendar Editor.

About the Calendar Editor user interface

The Calendar Editor is divided into two panes:

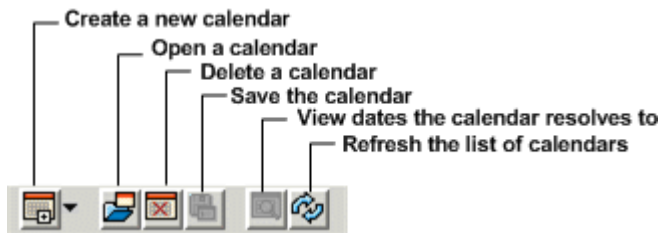
- The list of calendars in the left-hand pane
- The calendar definition in the right-hand pane

When you create a new calendar, you define the calendar in the right-hand pane. When you save the calendar, it appears in the list of calendars, under your user ID in the left-hand pane.



About the toolbar

The Calendar Editor toolbar looks like this:



About calendar names

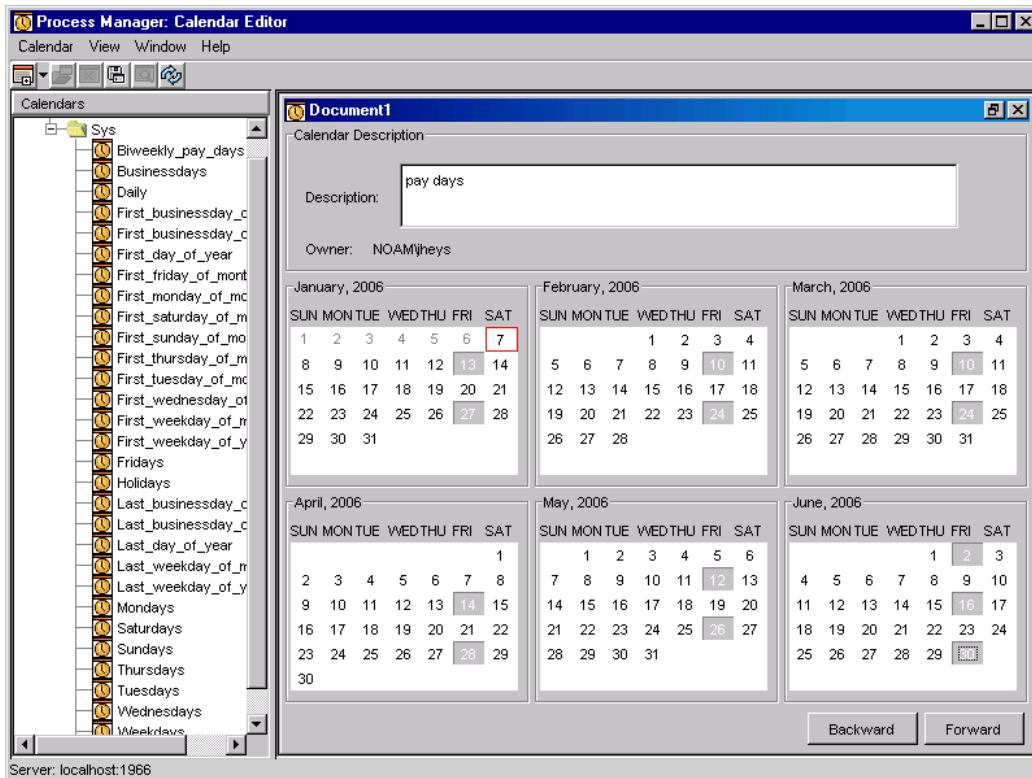
When you create a calendar, you need to save it with a unique name. Some rules apply:

- 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

Create a calendar with specific dates

Use this method to create a calendar when:

- The calendar needs to be valid for only one or two dates
 - The calendar needs to be valid for specific, random dates that do not repeat with any pattern
1. Ensure that Platform Process Manager is running, and open the Calendar Editor.
 2. From the Calendar menu, select New Calendar, and select Clicking on Date(s). The date selection dialog box appears. The dialog box is shown here with the list of calendars expanded on the left.



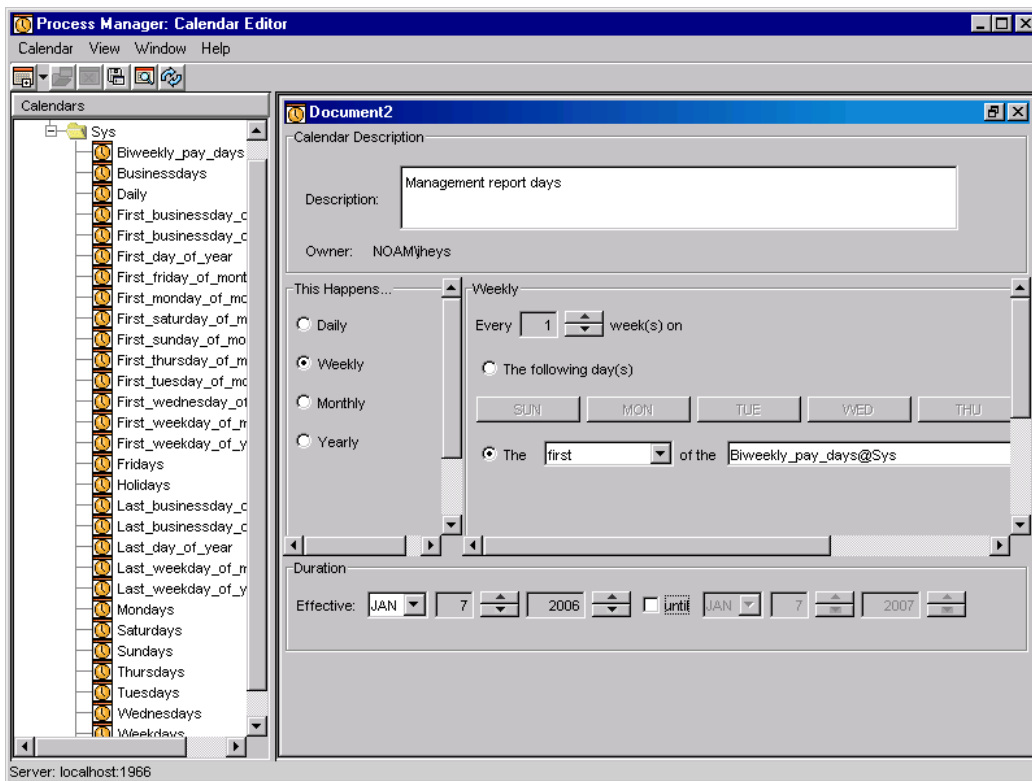
3. In the Description field, specify a description for the calendar that makes it obvious when this calendar is valid. This description is very useful: it is displayed in the fly-over text when you point to a calendar name in the list, and helps you quickly determine which calendar you want to use.
4. Select the dates on which you want this calendar to be valid by left-clicking on each date. You cannot select dates in the past.
5. When you have finished selecting dates, save the calendar: from the Calendar menu, select Save Calendar. When prompted, specify a meaningful name for the calendar. Click Save. The calendar is added to the list of centrally stored calendars, under your user name.

Create a calendar using an expression

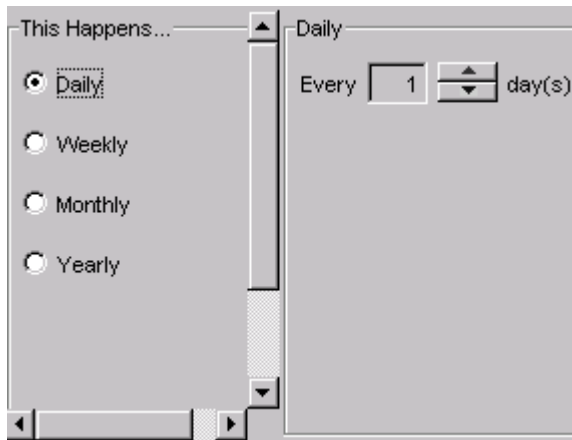
Use this method to create a calendar when:

- The calendar needs to be valid every n th day, week, month or year
- The calendar needs to be valid for the same dates every year
- The calendar needs to be valid for the same dates every month
- The calendar needs to be valid for the same days of the week every week

1. Open the Calendar Editor.
2. From the Calendar menu, select New Calendar and select Specify Pattern. The calendar expression dialog box appears. The dialog box is shown here with the list of calendars expanded on the left.

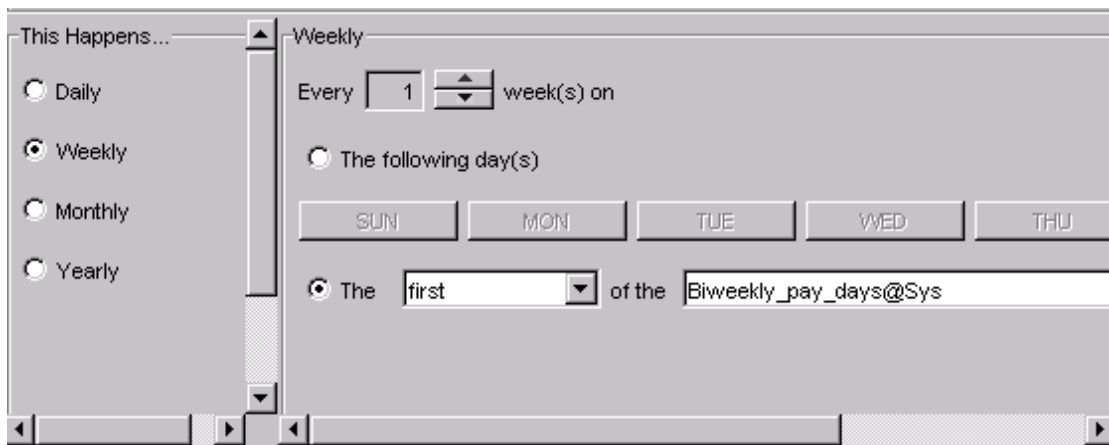


3. In the Description field, specify a description for the calendar that makes it obvious when this calendar is valid. This description is very useful: it is displayed in the fly-over text when you point to a calendar name in the list, and helps you quickly determine which calendar you want to use.
4. Choose one of the following options:
 - If the expression should be true every n days, in the This Happens ... field, select Daily. Then, in the Daily field, specify the number of days between occurrences. For example, if the expression should be true every day, leave the selections at Daily and Every 1 day(s).

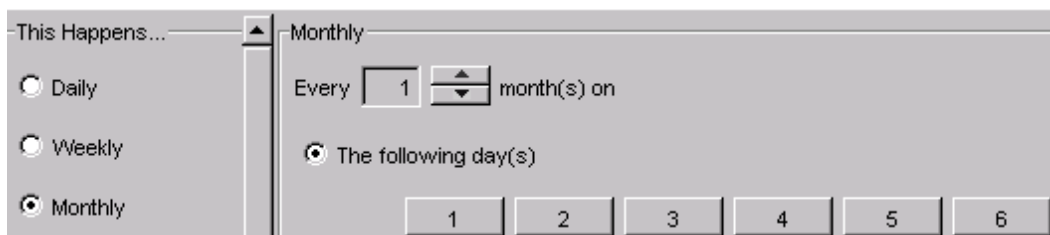


If the expression should be true every other day, specify Every 2 day(s).

- If the expression should be true on specific days every week, or every n weeks, in the This Happens ... field, select Weekly. Then specify how frequently this occurs, in the Every n week(s) field. Then click on the appropriate days of the week. For example, if the expression should be true on Mondays, Wednesdays and Fridays, click the MON button, then click the WED button, then click the FRI button.



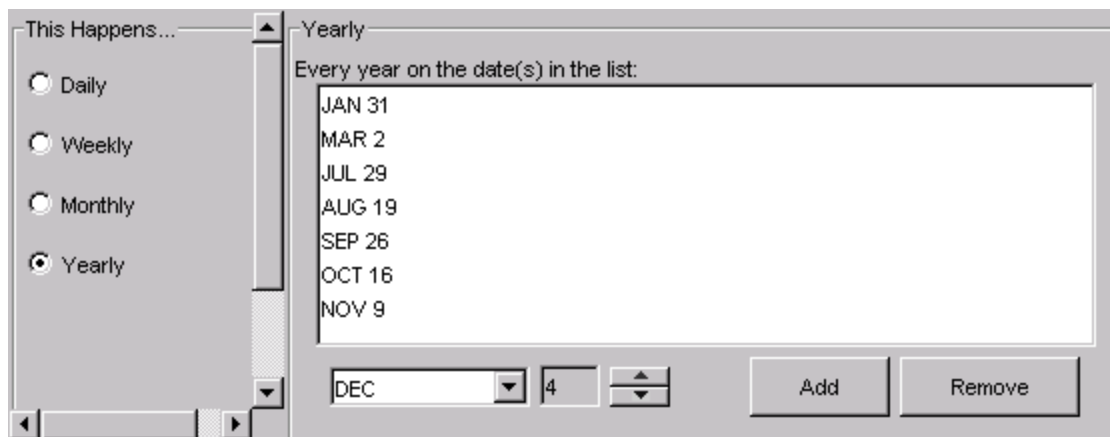
- If the expression should be true on specific days every month or every n months, in the This Happens ... field, select Monthly. Then specify how frequently this occurs, in the Every n month(s) field. Then click on the appropriate dates or days of the week. For example, if the expression should be true on the 6th day of every month, leave the frequency at every 1 month and click 6.



- If the expression should be true on specific days every year or every n years, in the This Happens ... field, select Yearly. Then specify each date by selecting the month and date and clicking Add. For example, if you want to specify the last date of each month, in the month field, select Jan, and in the date field, select 31. Continue to select the remaining dates.

Tip:

To quickly get to the later dates in the month, click down from 1 to get to 31.



5. Optional. In the Duration field, specify the time in which this calendar should be valid. If you want this calendar to be valid for an indefinite period of time, do not specify any end date for the duration. The beginning of the time period defaults to today's date.
6. Optional. Verify that the expression yields the correct results: from the View menu, select View Occurrences. A calendar is displayed with all of the resulting dates highlighted.
7. Save the calendar: from the Calendar menu, select Save Calendar. When prompted, specify a meaningful name for the calendar. Click Save. The calendar is added to the list of centrally stored calendars, under your user name.

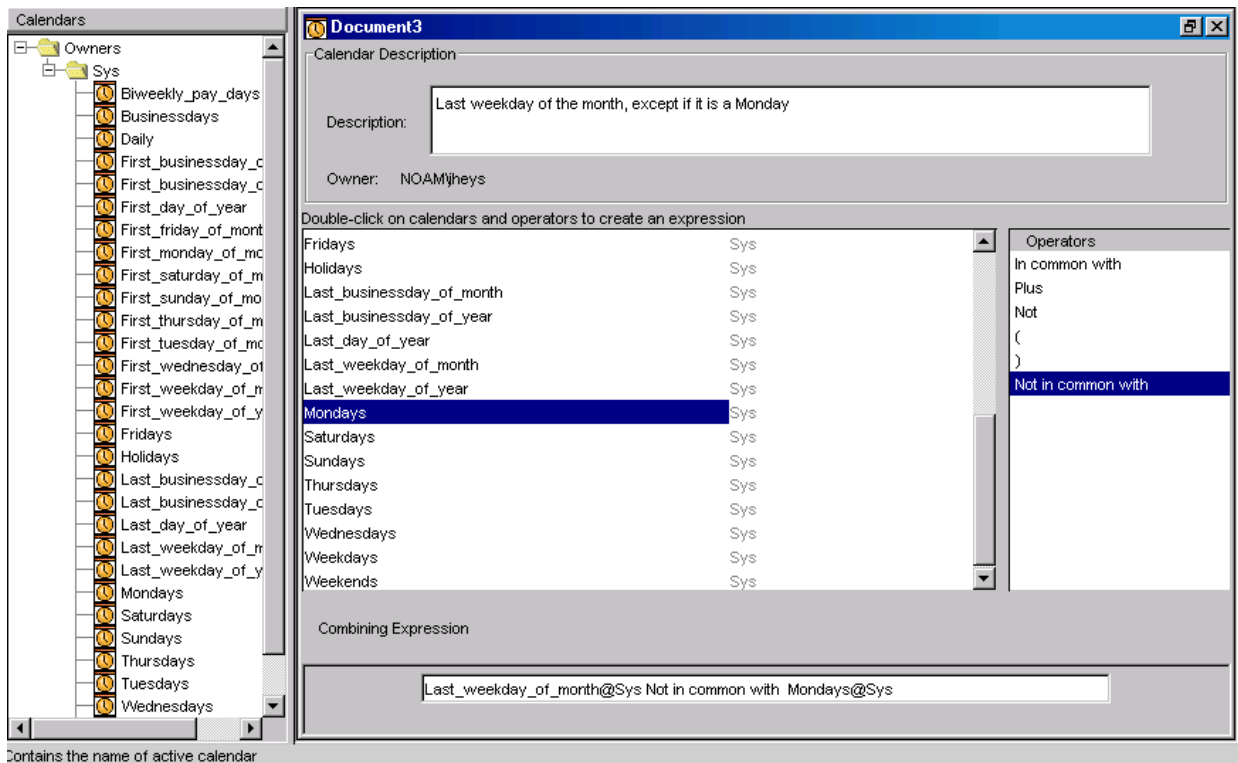
Create a calendar with a complex expression

Use this method to create a calendar when:

- The calendar needs to be valid on certain days, but must exclude other days, such as holidays
- The calendar needs to be valid on dates that are already defined in one calendar, and also on dates already defined in another calendar

1. Open the Calendar Editor.
2. From the Calendar menu, select New Calendar and select Combine Calendars. The combine calendars dialog box appears.

The dialog box is shown here with the list of calendars expanded on the left.



3. In the Description field, specify a description for the calendar that makes it obvious when this calendar is valid. This description is very useful: it is displayed in the fly-over text when you point to a calendar name in the list, and helps you quickly determine which calendar you want to use.
4. Create an expression that combines the calendars as required: double-click on calendar names and operators to create the desired expression. See “Operators and their meanings” for a description of the operators.

Tip:

To see the operators, you may need to drag the operator window over from the very right-hand side of the main window.

For example, if you want to create a calendar that is true on Mondays and Tuesdays, double-click on the calendar Mondays. Then double-click on Plus, then double-click on the calendar Tuesdays. The new calendar will be valid every day that Mondays is valid plus every day that Tuesdays is valid.

5. Optional. Verify that the expression yields the correct results: from the View menu, select View Occurrences. A calendar is displayed with all of the resulting dates highlighted.
6. Save the calendar: from the Calendar menu, select Save Calendar. When prompted, specify a meaningful name for the calendar. Click Save. The calendar is added to the list of centrally stored calendars, under your user name. For information on naming calendars, see “About calendar names” section.

Operators and their meanings

When creating a calendar expression, you can choose from the following operators:

Operator	Description
In common with	Use this operator to resolve to the intersection of two sets of dates—only those dates in common between two calendars
Plus	Use this operator to combine two sets of dates—any of the dates specified in the two calendars
Not	Use this operator to exclude the dates in a calendar—use only those dates that are not included in the calendar.
(Use this operator to begin an expression nested within the expression
)	Use this operator to end an expression nested within the expression.
Not in common with	Use this operator to exclude dates. First specify the expression that resolves to the dates you do want to include, then use this operator followed by those dates you do not want to include.

Calendar examples

Each of the these examples assumes that the Calendar Editor is up and running.

Example: Mondays, except on holidays

1. Ensure you have a calendar that repeats every Monday. Typically, that is the system calendar Mondays@Sys.
2. Ensure you have a calendar called 'Holidays' that defines all of the non-working holidays for your company. Typically, that is the system calendar Holidays@Sys.
3. From the Calendar menu, select New Calendar, then choose Combine Calendars.
4. Define a combining expression that specifies 'mondays' but not 'holidays' as follows:

Mondays@Sys Not in common with Holidays@Sys

Do this by double-clicking on Mondays@Sys, then double-click on Not in common with, and then Holidays@Sys.

Tip:

To see the operators, you may need to drag the operator window over from the very right-hand side of the main window.

5. Provide a meaningful description for the calendar, and save it with a unique name, such as 'workingmondays'.

Example: every second Friday

1. From the Calendar menu, select New Calendar, then choose Specify Pattern.
2. Click Weekly.
3. In the Every n weeks field, specify 2.
4. Ensure The following days is selected, and click FRI, as follows:

The screenshot shows a dialog box for configuring a calendar. On the left, under 'This Happens...', the 'Weekly' radio button is selected. On the right, under 'Weekly', the 'Every 2 week(s) on' field is set to 2. Below that, the 'The following day(s)' radio button is selected. A row of buttons for days of the week (SUN, MON, TUE, WED, THU, FRI, SAT) is shown, with 'FRI' highlighted. At the bottom, there is an option 'The first of the Biweekly_pay_days@Sys'.

5. Provide a meaningful description for the calendar, and save it with a unique name, such as 'oddfridays'.

Example: last working Friday of month

1. Ensure you have a calendar that defines working days. Typically, that is the system calendar businessdays@Sys.
2. Create a calendar called last_friday_of_month, similar to the system calendar first_friday_of_month@Sys.
3. From the Calendar menu, select New Calendar, then choose Combine Calendars.
4. Define a combining expression that specifies last_friday_of_month and the days it has in common with businessdays@Sys as follows:

last_friday_of_month@user In common with businessdays@Sys

Do this by double-clicking on `last_friday_of_month`, then double-click on `In common with`, and then `businessdays@Sys`.

Tip:

To see the operators, you may need to drag the operator window over from the very right-hand side of the main window.

5. Provide a meaningful description for the calendar, and save it with a unique name, such as `'lastworkingfriday'`.

Edit an existing calendar

You can edit an existing calendar to change the dates on which it is valid, or you can edit a calendar and save it using another name. You can use this method to create a new calendar that is similar to an existing one.

You can edit only those calendars owned by your user ID.

What you are able to change in a calendar depends on the method used to create the calendar. For example, if the calendar was created using an expression, you must change the expression to change the resulting dates.

You cannot change a calendar from one type to another. For example, if the calendar was created by clicking on dates, you cannot change it to contain an expression.

You use this option to change the dates on a calendar that was created by clicking on dates:

1. In the tree view on the left, double-click on the calendar you want to edit.
2. Deselect any previously selected dates you want to delete from the calendar by clicking on them.
3. Click on any new dates that you want to add to the calendar.
4. From the Calendar menu, select Save Calendar, or close the calendar, at which time you will be prompted to save it.

Edit a calendar to change the pattern

You use this option to change the dates on a calendar that was created by specifying a pattern:

1. In the tree view on the left, double-click on the calendar you want to edit.
2. Ensure you remove any previously selected dates you no longer want selected. For example, if the current pattern includes Mondays and Tuesdays, but the new pattern will be Tuesdays and Fridays, you need to click on Mondays to remove the selection.
3. Specify the new pattern.
4. From the Calendar menu, select Save Calendar, or close the calendar, at which time you will be prompted to save it.

Edit a calendar to change calendar combinations

You use this option to change the dates on a calendar that was created by combining calendars:

1. In the tree view on the left, double-click on the calendar you want to edit.
2. Edit the combining expression: you can delete terms in the expression by highlighting them and clicking the Delete button. However, if you want to insert terms from the list by double-clicking on them, they will be inserted at the end of the expression.
3. From the Calendar menu, select Save Calendar, or close the calendar, at which time you will be prompted to save it.

Delete a calendar

Periodically, you may want to delete unused calendars from Platform Process Manager. You can only delete those calendars owned by your user ID.

1. In the tree view on the left, right-click on the calendar you want to delete.
2. Select Delete Calendar.
3. Confirm that you want to delete the calendar by clicking Yes.

Define your flow

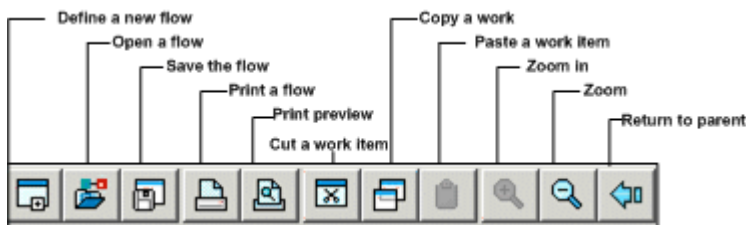
You use the Flow Editor to create or edit flow definitions that group related jobs, job arrays and subflows, so that they can be triggered, run, and controlled as a unit.

About the Flow Editor user interface

The Flow Editor user interface consists of a workspace and a design palette to define work items in the flow definition: jobs, job arrays, manual jobs, subflows, time events, file events, proxy events, link events, and alarms.

About the toolbar

The Flow Editor toolbar looks like this:



About the design palette

The Flow Editor design palette can be separated from the toolbar, and moved to a convenient location in the work space. It looks like this:



Ways to create a flow definition

There are many ways to create a flow definition. These are three of them:

1. Completely define one job at a time
2. Draw all of the work items in the flow and then fill in the details
3. A combination of the above methods—draw some of the work items and define them, then draw more work items and define them, and so on

Note:

To support multiple Platform Process Manager servers in a single LSF cluster, you must ensure that each combination of user name and flow name is unique within the cluster. This avoids conflicts and ensures that each job is unique among the multiple Process Manager servers.

1. One job at a time
 1. Draw the first job on the workspace
 2. Define the details of the job
 3. Draw the next job
 4. Define the details of that job
 5. Draw the dependency line between the two jobs
 6. Continue with the next job in the flow, and repeat
2. Draw the flow, then fill in the details
 1. Draw all of the jobs, flows and events on the workspace
 2. Draw the dependency lines between the work items, establishing the relationships between them
 3. Define the details for each job and job array, one at a time
3. Combine methods 1 and 2
 1. Draw a group of jobs, job arrays and flows on the workspace
 2. Draw the dependency lines between those work items
 3. Define the details for each job and job array, one at a time
 4. Create another section of the flow

There is no right or wrong way. Only you can decide which method works best for you

How do I know if a job or job array is undefined?

There are two ways in which Platform Process Manager informs you if you have drawn a job or job array but not yet defined it:

1. The system-assigned job name is displayed in red text
2. When you save a flow that contains undefined jobs or job arrays, you receive a message that lists all of the undefined work items in the flow

Using the example flows

The Platform Process Manager Client package includes sample flow definitions that you can use, to test your Platform Process Manager installation, or to learn from or to modify for your own use. These examples are located in the examples directory of the Client installation. Each example is a simple flow that illustrates a particular type of activity:

The flow definition named...	Illustrates...
Example_1	<ul style="list-style-type: none"> A simple job dependency, where one job runs when its predecessor completes successfully
Example_2	<ul style="list-style-type: none"> The use of a recovery job
Example_3	<ul style="list-style-type: none"> The use of a manual task and a job array
Example_4	<ul style="list-style-type: none"> The use of a file event to trigger some processing
Example_5	<ul style="list-style-type: none"> The use of a time event to run a job array, which raises an alarm if it misses its schedule*

*To successfully use this flow definition, your administrator must first define an alarm called Critical_Job_Failed

You can open these flows to view them from the Flow Editor, but if you want to run them or use them to create a new flow, unless you have Platform Process Manager administrator authority, you must change the owner of each work item in the flow before you can successfully run any of the sample flows.

View the sample flow definition

1. In the Flow Editor, from the File menu, select Open.
2. Locate the `examples` directory within the Client installation.
3. Select a flow definition and click OK.

Use a sample flow definition

1. Open the flow definition as described above.
2. Double-click on a job or job array in the flow definition.
3. On the General tab, locate the Run as... field at the bottom of the dialog.
4. Change the user name to your user ID, and click OK.

Create a flow diagram

For purposes of clarity, this topic assumes you will drag and drop all of the jobs onto the workspace, creating a visual representation of the work flow, and then define each job in the workspace.

Create a simple flow diagram

1. Click the Insert Job button to put the Flow Editor in job placement mode—when you left-click in the workspace, a job icon appears.
2. Drop the appropriate number of job icons in the workspace, placing them in the order in which you want the jobs to run, typically with the first job to run at the left and the last job to run at the right. Unique job names are assigned automatically to the jobs in the workspace. You can change these later if you like.
3. Change to job dependency mode by clicking the Insert Dependency button.
4. Draw job dependencies by left-clicking on the job that must run first, then left-clicking on the job that runs next. The job at the arrow end of the line cannot run until the job at the originating end of the arrow completes.

Refer to the following example:



Job J2 cannot run until job J1 completes.

5. Double-click on each job in the flow definition. The Edit Job dialog appears.
6. In the Command to run field, specify the command that this job is to run. For example, on Windows:

Define the Job

Name:

Command to run:

Login shell to use:

Part of project:

or on UNIX:

Define the Job

Name:

Command to run:

Login shell to use:

Part of project:

7. In the remaining input fields and tabs, specify any other details required to define the job.

8. Save the flow definition. You can save it in your local file system or in a shared file location.

Other things you can do

You can also do the following:

- Copy a job
- Print the flow definition

Copy a job

1. Right-click on the job you want to copy, and select Copy.
2. Right-click in the workspace where you want to place the new job, and select Paste.
3. Optional: double-click on the new job and change the name of the job.

Print the flow definition

1. From the File menu, select Print Preview to see how your flow definition looks on paper. You can adjust the spacing in your flow to avoid breaking icons at a page boundary.
2. From the File menu, select Print... and click OK.

Include a job array in the flow diagram

You can include a job array in the flow diagram. Using a job array is a convenient way to specify a group of jobs that share the same executable and resource requirements, but use different input data, with a single definition. All jobs in a job array have the same name and same job ID. Each job runs the same executable. Any parameters you specify apply to all jobs in the array. All jobs use an input file from the same location, and write to the same output file location. However, each element of a job array is distinguished by its array index. Before you can use a job array, you need to prepare your input files. See “Job Arrays” chapter in *Administering Platform LSF* for more information.

Insert a job array

1. Click the Insert Job Array button to change to job array placement mode—when you left-click in the workspace, a job array icon appears.
2. Left-click in the workspace in the location where you want to insert the job array. A job array icon appears in the workspace.
3. Draw the lines describing any dependencies the job array has on other work items in the flow.

Define job array details

1. Double-click on the job array. The Edit Job Array dialog appears.

Define the Job Array

Name: Index expression: ...

Command to run:

Login shell to use: ... Part of project: ...

2. In the Name field, specify a name for the job array. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_) in the job array name. A unique name is automatically assigned to the job array, but you can change it to make it more meaningful.
3. Create an index expression using one of the following methods.

- a) Freeform expression: In the Index expression field, specify an expression that defines the index for the array. The index expression can be a simple range of positive integers, such as

1-5

or a series of comma-separated numbers, such as

1,4,5,6,8

In this example, the job array will consist of five jobs.

The index expression can also be in the format

start-end[:step]

where *start* is used to specify the start of a range of indices, and *end* specifies the end of the range. *step* specifies the value to increment the indices in the range. The index begins at *start*, increments by the value of *step*, and does not increment past the value of *end*. For example:

1-10:2

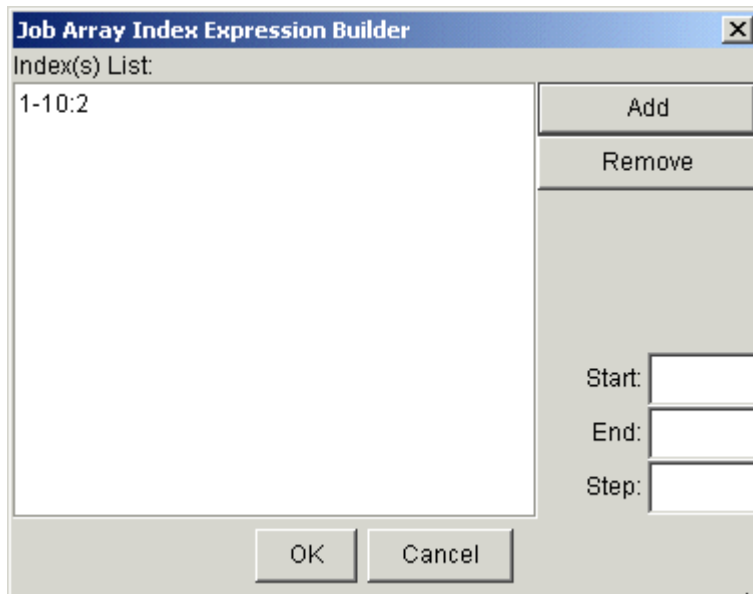
specifies a range of 1 to 10 with a step value of 2, creating indices of 1,3,5,7 and 9. This creates 5 jobs in the job array, whose input files will have suffixes of .1, .3, .5, .7 and .9 respectively.

You can also specify a user variable for any of the values in the index expression—*start*, *end* or *step*. For example:

1-#{COUNT}

where the value of COUNT might be set within a job that runs prior to this job array in this flow.

- b) Using the Job Array Index Expression Builder dialog.
 1. Click the ... button next to the Index expression field. The Job Array Index Expression Builder dialog appears.



2. In the Start field, specify the start of the range of indices, which will be the first job name suffix. For example: **1**. Or specify a variable. For example: **#{beginix}**
 3. In the End field, specify the end of the range of indices. For example: **10**. Or specify a variable. For example: **#{endix}**
 4. Optional. In the Step field, specify the value to increment the indices in the range. The default is a step of 1. For example: **2**. Or specify a variable. For example: **#{stepix}**
 5. Click Add to add the expression to the list.
 6. Repeat steps ii through v until you have completed specifying all the index ranges.
 7. Click OK. The expression you created is inserted in the Index expression field.
4. In the Command to run field, specify the command that each of the jobs in the array will run. Include any arguments the command requires.
 5. Ensure that all the input files for the jobs in your job array are in the same directory. By default, the current working directory is assumed. If the files are not in the current working directory, specify an absolute path as seen by the Process Manager Server.
- Also ensure that all the input files have the same name, with a numerical suffix that corresponds to the index of the job array element that will use it.
6. In the Input file field, specify the name of the input file, as follows:

input_file_name.%i

which specifies to use the input file that corresponds to the index for each element in the array. For example:

`i nput . 1`

7. In the Output file field, specify the name of the output file, as follows:

`output_file_name.%I.%J`

which specifies to create an output file that corresponds to the index, followed by the job ID for each element in the array. For example:

`out put . 1 . 3993`

8. Optional: In the Max. concurrent jobs field, specify the maximum number of jobs in the array that can run at the same time.
9. Specify any additional options on this tab and the other definition tabs.
10. Click OK.

Preparing job array input files

Platform Process Manager provides methods for coordinating individual input and output files for the multiple jobs created when submitting a job array. These methods require your input files to be prepared uniformly. To accommodate an executable that uses standard input and standard output, Platform Process Manager provides variables that are resolved at runtime: %I (job array index) and %J (job ID).

All input files for your job array must be located in the same directory. Specify an absolute path to the directory containing the input files when defining your job array.

Each file consists of two parts: a consistent name string and a variable integer that corresponds directly to an array index. For example, the following file names are valid input file names for a job array. They are made up of the consistent name `i nput` and integers that correspond to job array indices from 1 to 1000:

`i nput . 1, i nput . 2, i nput . 3, ..., i nput . 1000`

Include a job submission script in the flow diagram

You can define and submit customized job submission scripts to control and monitor jobs.

1. Click the Insert Job Submission Script button to change it to placement mode—when you left-click in the workspace, a job submission script icon appears.
2. Left-click in the workspace in the location where you want to insert the job submission script. A job submission script icon appears in the workspace.
3. Draw the lines describing any dependencies the job submission script has on other work items in the flow.

Define job details

1. Double-click on the job submission script. The Edit Job dialog appears.
2. In the Name field, specify a name for the job submission script. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_) in the job submission script name.
3. In the File field, specify the absolute path of the job submission script. Ensure that this path is accessed and executed by Platform Process Manager Daemon jfd. Make sure that your script meets the conditions as explained in [Content of the job/job array submission script](#) on page 56.
4. Specify any additional options on this tab and the other definition tabs. For detailed information about each of the options, see [Details of a job](#) on page 106.
5. Click OK.

Include a job array submission script in the flow diagram

Using a job array submission script is a convenient way to specify a group of job submission scripts that share the same executable and resource requirements, but use different input data, with a single definition. Any parameters you specify apply to all job submission scripts in the array. All jobs use a file from the same location. However, each element of a job array submission script is distinguished by its array index. Before you can use a job array submission script, you need to prepare your input files.

1. Click the Insert Job Array Submission Script button to change it to placement mode—when you left-click in the workspace, a job array submission script icon appears.
2. Left-click in the workspace in the location where you want to insert the job array submission script. A job array submission script icon appears in the workspace.
3. Draw the lines describing any dependencies the job array submission script has on other work items in the flow.

Define job array details

1. Double-click on the job array submission script. The Edit Job Array dialog appears.
2. In the Name field, specify a name for the job array submission script. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_) in the job array submission script name.
3. Create an index expression as explained in the job array details section.
4. Ensure that all the script files for the jobs in your job array are in the same directory. By default, the current working directory is assumed. If the files are not in the current working directory, specify an absolute path as seen by the Platform Process Manager Server.

Also ensure that all the input files have the same name, with a numerical suffix that corresponds to the index of the job array element that will use it.

5. In the File field, specify the job array submission script directory. This directory should be easily accessible for Platform Process Manager Daemon jfd. Make sure that your script meets the conditions as explained in [Content of the job/job array submission script](#) on page 56.
6. Specify any additional options on this tab and the other definition tabs. For detailed information about each of the options, see [Details of a job](#) on page 106.
7. Click OK.

Content of the job/job array submission script

Platform Process Manager can successfully track the job/job arrays submitted through the customized scripts only if these conditions are met:

1. Specify job name (JS_JOB_NAME) and job array index (JS_INDEX_LIST) in the job/job array submission script.

For example,

```
bsub -q short -R "mem>1000" -J $JS_JOB_NAME$JS_INDEX_LIST my_command
```

2. Submit only one job or job array through the job/job array submission script.
3. Do not use the following bsub options in the script:

- a) -I/-Ip/-Is — interactive jobs
 - b) -K — submit a job and wait for the job to complete
4. The script must exit zero upon successful submission, or non-zero otherwise.

Include a static subflow in the flow diagram

At any time, you can include an existing flow definition as a static subflow within a flow diagram. This is especially useful for standardized flow definitions that you would like to reuse, such as backup and recovery routines, database update routines, and so on.

1. Click the Insert Static Subflow button to put the Flow Editor in static subflow placement mode. The Open dialog appears.
2. Locate the flow definition file you want to include and click Open.
3. Left-click in the workspace in the location where you want to insert the static subflow. The static subflow icon is added to the flow diagram.
4. Draw the lines describing any dependencies the static subflow has on other work items in the flow definition.
5. Optional. Specify additional attributes for the static subflow.
 - a) Right-click the static subflow and select Attributes.
 - b) To specify a working directory at the subflow level, use the Working directory field.

Tip:

You can use user variables when specifying the working directory.

All valid inner work items (subflows, jobs, and job arrays) in the static subflow 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 static subflow.

- c) To add input variables to the static subflow, click Modify, which is beside the Input Variables field.

From the Input Variables window, specify variables in the form of **Name=Value**. You can also specify user variables as input values.

When Platform Process Manager expands static subflows, these input variables are set at the subflow level.

Note:

If you set default values (by enabling **Specify a Default Value**), this creates both environment variables and user variables.

- d) To specify a description or instructions regarding the subflow, use the Description field.
- e) To monitor this subflow for a particular exception and handle it automatically, click the Exception Handling tab and specify the exceptions and handlers.

For more detailed information about exceptions and handlers, see [Handling exceptions](#) on page 143.

- f) Click OK to save the subflow settings.

Tip:

You can view (and edit) the jobs in a subflow by double-clicking on the subflow, or by right-clicking the subflow and selecting **Open Definition**. Any changes you make apply only to the imbedded subflow.

Include a static flow array in the flow diagram

You can include a static flow array as a subflow within a flow diagram. A static flow array works in a similar fashion to a job array, but at a subflow level.

When the static flow array is instantiated, a specific number of flow elements start to run either in parallel, or sequentially, depending on what you selected in the flow array attributes. By default, flow elements run in parallel.

Within a flow array element, the `#{ JS_FLOW_INDEX }` scoped variable specifies the array index of the static flow array element.

1. Click the Insert Static Flow Array button to put the Flow Editor in static flow array placement mode. The Open dialog appears.
2. Locate the flow definition file you want to include and click Open.
3. Left-click in the workspace in the location where you want to insert the static flow array. The static flow array icon is added to the flow diagram.
4. Draw the lines describing any dependencies the static flow array has on other work items in the flow definition.
5. Define the static flow array.
 - a) Right-click the static flow array and select Attributes.

The Flow Array Attributes dialog displays.

- b) Specify the name of the static flow array in the Name field.

The default name assigned to the static flow array is the name of the subflow you selected.

- c) Optional. To specify the description of the static flow array, use the Description field.
- d) Define the size of the array by specifying the First Element and Last Element fields.

If you do not specify a value for First Element, it defaults to 1. You must specify a value for Last Element that is larger than First Element.

Tip:

You can use user variables in the element fields to specify the flow array index, thus allowing static flow arrays with dynamic element fields. At runtime, before the static flow array is started, Platform Process Manager replaces the variables with values, which is usually set by a predecessor job.

- e) Optional. Specify Run flow array elements: whether flow array elements run in parallel or sequentially. By default, flow array elements run in parallel, as checked.
6. Optional. To specify a working directory at the static flow array level, expand the static flow array and use the Working directory field in the Flow Attributes of the flow array.

Tip:

You can use user variables when specifying the working directory.

All valid inner work items (subflows, jobs, and job arrays) in the static flow array 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 static flow array.

7. Optional. Monitor this static flow array for a particular exception, and handle it automatically.

For more detailed information about exceptions and handlers, see [Handling exceptions](#) on page 143.

1. Right-click the static flow array and select **Expand** to enter the subflow level.
2. Right-click on the workspace and select **Flow Attribute**.
3. Click the **Exception Handling** tab and specify the exceptions and handlers.

Tip:

You can view (and edit) the jobs in a subflow by double-clicking on the subflow, or by right-clicking on the subflow and selecting **Expand**. Any changes you make apply only to the imbedded subflow.

8. Click **OK** to save your static flow array definition.

Flow array element names

When a static flow array is run, each element takes a unique name in the form of ***flow_array_name (array_index)***.

For example, if the name of the static flow array instance is 1: usr 1: FA, and its index is 1 to 3, the three elements have the name of 1: usr 1: FA(1) , 1: usr 1: FA(2) , and 1: usr 1: FA(3) .

Viewing a static flow array

1. Right-click the static flow array in the instance diagram and select **View Elements**.

A list of flow array elements displays.

2. Select one of the flow array elements and click **View Flow**.

The diagram for the selected flow array element displays.

Include a dynamic subflow in the flow diagram

You can include a dynamic flow array as a subflow within a flow diagram. A dynamic flow array works in a similar fashion to a job array, but at a subflow level.

When the dynamic flow array is instantiated, a specific number of flow elements start to run either in parallel, or sequentially, depending on what you selected in the flow array attributes. By default, flow elements run in parallel.

A dynamic flow array, also called a flow array by reference, refers to a target flow that has already been submitted to Process Manager. Only flows that have been submitted to 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.

Within a dynamic flow array element, the `#{JS_FLOW_INDEX}` scoped variable specifies the array index of the dynamic flow array element.

1. Click the Insert Dynamic Subflow button to put the Flow Editor in subflow placement mode.

The Dynamic Subflow - Insert dynamic subflow window displays.

2. Specify the attributes for the dynamic subflow.
 - a) In the Name field, specify a name for the dynamic subflow.
 - b) In the Choose a flow to insert field, select a target flow for the dynamic subflow.

The eligible target flows are shown in the form of *user_name:flow_name*. Only flows that have been submitted to Platform Process Manager and published are shown in this field.

- c) To add input variables to the dynamic subflow in the Specify input variable values field, click Modify. The Input Variables window displays.

The variables are set in the form of **Name=Value**. You can specify user variables as input variable values. When Platform Process Manager expands dynamic subflows, these input variables are set at the subflow level.

Note:

If you set default values (by enabling **Specify a default value**), this creates both environment variables and user variables.

- d) Select how you want Platform Process Manager to update the dynamic subflow when the main flow is instantiated.
 - Use the default setting in main flow
 - Automatically update to the default version
 - Manually update to the default version
 - e) Optional. Specify Execution order: whether flow array elements run in parallel or sequentially.
 - f) Click OK to save your dynamic flow definition.
3. Left-click in the workspace in the location where you want to insert the dynamic subflow. The dynamic subflow icon is added to the flow diagram.
 4. Draw the lines describing any dependencies the dynamic subflow has on other work items in the flow definition.

Include a dynamic flow array in the flow diagram

You can include a dynamic flow array as a subflow within a flow diagram. A dynamic flow array works in a similar fashion to a job array, but at a subflow level.

When the dynamic flow array is instantiated, a specific number of flow elements start to run either in parallel, or sequentially, depending on what you selected in the flow array attributes. By default, flow elements run in parallel.

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.

Within a dynamic flow array element, the `{ JS_FLOW_INDEX }` scoped variable specifies the array index of the dynamic flow array element.

1. Click the Insert Dynamic Flow Array button to put the Flow Editor in dynamic flow array placement mode.

The Dynamic Flow Array - Insert dynamic flow array window displays.

2. Specify the attributes for the dynamic flow array.
 - a) In the Name field, specify a name for the dynamic flow array.
 - b) In the Choose a flow to insert field, select a target flow for the dynamic flow array.

The eligible target flows are shown in the form of *user_name:flow_array_name*. Only flows that have been submitted to Platform Process Manager and published are shown in this field.

- c) Define the size of the array by specifying the First Element and Last Element fields.

If you do not specify a value for First Element, it defaults to 1. You must specify a value for Last Element that is larger than First Element.

Tip:

You can use user variables in the element fields to specify the flow array index, thus allowing flow arrays with dynamic element fields. At runtime, before the dynamic flow array is started, Platform Process Manager replaces the variables with values, which is usually set be a predecessor job.

- d) To add input variables to the dynamic flow array in the Specify input variable values field, click Modify. The Input Variables window displays.

The variables are set in the form of **Name=Value**. You can specify user variables as input variable values. When Platform Process Manager expands dynamic flow arrays, these input variables are set at the subflow level.

- e) Select how you want Platform Process Manager to update the dynamic flow array when the main flow is instantiated.
 - Use the default setting specified in main flow
 - Automatically update to the default version
 - Manually update to the default version
- f) Optional. Specify Run flow array elements: whether flow array elements run in parallel or sequentially.
- g) Click OK to save your dynamic flow array definition.

3. Left-click in the workspace in the location where you want to insert the dynamic flow array. The dynamic flow array icon is added to the flow diagram.
4. Draw the lines describing any dependencies the dynamic flow array has on other work items in the flow definition.

Dynamic flow array element names

When a dynamic flow array is run, each element takes a unique name in the form of ***flow_array_name (array_index)***.

For example, if the name of the dynamic flow array instance is 1: usr 1: FA, and its index is 1 to 3, the three elements have the name of 1: usr 1: FA(1) , 1: usr 1: FA(2) , and 1: usr 1: FA(3) .

Viewing a dynamic flow array

1. Right-click the dynamic flow array in the instance diagram and select View Elements.

A list of dynamic flow array elements displays.

2. Select one of the dynamic flow array elements and click View Flow.

The diagram for the selected dynamic flow array element displays.

Include a manual job in the flow diagram

You can include a manual job in the flow diagram wherever you want to indicate a manual process that must take place before the flow can continue. Successors of the manual job cannot run until the manual job is explicitly completed.

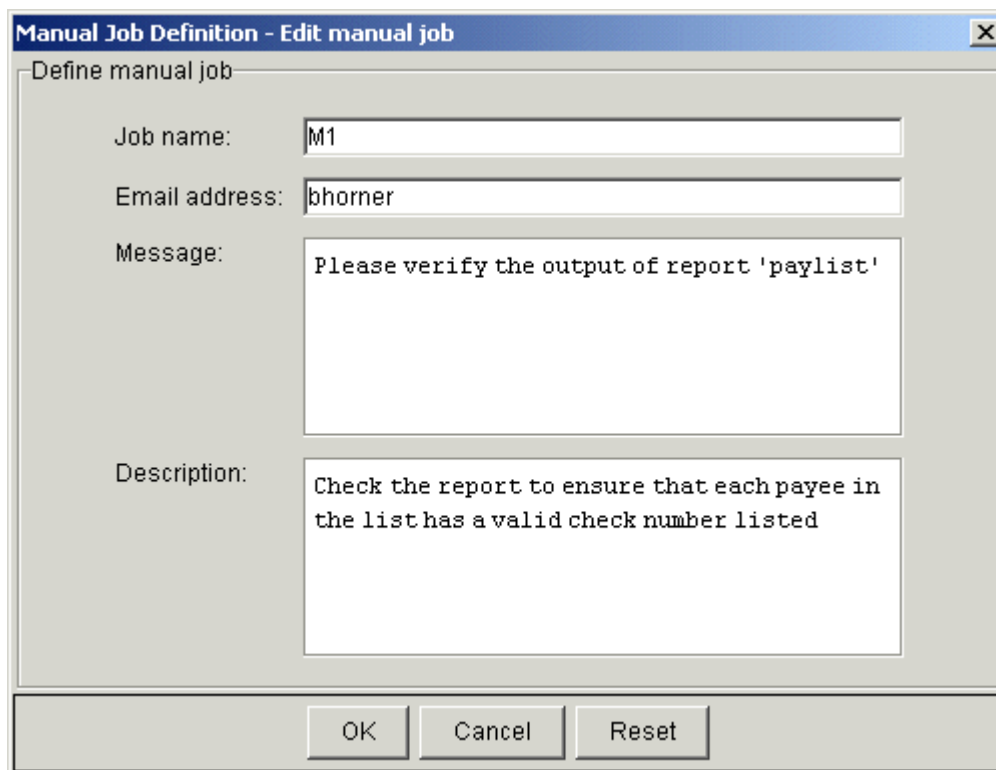
When the flow is ready for the manual job to be completed, an email is sent to the owner of the flow or job. When you define the manual job, you specify the email address and the text to be included in the email.

Including a manual job in a flow does not stop the entire flow from processing: only the specific path containing the manual job is halted until the job is completed.

1. Click the Insert Manual Job button to put the Flow Editor in manual job placement mode—when you left-click in the workspace, a manual job icon appears.
2. Left-click in the workspace in the location where you want to insert the manual job.
3. Draw the lines describing any dependencies the manual job has on other work items in the flow definition.

Define manual job details

1. Double-click on the manual job. The Manual Job dialog appears.



The screenshot shows a dialog box titled "Manual Job Definition - Edit manual job". The dialog has a close button (X) in the top right corner. The main area is titled "Define manual job" and contains four input fields:

- Job name:** A text box containing "M1".
- Email address:** A text box containing "bhorner".
- Message:** A text area containing the text "Please verify the output of report 'paylist'".
- Description:** A text area containing the text "Check the report to ensure that each payee in the list has a valid check number listed".

At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Reset".

2. In the Job name field, specify a unique, meaningful name for the manual job. You will use this name when completing the job as the flow runs. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_) in the manual job name. The Flow Editor assigns a unique name to

each manual job when you draw it on the workspace, so you are not required to change the name. However, if you want to change the name, you can. The name itself is required.

3. In the Email address field, specify the email address to notify when the job is ready for completion. If you do not change the email address, it defaults to your user ID. If you delete the email address, no notification is sent when the job requires completion.
4. In the Message field, specify the message text that should appear in the email notification. For example, if the manual job will be used to verify a report, you might include the following as the message text:

Verify the output of report payroll.

You can also specify a variable in the message. For example:

Check output from printer #{PRINT}.

5. In the Description field, add any descriptive text that may be used for managing this job within the flow. For example, if this job requires special instructions for operations staff, place those instructions here.
6. Click OK. The manual job appears in the workspace, and you can draw the appropriate dependency lines to any work items.

Specifying custom exit codes for successful job completion

By default, for a job to complete successfully, the exit code must be 0. Any other exit code indicates the job failed.

In some cases, however, you may want to use exit codes to pass information to subsequent work items and may want to use numbers other than 0 to indicate success.

You can do so by specifying these exit codes in the Job Definition dialog, Job Script Definition dialog, Manual Job Definition dialog, or Local Job Definition dialog, Non-zero success exit codes field.

This feature applies to LSF jobs, job scripts, local jobs, and manual jobs.

When you define custom success exit codes:

- 0 is always a success exit code, and is the default success exit code. You do not need to specify it.
- You can specify one number or a list of numbers separated by spaces, from 1 to 255.
- You can use user variables in the Non-zero success exit codes field. If the user variable cannot be resolved at runtime, it is ignored.
- When a job exits with 0 or any other specified success exit code, the job is considered successful and receives the Done state.
- When a job exits with an exit code other than 0 or the specified success exit codes, the job is considered to have failed and receives the status Exited.
- If you specify an application profile and `SUCCESS_EXIT_VALUES` is defined in `lsb. applications` for the application, `SUCCESS_EXIT_VALUES` is ignored.
- If a job is killed by a user in Process Manager or in LSF, custom success exit codes are ignored.

From Flow Editor

1. In Flow Editor, right-click on a job, job script, manual job, or local job, select Open Definition.
2. In the Non-success exit codes field, enter the exit codes that represent successful completion of the job.

Include a local job in the flow diagram

You can include a local job in the flow diagram.

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

There are several differences between local jobs that run on Windows, and local jobs that run on Linux and UNIX:

Item	Windows	Linux/UNIX
Behavior	The local job is blocking: when a local job is running, another local job will not be able to run until the local job that is running completes.	The local job is non-blocking: that is, several local jobs can be run in parallel.
Killing a local job	You cannot directly kill a local job in the same way as you kill any other job. The local job can only be killed as a result of the flow being killed, or if it runs for longer than the configured timeout value.	You can kill a local job in the same way as you kill any other job. The local job may also be killed as a result of the flow being killed, or because it ran for longer than the configured timeout value.
Suspending and resuming a local job	If you suspend or resume a flow that contains local jobs, the local jobs will be killed and rerun.	If you suspend or resume a flow that contains local jobs, the local jobs will also be suspended or resumed.
Viewing runtime attributes	You can view a local job's runtime attributes in Flow Manager. Note, however, that no resource usage is available for the local job.	You can view the exit status and CPU usage of a local job after the job completes. The process ID identifies the local job and you can view CPU usage for the job. You can also view the process ID of the job and CPU usage information with <code>jflows -l flow_id</code> and <code>jhists -C job</code> .
Timeout for local jobs	By default, a local job has a timeout so it will be killed if it was running for too long.	By default, a local job can run indefinitely; it does not have a timeout. The Process Manager administrator can, however, define a timeout value for a local job and it will be killed if it the job was running for too long.

1. Click the Insert Local Job button to put the Flow Editor in local job placement mode—when you left-click in the workspace, a local job icon appears.
2. Left-click in the workspace in the location where you want to insert the local job.
3. Draw the lines describing any dependencies the manual job has on other work items in the flow definition.

Define local job details

After placing a local job in the flow diagram, you can open and edit its definition as you would for other jobs.

1. Double-click on the local job. The Job Definition dialog appears.

2. Specify the local job details.

The following fields are mandatory

- In the Name field, specify a unique, meaningful name for the local job. The default value is *LJnumber* (where *number* is the total number of local jobs inserted into the same flow) and is automatically specified when you first define the local job. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_) in the local job name.
- In the Command to run field, specify the actual command to run, including the file path to the command and its arguments.
- In the User name field, specify the user name under which the command is run. If you are not one of the Platform Process Manager administrators, you must specify your own user name. The default is your own user name.

The following fields are optional. If undefined, Platform Process Manager uses the default values:

- In the Working Directory field, specify the directory in which the command is run. The default is the home directory of the user under which the command is run.
- In the Environment Variables field, specify a list of extra key-value pairs to be set up in the environment for use by the command. This field cannot be edited manually — you must click Modify and use the dialogs to specify the key-value pair. The default is undefined.
- In the Non-zero success exit codes field, specify a list of space-separated numbers from 1 to 255. Use this field to indicate which numbers in addition to 0 represent success for the job.
- In the Enter Description field, add any descriptive text that may be used for managing this job within the flow. The default is undefined.

You can specify user variables in all fields except the Name and Description fields. For the Environment Variables field, you can only specify user variables in the value section of the key-value pair.

3. Click OK to save your changes.

Running a local job

When Platform Process Manager triggers a flow containing a local job and the local job's dependencies are met, the local job will start to run.

The local job can be depended upon based on the following types of conditions:

- Completes successfully: The job completes with exit code 0, or any other number specified in the Non-zero success exit codes field.
- Fails: The job fails.
- Ends with any exit code: The job exits with any exit code, including 0.
- Ends with exit code: The job exits with a particular exit code pattern. For example, not-equal-to, equal-to.

Variables in Platform Process Manager

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.

You can use variables as part or all of a file name to make file names flexible, or you can use them to pass arguments to and from scripts. You can export the value of a variable to one or more jobs in a flow, or to other flows that are currently running on the same Platform Process Manager Server. You can also use variables in the index expression of a job array definition, in the message sent when a manual job requires completion, or when a job runs.

Types of variables

Platform Process Manager supports three types of variables:

- User variables
- Built-in variables
- Environment variables

User variables

User variables are those defined by the user, where the value is set within a job, job array, flow, or subflow, and made available to Platform Process Manager. User variables can be defined inside environment variables.

Built-in variables

Built-in variables are those defined by the Platform Process Manager system, where the value is obtained by Platform Process Manager and made available for use by a flow.

Environment variables

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

Platform Process Manager built-in variables

Currently, Platform Process Manager provides the following built-in variables:

%I

You use the built-in variable %I to obtain the index of a job array element. You can use %I in the following fields:

- On the Job Array Definition—Edit Job dialog, General tab, in the following fields:
 - Input file
 - Output file
 - Error file

Usage

Specify the variable as follows when you want to use its current value:

file_name.%I

Do not specify brace brackets and # sign.

%J

You use the built-in variable %J to obtain the job ID of a job array. You can use %J in the following fields:

- On the Job Array Definition—Edit Job dialog, General tab, in the following fields:
 - Output file
 - Error file

Usage

Specify the variable as follows when you want to use its current value:

file_name.%J

Do not specify # sign and brace brackets.

JS_EVENT[*n*]._FILENAME

You use the built-in variable JS_EVENT[*n*]._FILENAME when you need to use the name of the file that triggered this particular flow. If a flow is triggered by multiple files, multiple variables are created, each with a different value for *n*. The value of *n* is determined by the position of the triggering event in the list of possible flow-triggering events, including all types of events.

Consider the following examples, where a flow definition is submitted with multiple events that can trigger the flow.

Usage

Specify the variable as follows when you want to use its current value:

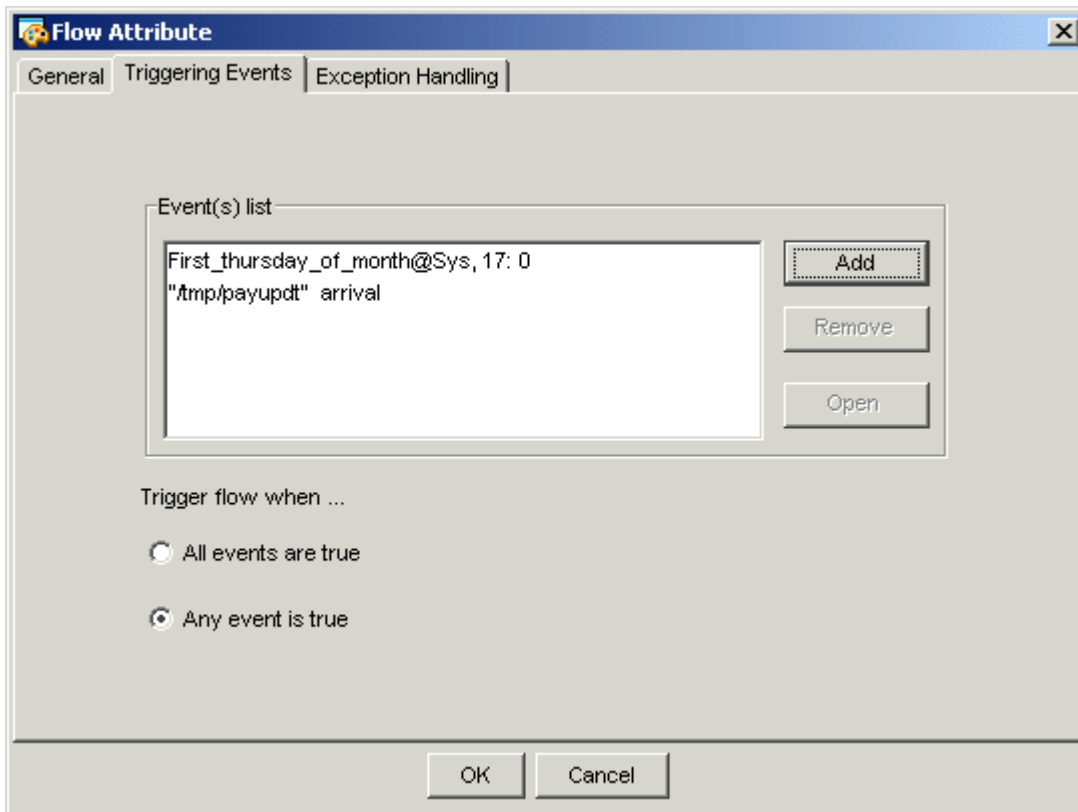
#{JS_EVENT[*n*]._FILENAME}

where *n* is the position of the triggering event in the list of possible events.

Example: one event at a time

In this example, myflow is triggered to run under either of the following conditions:

- At 5:00 p.m. on the first Thursday of the month (a time event)
- If a file called payupdt arrives in the tmp directory



When the file `/tmp/payupdt` arrives, the name and value of the built-in variable are as follows:

`JS_EVENT[2]_FILENAME=/tmp/payupdt`

If the flow is triggered by the time event, no value is set for `JS_EVENT[2]_FILENAME`.

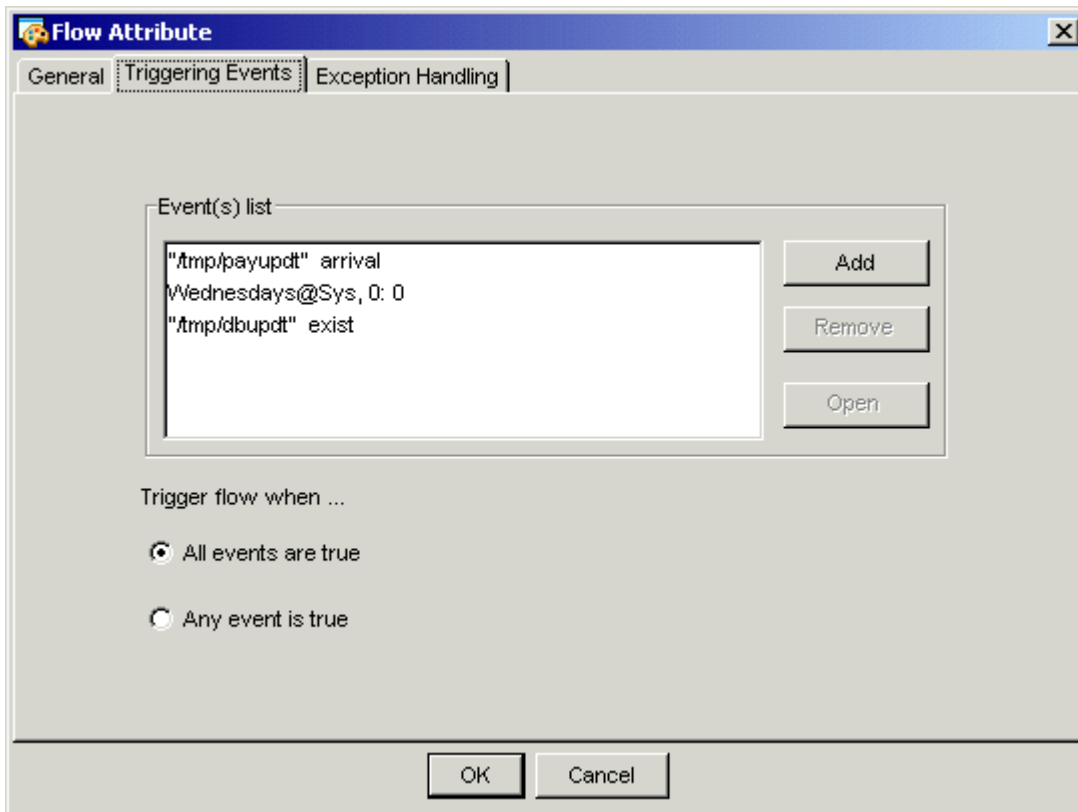
Note that the value of *n* in the name of the variable corresponds to the position of the file event in the list of events.

Example: multiple events

In this example, myflow is triggered to run when all of the following are met:

- A file called `payupdt` arrives in the `tmp` directory
- Today is Wednesday
- The file `/tmp/dbupdt` exists

Define your flow



In this example, all of the conditions must be met before the flow is triggered. When the flow triggers, the names and values of the built-in variables are as follows:

```
JS_EVENT[1]_FILENAME=/tmp/payupdt
```

```
JS_EVENT[3]_FILENAME=/tmp/dbupdt
```

Note that the value of *n* in the name of the variable corresponds to the position of the file event in the list of events.

JS_EVENT_n_FILENAME_BASE

The built-in variable JS_EVENT_n_FILENAME_BASE is the base file name, which is part of JS_EVENT[n]_FILENAME.

Usage

Specify the variable as follows when you want to use its current value:

```
{JS_EVENT_n_FILENAME_BASE}
```

JS_FLOW_ID

You use the built-in variable JS_FLOW_ID when you need to use the unique ID number of a flow.

Usage

Specify the variable as follows when you want to use its current value:

```
{JS_FLOW_ID}
```


JS_FLOW_NAME

You use the built-in variable `JS_FLOW_NAME` when you need to use the complete, unique name of a flow. The flow name is returned in the following format:

flow_ID:username.flowname

Usage

Specify the variable as follows when you want to use its current value:

`#{JS_FLOW_NAME}`

JS_FLOW_FULL_NAME

You use the built-in user variable `JS_FLOW_FULL_NAME` when you need to use the long version of a subflow name.

For example,

- For a subflow named `11: usr 1: F1: SF1: SSF1`, this variable is set to `11: usr 1: F1: SF1: SSF1`.
- For a main flow named `11: usr 1: F1`, this variable is set to `11: usr 1: F1`.
- For a flow array element named `11: usr 1: F1: FA(1)`, this variable is set to `11: usr 1: F1: 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.

Usage

`#{JS_FLOW_FULL_NAME}`

`#{JS_FLOW_FULL_NAME}(#{JS_FLOW_INDEX})`

JS_FLOW_SHORT_NAME

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 example, there are two dynamic subflows (`DSF1` and `DSF2`) in a main flow (`11: usr 1: 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: usr 1: F1: SF1: SSF1`, this variable is set to `SSF1`.
- For a main flow named `11: usr 1: F1`, this variable is set to `F1`.
- For a flow array element named `11: usr 1: 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.

Usage

Set the parent flow variable to `variable_name_#{JS_FLOW_SHORT_NAME}` in the job.

Define your flow

For example, for the `myvar` variable name, the DSF1 dynamic subflow will set `#{myvar_DSF1}` and the DSF2 dynamic subflow will set `#{myvar_DSF2}`. This allows both dynamic subflows to avoid variable name conflicts even though both dynamic subflows use the same target flow.

If you want to use an environment variable (such as `myvar_${JS_FLOW_SHORT_NAME}`, you must list `JS_FLOW_SHORT_NAME` as an input variable for the flow or job).

JS_ITERATION_COUNTER

You use the built-in variable `JS_ITERATION_COUNTER` when you have specified a rerun exception handler for a flow or subflow, and you need to know how many times the flow or subflow has been rerun. The first time the flow or subflow is run, the value of `JS_ITERATION_COUNTER` is 0. If the flow or subflow is rerun, the counter is incremented. For example, if the value of `JS_ITERATION_COUNTER` is 3, the flow or subflow has been rerun three times—it is running for the fourth time.

Usage

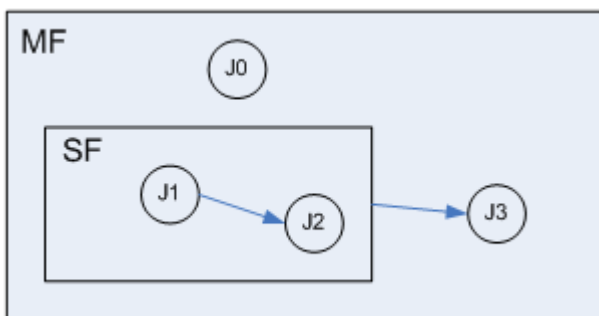
Specify the variable as follows when you want to use its current value:

```
#{JS_ITERATION_COUNTER}
```

Variable override order

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

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



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

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

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

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

Dynamic subflows

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

Include the variable evaluator to run jobs based on decision branches

The variable evaluator (VE) is a work item in the flow editor that allows jobs to depend on the evaluation of variable expressions instead of the traditional job exit status, time events, and file events. This work item contains no actual jobs to run, and therefore you cannot kill, suspend, or resume it. In addition, since the work item does not perform any actual work, you cannot use the variable evaluator as a triggering event, a proxy event to start the next work item, or as a flow completion criterion.

The variable evaluator serves as an intermediate step between jobs and the validation of variable decision branches. Typically, the predecessors of a variable evaluator assign values to various user variables. When all the variables are set, the variable evaluator will evaluate all of its variable expression branches and determine which successors to start executing. If there is no predecessor to the variable evaluator, the variable evaluator will start and run to completion immediately.

When you specify the variable expressions for each branch, you can use a combination of variables, operators, and constants. The variable evaluator supports the following operators: <, >, =, >=, <=, !=

The basic variable expression consists of one variable, one operator, and one constant: **variable operator constant**. For example, **#{A} > 1**.

You can also create larger and more complex expressions by joining smaller expressions using the following boolean operators:

&& (AND), **||** (OR), **!** (NOT), **()** (parentheses). However, you can only apply **!** (NOT) to variable expressions and not to literals. That is, **!({#A} > {#B})** is valid while **!({#A}) > {#B}** is not.

For example, the following are all valid combinations of variable expressions:

```
#{A} < 4 && #{B} > 3
!({#A} < 4 || #{B} > 4)
!({#A} > 2 && #{B} > 3) || #{C} > 5
```

You can also specify an else branch decision. The variable evaluator only evaluates else branches to TRUE if all other non-else branches evaluate to FALSE.

1. Click the Variable Evaluator button to put the Flow Editor in variable evaluator placement mode.
2. Left-click in the workspace in the location where you want to insert the variable evaluator.

The variable evaluator icon is added to the flow diagram.

3. If you want to include a name or description of the variable evaluator, double-click the variable evaluator icon that you added and specify these fields.
4. Draw the lines linking any predecessors (dependencies) the variable evaluator has on other work items in the flow definition.

The predecessors will likely have variable definitions on which the variable evaluator will make decisions.

If you do not define any dependencies for the variable evaluator, the variable evaluator will be triggered immediately when the flow runs.

5. Draw the lines linking the variable evaluator to the decision branching jobs that depend on the variable evaluator.

In the flow editor, the decision branching can be drawn as a link object from the variable evaluator to any successive jobs. The link dependencies definition has the following options:

- Specify a variable expression

The variable evaluator branches to this job if the variable expression that you specify is TRUE. Specify a variable expression or a combination of variable expressions.

If you specify multiple branches that evaluate to TRUE in a workflow, they will all run their next jobs.

Note:

If, at runtime, a variable is undefined and thus cannot be resolved to a value, the variable evaluator evaluates the corresponding sub-expression to FALSE; however, the whole expression might not be evaluated to FALSE.

- else

The variable evaluator branches to this job if all the other variable expressions are FALSE.

If you specified multiple else statements in a workflow, they will all run their next jobs if the all of the other (non-else) variable expressions are FALSE.

6. Click OK.

Use global variables

1. In the Flow Manager, from the View menu, select Global Variables. The list of global variables and their values is displayed.

Add a global variable

1. In the Flow Manager, from the View menu, select Global Variables.
2. Click Add in the Global Variables window. The Add Variable window is displayed.
3. In the Name field, specify a unique variable name. You can use alphabetic characters, numerals 0 to 9, space, and underscore () in the variable name.

Note:

Name cannot start with a numeric character.

4. Set a value to the variable.

Note:

Do not include equal or semicolon characters in the value.

5. Click OK to add the global variable. The added variable can be referenced in a flow instance.

Remove a global variable

1. In the Flow Manager, from the View menu, select Global Variables.

2. Select a variable and click Remove in the Global Variables window.

Edit a global variable

1. In the Flow Manager, from the View menu, select Global Variables.
2. Click Edit in the Global Variables window. The Edit Variable window is displayed.
3. Set another value to the existing variable.

Note:

Do not include equal or semicolon characters in the value.

4. Click OK. The edited variable can be referenced in a flow instance.

Use a user variable

1. Define a job that sets a runtime value for the variable, ensuring that the value will be available to Platform Process Manager before the job that needs the value runs.
2. Submit the job to a queue that has been configured for setting user variables. See your Platform Process Manager administrator for queue names.
3. Define the flow or work item that needs the value of the variable. In the appropriate input field, specify the variable in the following format:

`#{variable}`

For example:

The screenshot shows a dialog box titled "Define the Job". It has two input fields. The first field is labeled "Name:" and contains the text "J1". The second field is labeled "Command to run:" and contains the text "#{execPATH}/paydata".

Value of a variable

You can use a variable in the following places in the Flow Editor:

- On the Job/Job Array Definition—Edit Job dialogs, General tab, in the following fields:
 - Name—applies only to jobs, not job arrays

Note:

If you used a user variable to describe the job name, and your job is not in a flow array, you must ensure that your flow is arranged such that the user variable is resolved before triggering the flow.

This is because if the user variable is not resolved when the flow is triggered, the flow will fail to be triggered unless the job is in a flow array. Once the job name is resolved with the user variable, the job name will not change later even if the value of the user variable changes.

- Command to run

- Index expression—applies only to job arrays
- Queue name
- Project name
- Input file
- Output file
- Error file
- Email address
- Run as user name
- On the Job Definition—Edit Job dialog, Processing tab, in the following fields:
 - Host name
 - Priority
- On the Job Definition—Edit Job dialog, Resources tab, in the following field:
 - Expression
- On the Job Definition—Edit Job dialog, File Transfer tab, in the following fields:
 - Local path including name
 - File on execution host
- On the File Event Definition dialog, in the following field:
 - File name
- On the Manual Job Definition dialog, in the following field:
 - Message
- On the Alarm Definition dialog, in the following field:
 - Description
- On the Flow Attribute dialog, in the following field:
 - Email address

Set a user variable in a Windows bat file

Within the batch file, set the values and scopes of multiple variables by specifying the files containing these variables. The jobs write to the files in the following format, with each line containing a variable-value pair:

```
VARIABLE1=VALUE1  
VARIABLE2=VALUE2  
...
```

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

For job arrays, you must append the `LSB_JOBINDEX` environment variable to the file names to indicate the index of each job array element.

1. Define a job that runs a batch file, or wraps the command to run within a batch file.
2. Within the batch file, set the scope of the variable by specifying the variable-value pair in the scope-specific file, as follows:
 - a) To set local variables, whose values are not available outside the scope of this flow (or subflow), from a file, use the `JS_FLOW_VARIABLE_FILE` environment variable to access the file.
 - To set a local variable-value pair for a job, append to the `%JS_FLOW_VARIABLE_FILE%` file:
echo variable=value > %JS_FLOW_VARIABLE_FILE%

- To set a local variable-value pair for a job array, append to the `%JS_FLOW_VARIABLE_FILE% [%LSB_JOBINDEX%]` file:


```
echo variable=value > %JS_FLOW_VARIABLE_FILE%[%LSB_JOBINDEX%]
```
- b) To set global variables, whose values are available to all flows within the Platform Process Manager Server, from a file, use the `JS_GLOBAL_VARIABLE_FILE` environment variable to access the file.
 - To set a global variable-value pair for a job, append to the `%JS_GLOBAL_VARIABLE_FILE%` file:


```
echo variable=value > %JS_GLOBAL_VARIABLE_FILE%
```
 - To set a global variable-value pair for a job array, append to the `%JS_GLOBAL_VARIABLE_FILE% [%LSB_JOBINDEX%]` file:


```
echo variable=value > %JS_GLOBAL_VARIABLE_FILE%[%LSB_JOBINDEX%]
```
- c) To set parent flow variables, whose values are available to the scope of the parent flow for this flow (or subflow), from a file, use the `JS_PARENT_FLOW_VARIABLE_FILE` environment variable to access the file. If this flow is the main flow, the parent flow is also the main flow.
 - To set a parent variable-value pair for a job, append to the `%JS_PARENT_FLOW_VARIABLE_FILE%` file:


```
echo variable=value > %JS_PARENT_FLOW_VARIABLE_FILE%
```
 - To set a parent variable-value pair for a job array, append to the `%JS_PARENT_FLOW_VARIABLE_FILE% [%LSB_JOBINDEX%]` file:


```
echo variable=value > %JS_PARENT_FLOW_VARIABLE_FILE%[%LSB_JOBINDEX%]
```

Platform Process Manager sets the file environment variables as follows:

- Platform Process Manager sets `%JS_FLOW_VARIABLE_FILE%` to `JS_HOME\work\var_comm\flowvar.job_name`.
- Platform Process Manager sets `%JS_GLOBAL_VARIABLE_FILE%` to `JS_HOME\work\var_comm\globalvar.job_name`.
- Platform Process Manager sets `%JS_PARENT_FLOW_VARIABLE_FILE%` to `JS_HOME\work\var_comm\parentflowvar.job_name`.

Within the appropriate scope, Platform Process Manager reads these files and records the variable-value pairs to the corresponding variable list environment variable (for example, `%JS_FLOW_VARIABLE_LIST%` for local variables or `%JS_GLOBAL_VARIABLE_LIST%` for global variables)

Set a user variable in a UNIX script

Within the script, set the values and scopes of multiple variables by specifying the files containing these variables. The jobs write to the files in the following format, with each line containing a variable-value pair:

```
VARIABLE1=VALUE1
VARIABLE2=VALUE2
...
```

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

For job arrays, you must append the `LSB_JOBINDEX` environment variable to the file names to indicate the index of each job array element.

1. Define a job that runs a script, or wraps the command to run within a script.
2. Within the script, set the scope of the variable by specifying which list of variables to create, as follows:

- a) To set local variables, whose values are not available outside the scope of this flow (or subflow), from a file, use the `JS_FLOW_VARIABLE_FILE` environment variable to access the file.
 - To set a local variable-value pair for a job, append to the `$JS_FLOW_VARIABLE_FILE` file:


```
echo variable=value > $JS_FLOW_VARIABLE_FILE
```
 - To set a local variable-value pair for a job array, append to the `$JS_FLOW_VARIABLE_FILE\ [$LSB_JOBINDEX\]` file:

```
echo variable=value > $JS_FLOW_VARIABLE_FILE[$LSB_JOBINDEX]
```

The following is a sample perl script for jobs to set flow variables:

```
#!/bin/perl
$flowVarFile = $ENV{JS_FLOW_VARIABLE_FILE};
open (OUT, ">$flowVarFile") || die "Can't open $flowVarFile: $!\n";
print OUT "Local Var1=value1\n";
print OUT "Local Var2=\"value2 with value\"\n";
close(OUT);
```

- b) To set global variables, whose values are available to all flows within the Platform Process Manager Server, from a file, use the `JS_GLOBAL_VARIABLE_FILE` environment variable to access the file.
 - To set a global variable-value pair for a job, append to the `$JS_GLOBAL_VARIABLE_FILE` file:


```
echo variable=value > $JS_GLOBAL_VARIABLE_FILE
```
 - To set a global variable-value pair for a job array, append to the `$JS_GLOBAL_VARIABLE_FILE\ [$LSB_JOBINDEX\]` file:

```
echo variable=value > $JS_GLOBAL_VARIABLE_FILE[$LSB_JOBINDEX]
```

The following is a sample perl script for jobs to set global variables:

```
#!/bin/perl
$globalVarFile=$ENV{JS_GLOBAL_VARIABLE_FILE};
open (APP, ">$globalVarFile") || die "Can't open $globalVarFile: $!\n";
print APP "Global Var1=Gvalue1\n";
print APP "Global Var2=\"Gvalue2 with space\"\n";
close(APP);
```

Platform Process Manager sets the `$JS_GLOBAL_VARIABLE_FILE` environment variable to `JS_HOME/work/var_comm/global var. job_name`.

- c) To set parent flow variables, whose values are available to the scope of the parent flow for this flow (or subflow), from a file, use the `JS_PARENT_FLOW_VARIABLE_FILE` environment variable to access the file. If this flow is the main flow, the parent flow is also the main flow.
 - To set a parent flow variable-value pair for a job, append to the `$JS_PARENT_FLOW_VARIABLE_FILE` file:


```
echo variable=value > $JS_PARENT_FLOW_VARIABLE_FILE
```
 - To set a parent flow variable-value pair for a job array, append to the `$JS_PARENT_FLOW_VARIABLE_FILE\ [$LSB_JOBINDEX\]` file:

```
echo variable=value > $JS_PARENT_FLOW_VARIABLE_FILE[$LSB_JOBINDEX]
```

Platform Process Manager sets the file environment variables as follows:

- Platform Process Manager sets `$JS_FLOW_VARIABLE_FILE` to `JS_HOME/work/var_comm/flowvar. job_name`.
- Platform Process Manager sets `$JS_GLOBAL_VARIABLE_FILE` to `JS_HOME/work/var_comm/global var. job_name`.
- Platform Process Manager sets `$JS_PARENT_FLOW_VARIABLE_FILE` to `JS_HOME/work/var_comm/parentflowvar. job_name`.

Within the appropriate scope, Platform Process Manager reads these files and records the variable-value pairs to the corresponding variable list environment variable (for example, `$JS_FLOW_VARIABLE_LIST` for local variables or `$JS_GLOBAL_VARIABLE_LIST` for global variables)

Set a flow variable using the flow manager

View and set flow variables in flows, subflows, and flow arrays from the flow manager. In order to view and set flow variables for these items, the main flow must be either in a Running, Pending, Exit, Waiting, or Suspended state, and the flow item itself (the flow, subflow, or flow array) must be in an Exit state.

1. From the flow manager, right-click on an Exited flow, subflow, or a flow array, and select Set Flow Variables.

The Flow Variables dialog displays. This dialog shows a list of the flow variable names and values for the selected flow item.

2. Modify the flow variables in the selected flow item.

- To add a new flow variable, click Add.
- To delete a flow variable, select the flow variable and click Remove.
- To modify an existing flow variable, select the flow variable and click Edit.

Note:

Platform Process Manager does not currently support editing flow variables for a flow, subflow, or flow array if the corresponding main flow that is in a **Done** or **Killed** state.

3. Save or discard your changes.
 - To save the flow variable changes and close the Flow Variables dialog, click OK.
 - To discard the flow variable changes and close the Flow Variables dialog, click Cancel.
 - To save the flow variable changes and continue editing variables, click Apply.

User variables within a flow definition

When defining a job that sets one or more variables, ensure that you specify a queue that is configured to support user variables.

Tip:

Before you can set or use user variables in a flow, your Platform Process Manager system needs to have one or more queues configured to accept them. Check with your Platform Process Manager administrator to see which queues are configured to support variables.

Types of user variables you can set

There are two types of variables you can set:

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

Setting the value of a user variable

You can set the value of one or more variables as follows:

Define your flow

- On UNIX, by setting the value within a script
- On Windows, by setting the value within a bat file

Multiple variables in a list

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

When the value of a variable is evaluated

The value of a variable is resolved just before the job or job array is dispatched for execution. If the variable is used in a file event, the value is resolved periodically, when the condition of the event is evaluated.

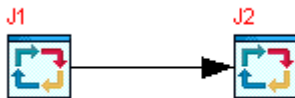
When a variable is used for a manual job message, the value of the variable is resolved just before the email is sent.

Job dependencies

When you draw a line between two work items in the flow diagram, you are establishing dependencies between the two jobs. The default type of dependency is assigned, which is a dependency on the first job to complete successfully. However, you may want a job to run only after a predecessor job has failed, or when a job completes with a particular exit code. In these cases, you need to edit the job dependency.

You can choose from the following criteria:

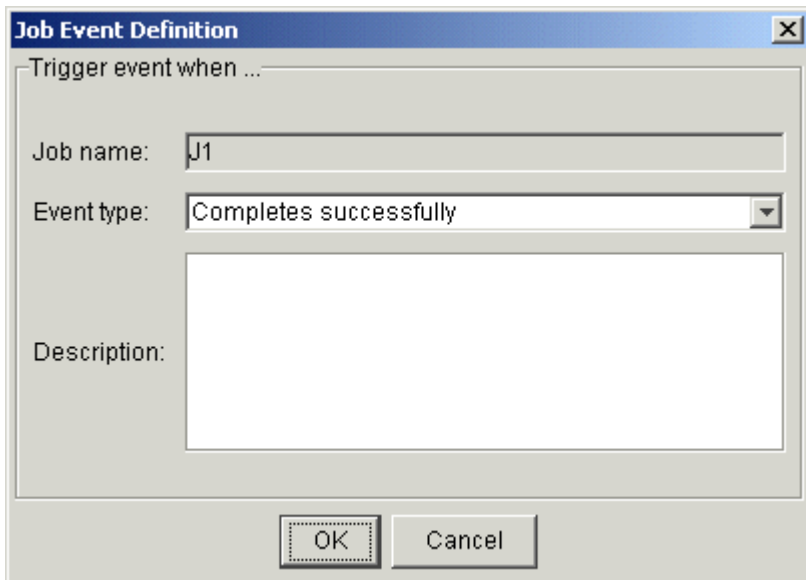
- Run a job when the predecessor completes successfully
 - Run a job when the predecessor job starts
 - Run a job when the predecessor is submitted
 - Run a job when the predecessor job fails
 - Run a job when the predecessor job ends, regardless of success or failure
 - Run a job when the predecessor job overruns
 - Run a job when the predecessor underruns
 - Run a job if the predecessor fails to start
 - Run a job when the predecessor cannot run
 - Run a job if the predecessor misses its scheduled start time
1. Draw both the predecessor job and the job that succeeds it.
 2. Change to job dependency mode by clicking the Insert Dependency button.
 3. Draw job dependencies by left-clicking on the job that must run first, then left-clicking on the job that runs next.



Job J2 cannot run until job J1 completes successfully. J2 cannot run until the dependency condition is met.

4. To change the type of dependency, right-click on the dependency line and select Open Definition. The Event Definition dialog box appears.

Define your flow

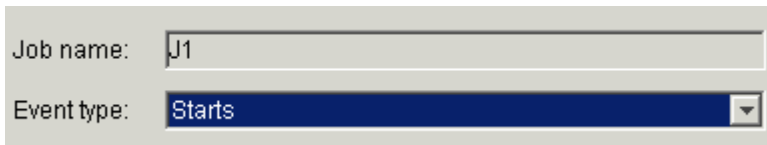


The image shows a dialog box titled "Job Event Definition". It has a close button (X) in the top right corner. The main area is labeled "Trigger event when ...". It contains three fields: "Job name:" with the text "J1", "Event type:" with a dropdown menu showing "Completes successfully", and "Description:" with an empty text area. At the bottom, there are two buttons: "OK" and "Cancel".

5. In the Event Type field, select the type of dependency you want to use to trigger the successor job, and the appropriate operator and values. See the examples that follow for job dependencies you can use.
6. In the Description field, add any descriptive text that may be used for understanding this event. For example, if this event requires special instructions for operations staff, place those instructions here.
7. Click OK.

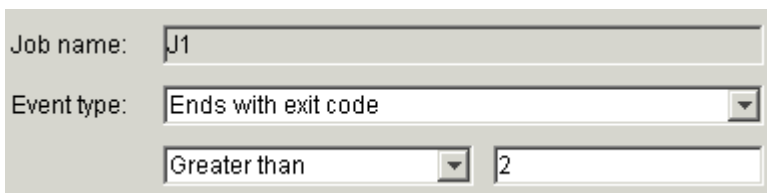
Examples

Run a job when predecessor starts



This is a screenshot of the "Job Event Definition" dialog box. The "Job name:" field contains "J1". The "Event type:" dropdown menu is open, and "Starts" is selected and highlighted in blue.

Run when predecessor has exit code greater than 2



This is a screenshot of the "Job Event Definition" dialog box. The "Job name:" field contains "J1". The "Event type:" dropdown menu is open, and "Ends with exit code" is selected. Below this, there is a second dropdown menu showing "Greater than" and a text input field containing the number "2".

Run when the predecessor's exit code is 10, 15, or 22

You can specify multiple exit codes to indicate to run when any of these exit codes are encountered. You specify a space-separated list of exit codes in numbers from 0 to 255.

Job name: J2

Event type: Ends with exit code

Equal to 10 15 22

Run when the predecessor's exit code is not 9 or 11

You can specify multiple exit codes to indicate to run when any of these exit codes are not encountered. You specify a space-separated list of exit codes in numbers from 0 to 255.

Job name: J2

Event type: Ends with exit code

Not equal to 9 11

Specify dependency on the start or submission of specific jobs

By default, when you establish a dependency on a job to complete successfully, dependent jobs are not submitted until dependencies are satisfied. For example, job124 depends on job 123. Job 123 is submitted, and job 124 is not submitted until job 123 completes.

In some cases, you may have a data preparation job that takes a long time, followed by a computation job. If you have a busy cluster, if your second job gets submitted after the first job has completed, your second job may be waiting a long time in the queue to be scheduled. For these kinds of cases, you can specify to Process Manager to pre-submit dependent jobs, reducing the time a job waits in the queue to be scheduled.

Examples:

- You want your job to be started as early as possible

For example, you have job 123 followed by job 124. You want job 124 to be submitted as early as possible. In this case, you pre-submit job 124 and specify the Is Submitted dependency. Job 124 will be submitted right after job 123 has been submitted to LSF.

- The next job cannot start until the execution host is known for the previous job

For example, job124 needs to run on the same host as the preceding job. You pre-submit job 124 and specify the Starts dependency. Job 124 will be submitted right after job 123 has started to run in LSF.

Requirements to pre-submit dependent jobs

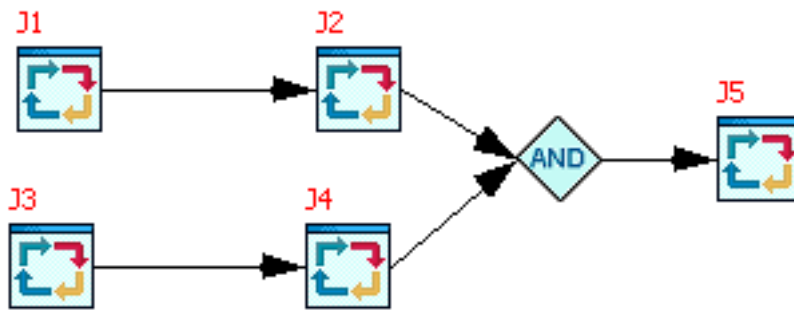
- Only LSF jobs, job arrays, job scripts, job array scripts, and template jobs can be pre-submitted.
- Only jobs, job scripts, job arrays, job array scripts, and template jobs can be preceding jobs to the dependent job to be pre-submitted.
- The jobs to be pre-submitted must be direct links. They cannot be more than one link away.
- The dependencies of all predecessors must be logically connected with AND.
- In Flow Editor, the Event type in the Job Event Definition for the preceding jobs to the other job must be set to Starts or Is Submitted.
- If you specify dependent jobs to be pre-submitted, and the condition is never met, it is possible for the flow to be “stuck”. To handle this, define an overrun exception handler to kill the last job if it runs or pends for more than a certain period of time.

Examples

Example: Simple flow, pre-submission with "Starts" dependency

In this flow, you can only specify J2 and J4 to pre-submit J5. As a result, J5 will be submitted right after J2 and J4 start to run in LSF.

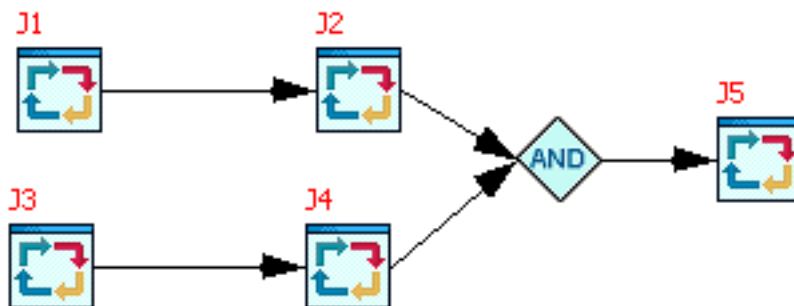
J1 and J3 cannot be considered because they are more than one link away from J5.



Example: Simple flow, pre-submission with "Is submitted" dependency

In this flow, you can only specify J2 and J4 to pre-submit J5. As a result, J5 will be submitted right after J2 and J4 are submitted to LSF.

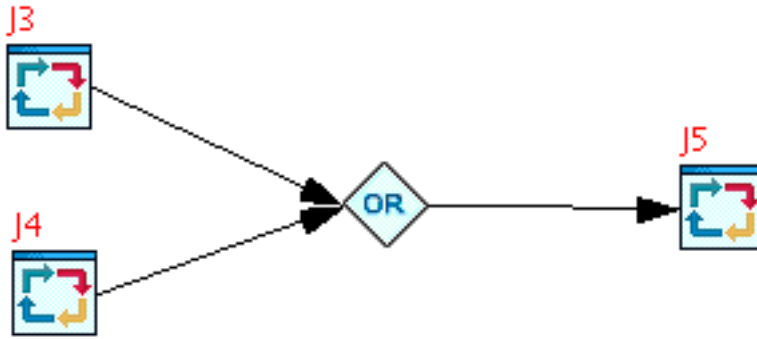
J1 and J3 cannot be considered because they are more than one link away from J5.



Example: Pre-submission not possible

In this flow, you cannot specify any pre-submission. This is because jobs must be logically connected with AND.

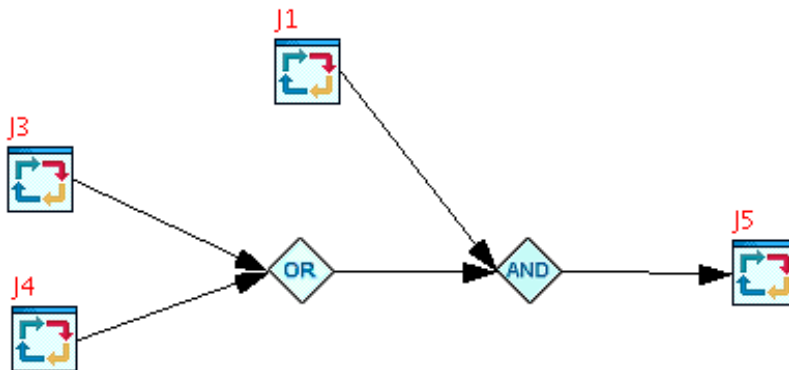
Define your flow



Example: Complex flow with "Starts" dependency

In this flow, you can only specify J1 to pre-submit J5. As a result, J5 will be submitted right after J1 starts to run in LSF.

J3 and J4 cannot be considered because they are more than one link away from J5, and there is also a logical OR.



How to submit a dependent job after selected jobs start running

1. Draw both the predecessor job and the job that succeeds it.
2. Change to job dependency mode by clicking the Insert Dependency button.
3. Draw job dependencies by left-clicking on the job that must run first, then left-clicking on the job that runs next.
4. To change the type of dependency, right-click on the dependency line and select Open Definition.
The Event Definition dialog box is displayed.
5. In the Event Type field, select Starts.
6. Click OK.
7. Double-click the dependent job that depends on other jobs to start.

The Job Definition dialog is displayed.

8. Select the Advanced tab.
9. In the Pre-submit section, select jobs from the Available column and click the Add button to put them in the Selected column.

The current job will be submitted right after the selected jobs start running in LSF.

How to submit a dependent job after selected jobs are submitted

1. Draw both the predecessor job and the job that succeeds it.
2. Change to job dependency mode by clicking the Insert Dependency button.
3. Draw job dependencies by left-clicking on the job that must run first, then left-clicking on the job that runs next.
4. To change the type of dependency, right-click on the dependency line and select Open Definition.

The Event Definition dialog box is displayed.

5. In the Event Type field, select Is Submitted.
6. Click OK.
7. Double-click the dependent job that depends on other jobs to start.

The Job Definition dialog is displayed.

8. Select the Advanced tab.
9. In the Pre-submit section, select jobs from the Available column and click the Add button to put them in the Selected column.

The current job will be submitted right after the selected jobs are submitted to LSF.

Specify a dependency on a file

Sometimes you do not want a work item within a flow to run until something happens to a particular file. You can specify the following circumstances:

- The file arrives
- The file exists (includes is created)
- The file does not exist (includes is deleted)
- The file size meets a certain criteria
- The file is updated within a certain time period

When you specify a file event for a job within a flow, that job will run once, when the file meets the specified condition. Even if the flow is still active the next time that file meets the specified condition, the file event triggers the job only once.

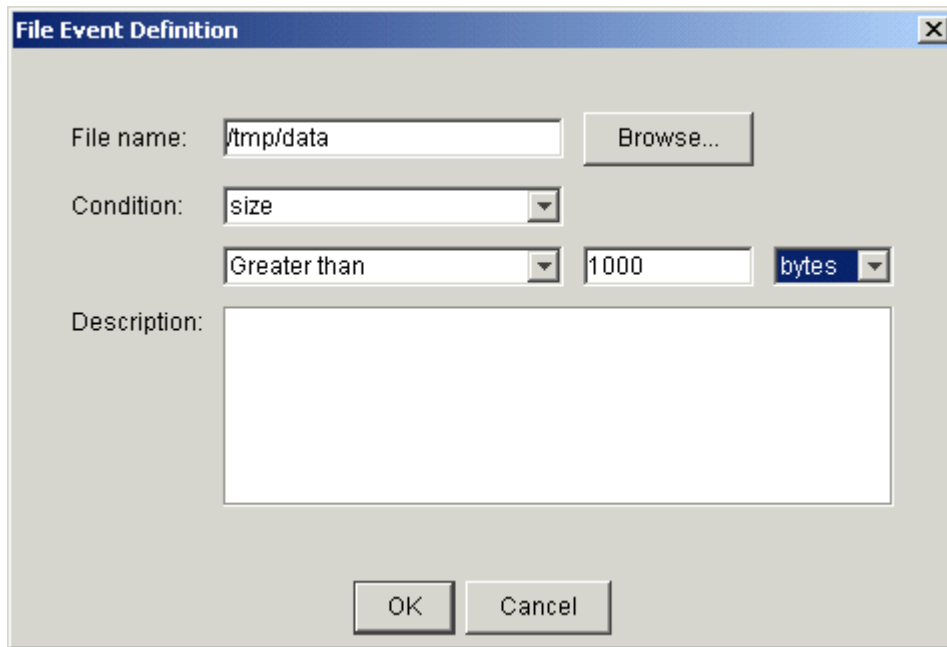
1. Change to file event mode by clicking the Insert File Event button.
2. Click in the workspace where you want to insert the file event. The file event icon does not yet appear in the workspace. The Event Definition dialog box appears.
3. In the File name field, specify the full path name of the file as the Platform Process Manager Server sees it, or click the Browse button to point to the file on which this event depends.

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.

You can also specify a variable for the file name, provided your system is configured to support them.

4. In the Condition field, choose the appropriate condition from the list, and specify the number of bytes if applicable. See the examples that follow this topic.
5. In the Description field, add any descriptive text that may be used for understanding this event. For example, if this event requires special instructions for operations staff, place those instructions here.



The image shows a dialog box titled "File Event Definition". It has a blue title bar with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- File name:** A text input field containing "/tmp/data" and a "Browse..." button to its right.
- Condition:** A dropdown menu currently showing "size".
- Operator:** A dropdown menu currently showing "Greater than".
- Value:** A text input field containing "1000".
- Unit:** A dropdown menu currently showing "bytes".
- Description:** A large empty text area.
- Buttons:** "OK" and "Cancel" buttons at the bottom center.

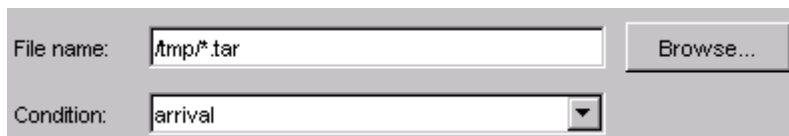
6. Click OK. The file event appears in the workspace, and you can draw the appropriate dependency lines to any jobs that are awaiting its arrival.

Note:

You can change the text of the label that appears above the file event in the workspace if the label text is too long.

Example

Run a job when a file arrives

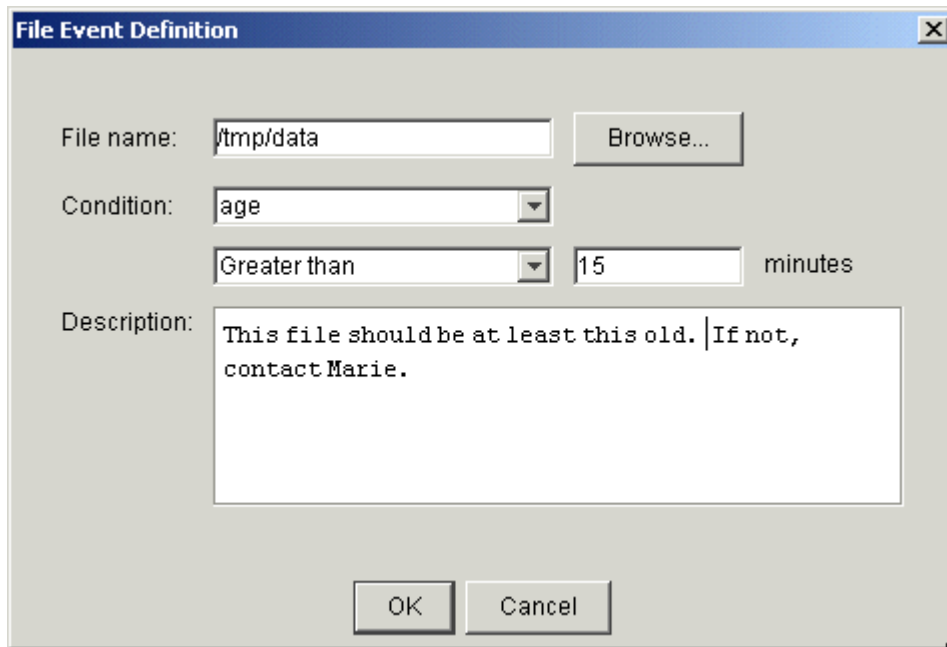


The image shows a simplified version of the "File Event Definition" dialog box. It contains the following fields and controls:

- File name:** A text input field containing "/tmp/*.tar" and a "Browse..." button to its right.
- Condition:** A dropdown menu currently showing "arrival".

Run a job if file is updated within a certain time period

Define your flow

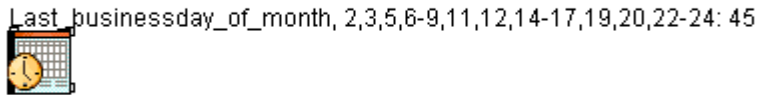


The image shows a dialog box titled "File Event Definition" with a close button in the top right corner. It contains the following fields and controls:

- File name:** A text input field containing the path `/tmp/data`, followed by a "Browse..." button.
- Condition:** A dropdown menu currently showing "age".
- Operator:** A second dropdown menu currently showing "Greater than".
- Value:** A text input field containing the number "15", followed by the text "minutes".
- Description:** A text area containing the text: "This file should be at least this old. |If not, contact Marie."
- Buttons:** "OK" and "Cancel" buttons at the bottom center.

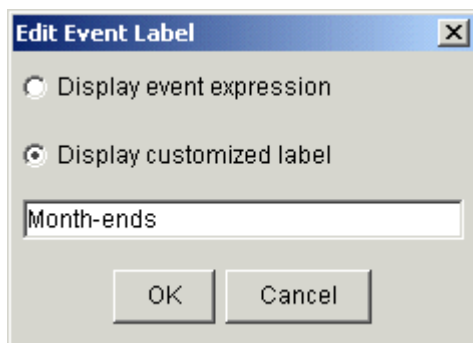
Change the label displayed for an event

Sometimes when you define an event, the label that appears in the workspace for the event is long and cumbersome, and interferes with the readability of the flow. This is because the label used is derived from the expression that defines the event. In a time event, for example, the label displays the calendar name and all the times the event will trigger. For example:



You can change the label for a file or time event by creating a customized label. That way, you can control the text that appears in the workspace for each event.

1. In the Flow Editor, right-click on the event whose label you want to change.
2. From the menu, select Edit Label. The Edit Event Label dialog appears.
3. Select Display customized label, and type the text you want to appear in the input field provided.



4. Click OK. The new label is now displayed.

Dependency on a date and time

While a flow may trigger at a particular time each day, or each week, you may want a work item within the flow to wait until after a specific time before it can run. For example, the flow Backup may run every night at midnight. However, within that flow, the job Report cannot be submitted until after 6:00 a.m. You can create a time event that tells Report to wait until 6:00 a.m. before it runs.

When you specify a time event for a job within a flow, that job will run once, when the combination of the date and time is true. Even if the flow is still active the next time that date and time combination is true, the time event triggers the job only once.

You can create the following types of dependencies using time events:

- You can specify a date and time when you want the job to run
- You can specify a particular frequency, such as daily, weekly or monthly, or at every nth interval, such as every 2 days, every 3 weeks or every 6 months
- You can specify a particular day of the week or month of the year
- You can combine calendar expressions to create complex scheduling criteria

When you create a time event, you point to a particular calendar, which defines the date component of the time event. To use a calendar, that calendar must first be defined.

1. Change to time dependency mode by clicking the Insert Time Event button.
2. Click in the workspace where you want to insert the time event. The Event Definition dialog box appears.
3. In the Calendar name field, specify the calendar that resolves to the dates on which you want this job to run.
4. In the Time zone section, specify the time zone for this time event.
5. In the Hours and Minutes fields, specify an expression that resolves to the time or times when you want the job to start running. Be sure to specify the time as it appears on a 24-hour clock, where valid values for hours are from 0 to 23.

Note:

Do not a time between 2:00 a.m. and 3:00 a.m. on the day that daylight savings time begins (the second Sunday in March), as the flow will not run and any subflows that are scheduled to start after this flow will also not run.

This is because the 2:00 a.m. to 3:00 a.m. hour is removed to start daylight savings time in North America.

6. In the Duration of event field, specify the length of time in minutes for which you want this event to be valid. This value, when added to the trigger time of the event, is the time by which this job must be submitted. After this time expires, the job will not be submitted. For example, if a job must run after 5 p.m. but cannot be submitted after 6 p.m., specify 17: 00 in the Time field, and 60 minutes in the Duration of event field.

Tip:

If you want to prevent a job from running after a particular time, and the job has a dependency on another event, ensure you use an AND link to combine the two events, not an OR link.

7. Optional. In the End after ... occurrences field, specify the maximum number of occurrences of this time event before you want it to end.
8. In the Description field, add any descriptive text that may be helpful for understanding this event. For example, if this event requires special instructions for operations staff, place those instructions here.

Specify the date by selecting a calendar and time zone then specify the time in hours and minutes.

Calendar Name:

Time zone:

The time zone of this host (Eastern Time)
 The time zone of the Process Manager server host
Current server time is Fri Jun 12 14:45:32 GMT-04:00 2009.
 UTC
 A specific time zone

Hours: (e.g. * or 1,6 or 1-7)

Minutes: (e.g. * or 10,40 or 20-30)

Duration of event: minutes

End after: occurrences

Description:

9. Click OK. The time event appears in the workspace, and you can draw the appropriate dependency lines to the job or jobs that are depending on this time.

Note:

You can change the text of the label that appears above the file event in the workspace if the label text is too long.

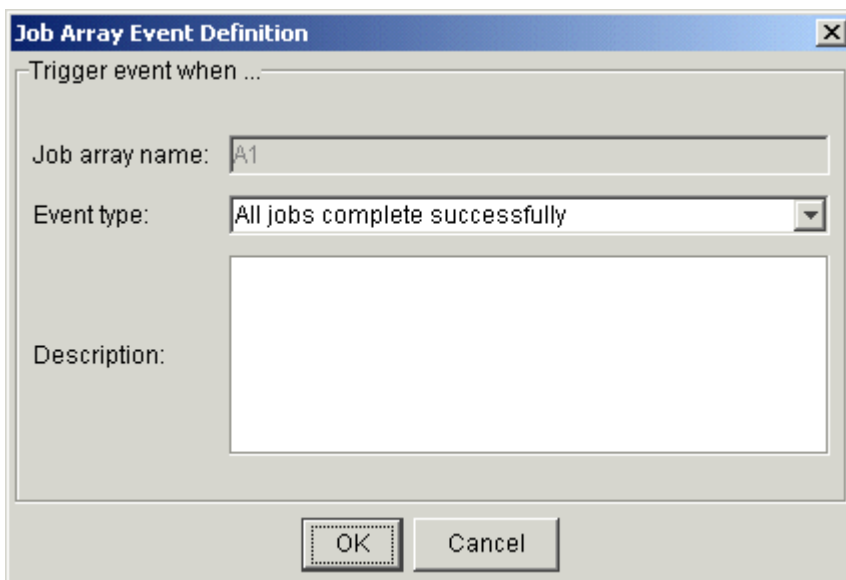
Specify dependencies on a job array

When you draw a dependency line from a job array to another work item in the flow definition, the dependency you create is the default: the work item cannot run until all of the jobs in the job array complete successfully.

You can specify the type of dependency to be one of the following:

- All jobs in the job array complete successfully—this is the default
- All jobs in the job array end, regardless of success or failure
- The sum of the exit codes of all the jobs in the job array has a value
- A specified number of jobs in the job array complete successfully
- A specified number of jobs in the job array fail
- A specified number of jobs in the job array end regardless of success or failure
- A specified number of jobs in the job array have started
- The job array runs longer than it should
- The job array runs an abnormally short length of time
- The job array fails to start
- The job array cannot run
- The job array misses its scheduled start time

1. Draw both the predecessor job array and the work item that succeeds it.
2. Change to job dependency mode by clicking the Insert Dependency button.
3. Draw a dependency line from the job array to the work item that depends on the array.
4. To change the type of dependency, right-click on the dependency line and select Open Definition. The Event Definition dialog box appears.



5. In the Event type field, select the type of dependency you want to use to trigger the successor job, and the appropriate operator and value if applicable. See the examples that follow for some of the job array dependencies you can use.

- In the Description field, add any descriptive text that may be helpful for understanding this event. For example, if this event requires special instructions for operations staff, place those instructions here.
- Click OK.

Examples

All jobs in the array end, regardless of success or failure

Job array name:

Event type:

Sum of the exit codes is...

Job array name:

Event type:

Number of successful jobs is...

Job array name:

Event type:

Number of unsuccessful jobs is...

Job array name:

Event type:

Example: number of jobs started is...

Job array name:

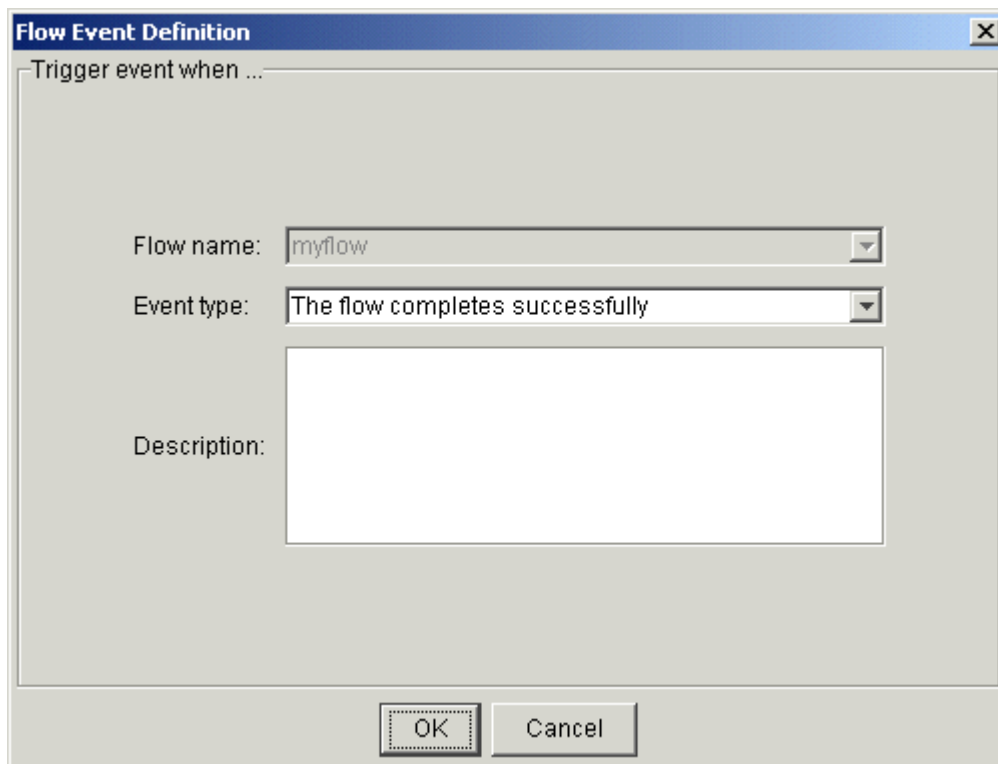
Event type:

Specify dependencies on a subflow

When you draw a dependency line from a subflow to another work item in the flow definition, provided that the subflow does not have a pre-defined exit condition, the dependency you create is the default: the work item cannot run until all of the jobs in the subflow complete successfully.

You can specify the type of dependency to be one of the following:

- All jobs in the subflow complete successfully—the default
 - All jobs in the subflow end, regardless of the exit code
 - The sum of the exit codes of all the jobs in the subflow has a value
 - A specified number of jobs in the subflow complete successfully
 - A specified number of jobs in the subflow fail
 - A specified number of jobs in the subflow end regardless of success or failure
 - A specified number of jobs in the subflow have started
 - The subflow runs longer than it should
 - The subflow runs for an abnormally short length of time
 - The subflow misses its schedule
1. Draw both the predecessor subflow and the work item that succeeds it.
 2. Change to job dependency mode by clicking the Insert Dependency button.
 3. Draw a dependency line from the subflow to the work item that depends on the subflow.
 4. To change the type of dependency, right-click on the dependency line and select Open Definition. The Event Definition dialog box appears.



The image shows a dialog box titled "Flow Event Definition". It has a close button (X) in the top right corner. The main area is labeled "Trigger event when ...". There are two dropdown menus: "Flow name:" with the value "myflow" and "Event type:" with the value "The flow completes successfully". Below these is a large empty text area labeled "Description:". At the bottom, there are two buttons: "OK" and "Cancel".

5. In the Event type field, select the type of dependency you want to use to trigger the successor job, and the appropriate operator and values. See the examples that follow for subflow dependencies you can use.
6. In the Description field, add any descriptive text that may be helpful for understanding this event. For example, if this event requires special instructions for operations staff, place those instructions here.
7. Click OK.

Examples

Sum of the exit codes has a specific value

Flow name:

Event type:

Specified number of jobs complete successfully

Flow name:

Event type:

Specified number of jobs fails

Use this case when you want to run a job if the subflow fails. If you want to trigger the event when a certain number of jobs fail, specify the number of jobs that must complete successfully. This trigger occurs in real time—as soon as the specified number is met, the event triggers. It does not wait until the flow completes to test the condition.

Flow name:

Event type:

Specified number of jobs have started

Flow name:

Event type:

Specify dependencies on an unconnected work item

Specify a dependency on a proxy job

You can specify a dependency on another flow, or a work item that is running within another flow, or on a work item elsewhere in the current flow by creating a proxy event. You can specify the following types of dependencies:

- When the work item is submitted
- When the work item starts
- When the work item ends successfully
- When the work item exits with any exit code
- When the work item exits with a specific exit code
- When the work item exits with any of the specified exit codes. You can specify a list of space-separated exit codes with the event type Ends with exit code... and Equal to and Not equal to. You can specify a list of exit codes for a proxy job, proxy template job, proxy job script, and proxy local job.
- When a specific number of jobs in a job array or subflow starts
- When a specific number of jobs in a job array or subflow ends
- When a specific number of jobs in a job array or subflow exits

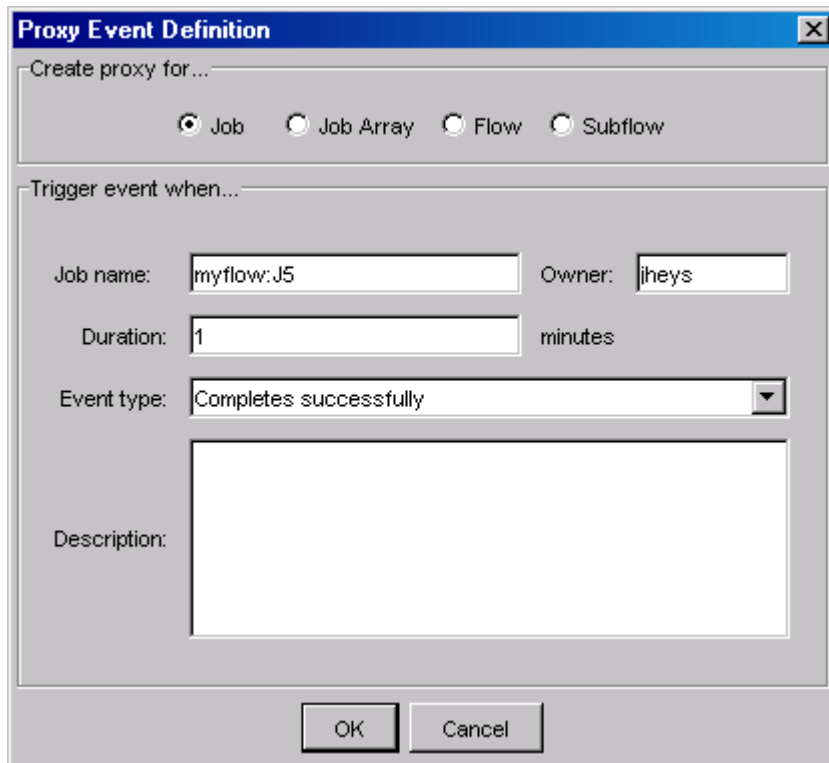
1. Change to proxy event mode by clicking the Insert Proxy Event button.
2. Click in the workspace where you want to insert the proxy event. The proxy event icon does not yet appear in the workspace. The Proxy Event Definition dialog box appears.
3. In the Create proxy for... box, leave the default at Job.
4. In the Job name field, specify the fully qualified name of the job, in the following format:

flow_name:subflow_name:job_name

If the job is not defined within a subflow, simply specify the flow name and the job name, separated by a colon.

Note: You cannot specify a proxy for a manual job.

5. If the flow containing the job is not owned by your user ID, in the Owner field, specify the user ID that owns the flow containing the proxy job.
6. In the Duration field, specify the number of minutes in the past to detect the proxy event.
7. In the Event type field, select the type of dependency you want to use to trigger the successor job, job array, subflow or flow, and the appropriate operator and values.
8. In the Description field, add any descriptive text that may be used for understanding this event.



The image shows a dialog box titled "Proxy Event Definition" with a close button (X) in the top right corner. The dialog is divided into two main sections: "Create proxy for..." and "Trigger event when...".

In the "Create proxy for..." section, there are four radio buttons: "Job" (selected), "Job Array", "Flow", and "Subflow".

In the "Trigger event when..." section, there are several input fields:

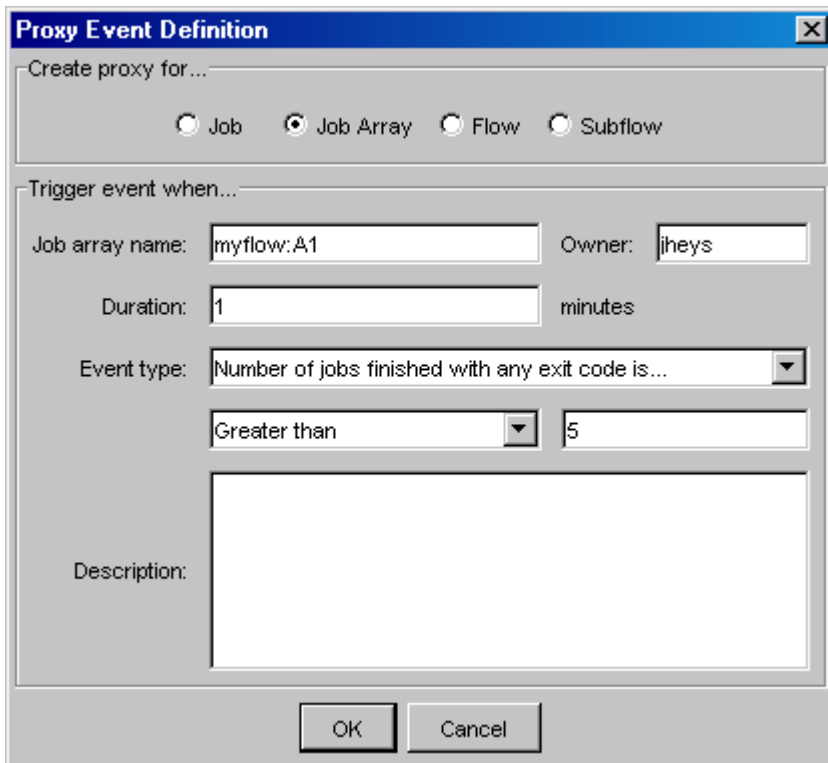
- "Job name:" with the text "myflow:J5" entered.
- "Owner:" with the text "ltheys" entered.
- "Duration:" with the text "1" entered, followed by the label "minutes".
- "Event type:" with a dropdown menu showing "Completes successfully".
- "Description:" with a large empty text area below it.

At the bottom of the dialog, there are two buttons: "OK" and "Cancel".

9. Click OK. The proxy event appears in the workspace, and you can draw the appropriate dependency lines to any work items.

Specify a dependency on a proxy job array

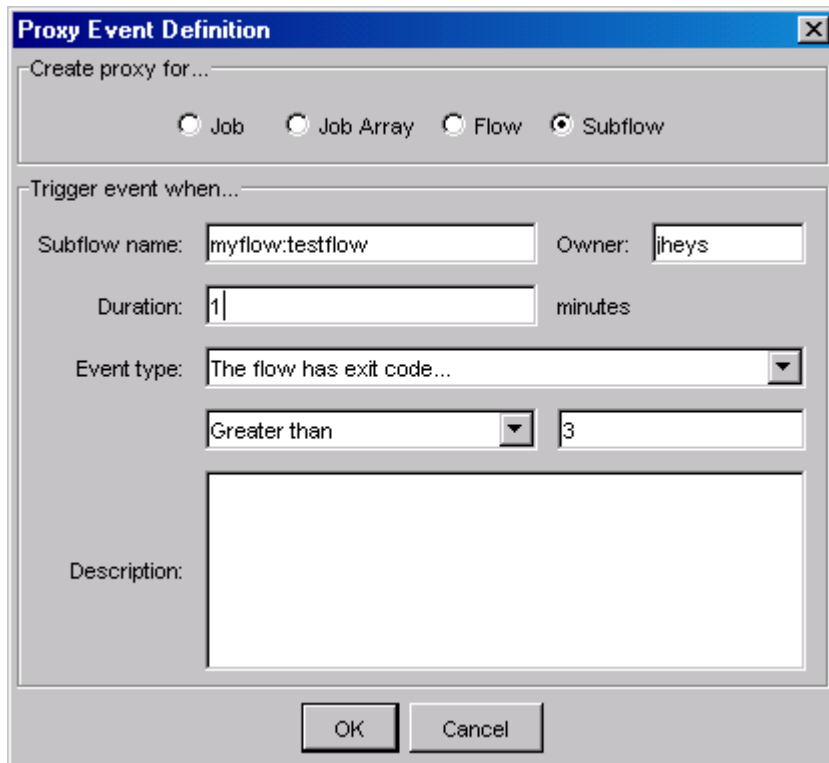
1. Change to proxy event mode by clicking the Insert Proxy Event button.
2. Click in the workspace where you want to insert the proxy event. The proxy event icon does not yet appear in the workspace. The Proxy Event Definition dialog box appears.
3. In the Create proxy for... box, select Job Array.
4. In the Job array name field, specify the fully qualified name of the job array, in the following format:
flow_name:subflow_name:job_array_name
 If the job array is not defined within a subflow, simply specify the flow name and the job array name, separated by a colon.
5. If the flow containing the job array is not owned by your user ID, in the Owner field, specify the user ID that owns the flow containing the proxy job array.
6. In the Duration field, specify the number of minutes in the past to detect the proxy event.
7. In the Event type field, select the type of dependency you want to use to trigger the successor job, job array, subflow or flow, and the appropriate operator and values.
8. In the Description field, add any descriptive text that may be used for understanding this event.



9. Click OK. The proxy event appears in the workspace, and you can draw the appropriate dependency lines to any work items.

Specify a dependency on a proxy subflow

1. Change to proxy event mode by clicking the Insert Proxy Event button.
2. Click in the workspace where you want to insert the proxy event. The proxy event icon does not yet appear in the workspace. The Proxy Event Definition dialog box appears.
3. In the Create proxy for... box, select Subflow.
4. In the Subflow name field, specify the fully qualified name of the subflow, in the following format:
flow_name:subflow_name
5. If the flow containing the subflow is not owned by your user ID, in the Owner field, specify the user ID that owns the flow containing the proxy subflow.
6. In the Duration field, specify the number of minutes in the past to detect the proxy event.
7. In the Event type field, select the type of dependency you want to use to trigger the successor job, job array, subflow or flow, and the appropriate operator and values.
8. In the Description field, add any descriptive text that may be used for understanding this event.



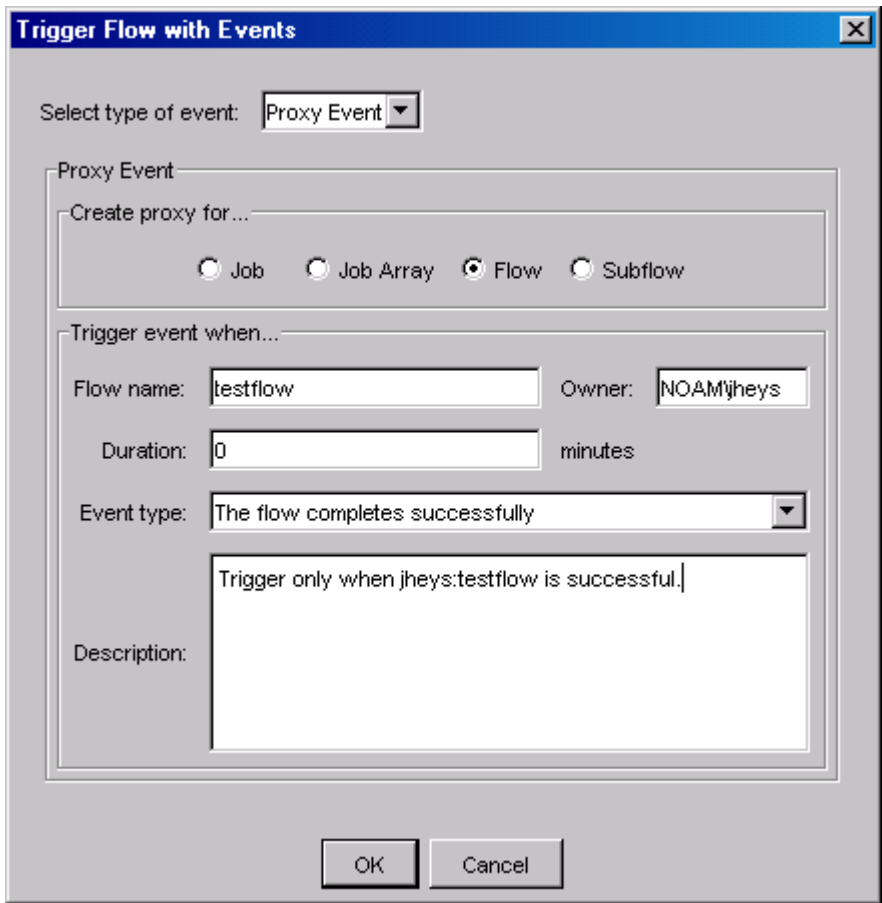
The image shows a 'Proxy Event Definition' dialog box with the following fields and options:

- Create proxy for...:** Radio buttons for Job, Job Array, Flow, and Subflow. 'Subflow' is selected.
- Trigger event when...:**
 - Subflow name: myflow:testflow
 - Owner: jheys
 - Duration: 1 minutes
 - Event type: The flow has exit code...
 - Operator: Greater than
 - Value: 3
 - Description: (empty text area)
- Buttons:** OK and Cancel.

9. Click OK. The proxy event appears in the workspace, and you can draw the appropriate dependency lines to any work items.

Specify a dependency on a proxy flow

1. Change to proxy event mode by clicking the Insert Proxy Event button.
2. Click in the workspace where you want to insert the proxy event. The proxy event icon does not yet appear in the workspace. The Proxy Event Definition dialog box appears.
3. In the Create proxy for... box, select Flow.
4. In the Flow name field, specify the name of the flow.
5. If the flow is not owned by your user ID, in the Owner field, specify the user ID that owns the flow.
6. In the Duration field, specify the number of minutes in the past to detect the proxy event.
7. In the Event type field, select the type of dependency you want to use to trigger the successor job, job array, subflow or flow, and the appropriate operator and values.
8. In the Description field, add any descriptive text that may be used for understanding this event.



9. Click OK. The proxy event appears in the workspace, and you can draw the appropriate dependency lines to any work items.

Note:

You can change the text of the label that appears above the proxy event in the workspace if the label text is too long.

Specifying multiple dependencies

A job (or job array or subflow) can have dependencies on other jobs, job arrays, subflows, files or dates and times. You can define these dependencies so that all of them must be met before the job can run, or you can define these dependencies so that only one of them needs to be met before the job can run.

Specify that all dependencies must be met

1. Change to AND link mode by clicking the Insert LinkEvent - 'AND' button.
2. Click in the workspace where you want to insert the AND link.
3. Change to job dependency mode by clicking the Insert Dependency button.
4. Draw a dependency line from the AND link to the work item it triggers.
5. Draw a dependency line from each work item that must precede the AND link to the AND link.

Note:

If you draw a second dependency line to any work item in the flow, an AND link is automatically created for you. Also, you can change an OR link into an AND link by double-clicking on the OR icon.

Specify that at least one dependency must be met

1. Change to OR link mode by clicking the Insert LinkEvent - 'OR' button.
2. Click in the workspace where you want to insert the OR link.
3. Change to job dependency mode by clicking the Insert Dependency button.
4. Draw a dependency line from the OR link to the work item it triggers.
5. Draw a dependency line from each work item that must precede the OR link to the OR link.
6. Consider specifying an exit condition for the flow: when you specify an OR link, it is possible for some predecessor jobs to the OR link to still be running when the remainder of the flow is finished.

Tip:

You can change an AND link into an OR link by double-clicking on the AND icon.

Details of a job

When you double-click on a job icon, the Edit Job dialog appears. You use this dialog to specify any information required to define the job itself, such as the command it runs, and to specify any requirements the job has, such as resources it needs to run.

The General tab

The screenshot shows the 'Job Definition - Edit job' dialog box with the 'General' tab selected. The dialog has a title bar with a close button (X) and a menu bar with tabs: General, Submit, Processing, Resources, Limits, File Transfer, Advanced, Exception Handling, and Description. The main area is divided into several sections:

- Define the Job:** Contains fields for Name (j2), Command to run, Login shell to use (dropdown), Part of project, Working directory, Environment Variables (with a Modify... button), and Non-zero success exit codes.
- Files:** Contains fields for Input file, Error file, and Output file.
- Email Notification:** Contains a checkbox for 'Notify when job', a dropdown menu set to 'starts', and an 'Email address' field set to 'root'.
- Run As:** Contains a 'User name' field set to 'root'.

At the bottom of the dialog are three buttons: OK, Cancel, and Reset.

Name the job

Every job in a flow definition requires a unique name—it cannot be the same name as any other work item within the flow. The Flow Editor assigns a unique name to each job when you draw it on the workspace, so you are not required to change the name. However, if you want to change the name, you can.

In the Name field, specify a unique name using alphanumeric characters, periods (.), underscores (_) or dashes (-). You cannot use a colon (:), semicolon (;) or pound sign (#) in a job name.

Specify the command the job runs

The purpose of a job is to run a command, so a command name is mandatory. In the Command to run field, specify the name of the command this job runs, and any arguments required by the command, ensuring the syntax of the command is correct, or specify the script to run. Because the job will run under

your user ID, ensure the path to the script is specified in your path environment variable, or specify the full path to the script.

If running this command or script requires access to a file or application, ensure the files are in a shared location that is accessible to the Platform Process Manager Server. If applicable, specify any files that need to be transferred to where the job will run on the File Transfer tab.

If running this command requires access to specific resources, ensure you specify the appropriate resource requirements on the Resources tab.

Specify a login shell to initialize the execution environment

If the execution environment needs to be initialized using a specific login shell, in the Login shell to use field, select the login shell to be used from the list provided. This is not necessarily the shell under which the job runs.

The value specified for the login shell must be the absolute path to the login shell.

Run the job as part of a project

If you are using project codes to collect accounting information, and you want to associate this job with a project, in the Part of project field, select or specify the name of the project.

Specify a job working directory

Specify the directory where the job should run. To specify it, use an absolute path rather than a relative path. When no working directory is specified, the default working directory is used (this is the user's home directory, for example, C: \Documents and Settings*user_name* on Windows or /home/*user_name*/ on Unix). When working directory does not exist, then the working directory used is % LSF_TOP%\tmp.

Submit the job with environment variables

You can submit a job that has environment variables that are used when the job runs. Environment variables can only contain alphanumeric characters, underscores, and user variable definitions. No semicolons can be part of the name or value.

User variables can be used in the environment variable name or value. A user variable definition must be in the form `#{ user_variable_name }` and must be defined.

To add an environment variable, click New and then fill in the environment variable name and value, and click OK.

To modify an environment variable, select the one you want to modify and click Edit.

To remove an environment variable, select the one you want to remove and click Remove.

Specify input, output and error files

You can use standard input, output and error files when submitting a job.

To get the standard input for the job from a file, in the Input file field, specify an absolute path to the file or a path relative to the current working directory.

To append the standard output of the job to a file, in the Output file field, specify a path to the file. If the current working directory is not accessible to the execution host, the standard output file is written to /tmp/.

To append the standard error output of the job to a file, in the Error file field, specify a path to the file. If the current working directory is not accessible to the execution host, the standard error output file is written to /tmp/.

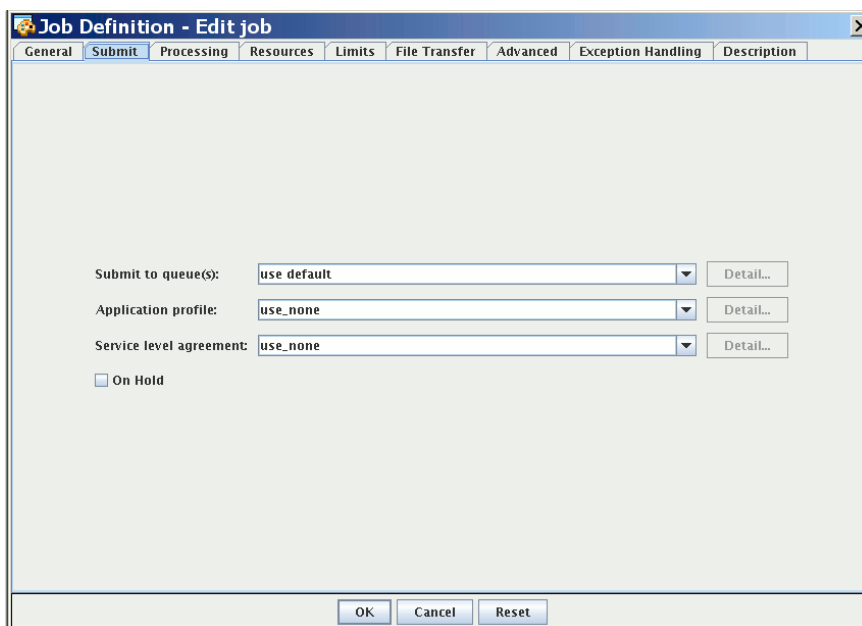
Notify a user when the job ...

You can instruct the system to send an email to you or another user when the job ends, when it starts, or once when it starts, and again when it ends. By default, you will not receive an email, except when the flow completes. To specify an email notification, check the notification box, and in the Notify when job field, select when you want the email sent. Then in the Email address field, specify the email address you want to notify.

Run the job under another user name

If you have administrator authority, you can specify a different user name under which to run the job. In the User name field, specify the user ID under which to run the job.

The Submit tab



Submit the job to a queue

Job queues represent different job scheduling and control policies. All jobs submitted to the same queue share the same scheduling and control policy. Platform Process Manager administrators configure queues to control resource access by different users and application types. You can select the queues that best fit the job.

If you want to submit your job to a particular queue, in the Submit to queue(s) field, select or specify the queue name. If you want to specify a list of queues, specify the queue names separated by a space.

When you specify a list of queues, the most appropriate queue in the list is selected, based on any limitations or resource requirements you specify, and the job submitted to that queue.

This field is optional. If you do not specify a queue name, the configured default queue is used.

Application Profile

Use application profiles to map common execution requirements to application-specific job containers. For example, you can define different job types according to the properties of the applications that you use; your FLUENT jobs can have different execution requirements from your CATIA jobs, but they can all be submitted to the same queue.

In the Application Profile drop-down list, select an Application Profile name. The drop-down lists all the Application Profile names that are configured in LSF.

Service Level Agreement

Goal-oriented Service Level Agreement (SLA) scheduling policies help you configure your workload so that your jobs are completed on time and reduce the risk of missed deadlines. They enable you to focus on the "what and when" of your projects, not the low-level details of "how" resources need to be allocated to satisfy various workloads.

In the Service level agreement drop-down list, select a goal. The drop-down lists all the SLAs that are configured in LSF.

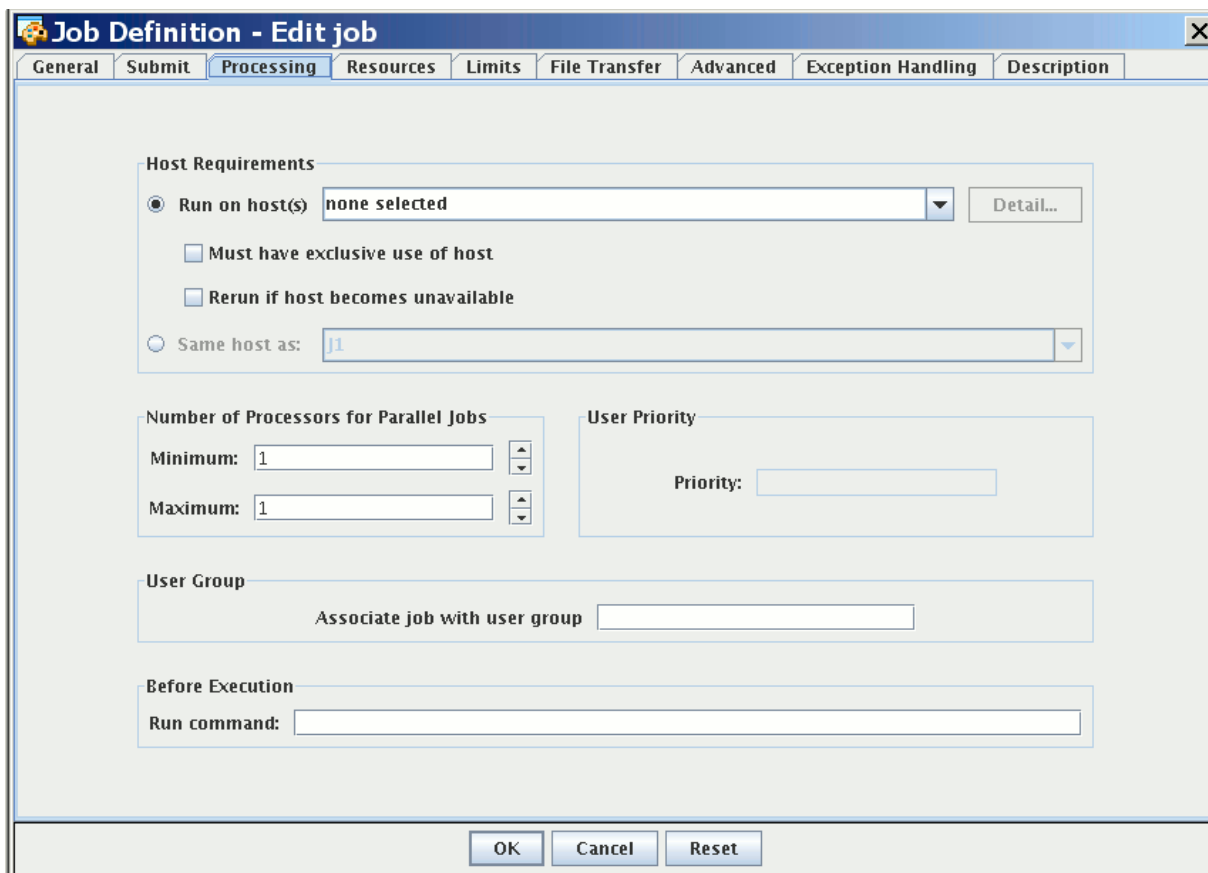
Submit the job on hold

If you are creating a flow definition that contains a job whose definition you want to include in the flow, but you do not yet want the job to run for multiple iterations of this flow, you can submit the job on hold. At a later time, you can edit the flow definition, deselect this option, and resubmit the flow. At that time, the job will run as part of the flow.

In the Flow Manager, when you look at a job that has been submitted on hold, the job is grayed out.

To submit a job on hold, check On hold.

The Processing tab



Run on a specific host

When you define a job, you can specify a host or series of hosts on which the job is to be run. If you specify a single host name, you force your job to wait until that host is available before it can run. Click Run on host(s), and select or specify the hosts on which to run this job.

Run with exclusive use of host

When you define a job, you can specify that the job must have exclusive use of the host while running—no other LSF jobs can run on that host at the same time. Check Must have exclusive use of host. The job is dispatched to a host that has no other jobs running, and no other jobs are dispatched to that host until this job is finished.

Rerun on another host if this one becomes unavailable

When you define a job, you can specify that if the host the job is running on becomes unavailable, the job should be dispatched to another host. Under the default behavior, the job exits, unless your Platform Process Manager administrator specified automatic rerun at the queue level. Check Rerun if host becomes unavailable.

Run on the same host as another job

When you define a job, you can specify to run it on the same host that another job runs on. This is useful when a job generates a large amount of data—you do not need to transfer the data to run the next job. Click **Same host as:** and select the job on whose host this job should run. The other job must have at least started to run when this job is submitted, so the Platform Process Manager can determine the correct host.

Specify number of processors for parallel jobs

If you are running a parallel job, you can specify a minimum and maximum number of processors that can be used to run the job. The maximum number is optional—if you specify only a minimum number, that is the number of processors used.

In the **Minimum** field, specify the minimum number of processors required to run the job. When this number of processors is available, and all of its other dependencies are met, the job can be dispatched.

Assign the job a priority

You can assign your jobs a priority, which allows you to order your jobs in a queue.

In the **Priority** field, specify a number from 1 to the maximum user priority value allowed at your site. See your Platform Process Manager administrator for this value.

Run a command before running the job

You can run a command on the execution host prior to running the job. Typically, you use this to set up the execution environment.

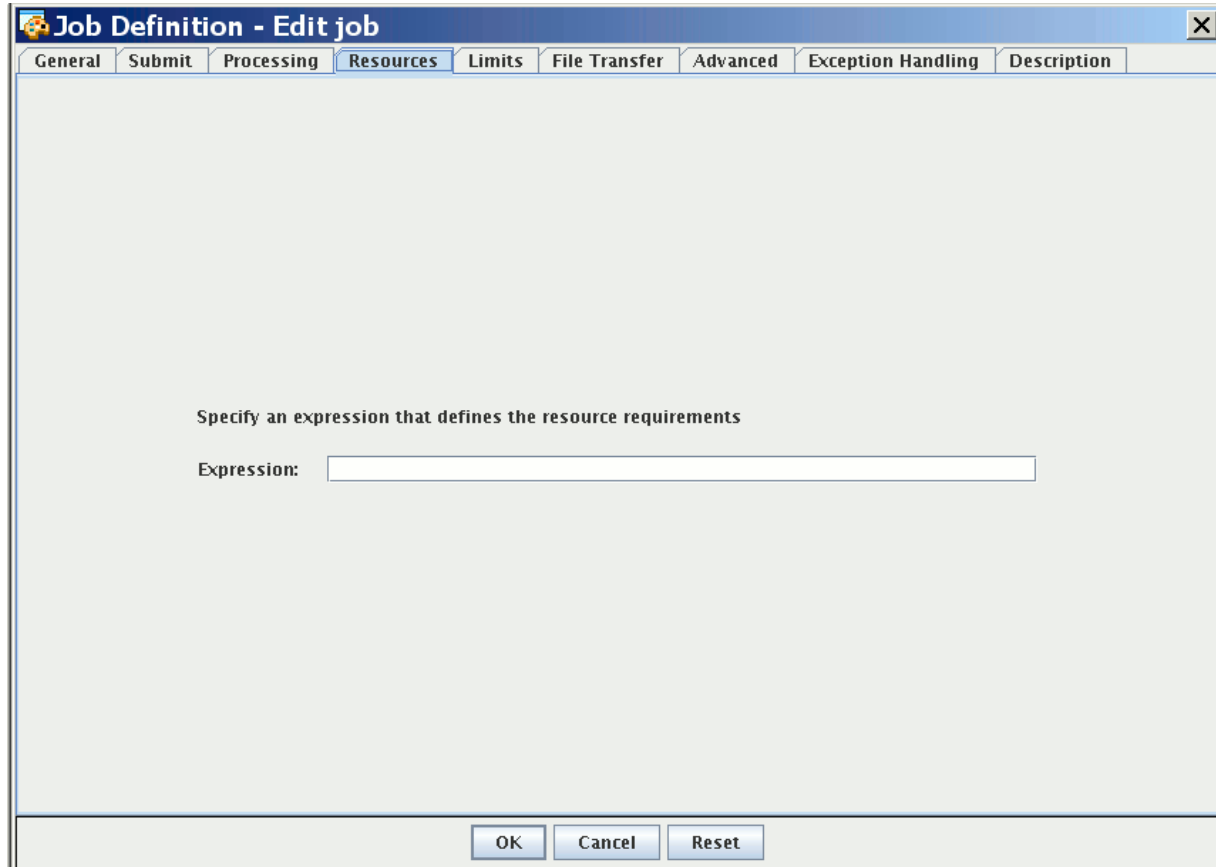
In the **Run command** field, specify the command to be run on the execution host before running the actual job. If the command runs successfully, the job can run. Otherwise the command and job are rescheduled. Be sure the command is capable of being run multiple times on the target host.

Assign the job to a fairshare group

You can assign the job to a fairshare group. You must be a member of the group you specify.

In the **Associate job with user group** field, specify the name of the group.

The Resources tab



Specify resources required to run the job

You can specify a string that defines the resource requirements for a job. There are many types of resources you can specify. For complete information on specifying resource requirements, see the *Administering Platform LSF*. However, some typical resource requirements are illustrated here. Some of the more common resource requirements are:

- I want to run the job on a particular type of host
- The job requires a specific number of software licenses
- The job requires a certain amount of swap space and memory available

You can use user variables when specifying resource requirements.

Run on host type

The following example specifies that the job must be run on a Solaris 7, 32-bit host:

```
select[type==sol732]
```

Float software licenses

The following example specifies that the job requires 3 Verilog licenses. The `rusage` statement reserves the licenses for 10 minutes, which gives the job time to check out the licenses:

```
select[verilog==3] rusage[verilog=3:duration=10]
```


In the above example, verilog must first be defined as a shared resource in LSF.

Swap space and memory

The following example specifies that the job requires at least 50 MB of swap space and more than 500 MB of memory to run:

```
select[swp>=50 && mem>500]
```

The Limits tab

Job Definition - Edit job

General Submit Processing Resources **Limits** File Transfer Advanced Exception Handling Description

Host Limits

Host name or model:

Maximum CPU time: hours and minutes

Maximum run time: hours and minutes

Job Limits

Maximum file size: (Kbytes per process)

Maximum core file size: (Kbytes)

Maximum memory size: (Kbytes per process)

Maximum data size: (Kbytes per process)

Maximum stack size: (Kbytes per process)

OK Cancel Reset

Specify host limits

You can specify criteria that ensure that the job is run on a particular host, or specific model of host. You can also limit the normalized CPU hours and minutes a job can use, or the number of hours and minutes a job can run. If the job exceeds these limits, it is killed automatically.

You can specify a host name or model in the Host name or model field.

To limit the job's usage of CPU time, in the Maximum CPU time fields, specify the number of hours and minutes the job can use before it should be killed.

To limit the job's run time, in the Maximum run time fields, specify the number of hours and minutes the job can run before it should be killed.

Specify job limitations

You can specify job limits, that restrict the following:

- The file size per job process, in kilobytes
- The core file size, in kilobytes
- The memory size per job process, in kilobytes
- The data size per job process, in kilobytes
- The stack size per job process, in kilobytes

The File Transfer tab

The screenshot shows a dialog box titled "Job Definition - Edit job" with a close button (X) in the top right corner. The dialog has several tabs: "General", "Submit", "Processing", "Resources", "Limits", "File Transfer" (which is selected), "Advanced", "Exception Handling", and "Description". The "File Transfer" tab contains three main sections:

- File Location:** Two text input fields. The first is labeled "Local path including name:" and the second is labeled "File on execution host:".
- Operation:** Three radio button options:
 - Copy file to remote host before running job
 - Copy file to local location after running job
 - Append file to local location after running job
- Expression(s):** A large text area for entering file transfer expressions. To its right are three buttons: "Add", "Remove", and "Replace".

At the bottom of the dialog are three buttons: "OK", "Cancel", and "Reset".

You use this tab to transfer required files to the host where the job runs, and to transfer output files after the job has completed. You can transfer multiple files, and perform any or all of the operations available on this tab. Simply create a list of each required file transfer in the Expression(s) field.

Transfer a local file

If the job you are defining requires one or more applications or data files to run, and those files do not exist on the host on which the job runs, you need to transfer the files to the host when the job is dispatched.

1. In the Local path including name field, specify the full path name of the file to be transferred.
2. If the location on the host where the job will run is different from the local path, in the File on execution host field, specify the full path where the file should be located when the job runs.

3. Select Copy file to remote host before running job.
4. Click Add to add this operation to the list of operations to perform.
5. Repeat as required.

Transfer an output file locally after the job runs

If the job you are defining produces output files that must be transferred to another location after the job completes, you need to copy the output files locally after the job runs.

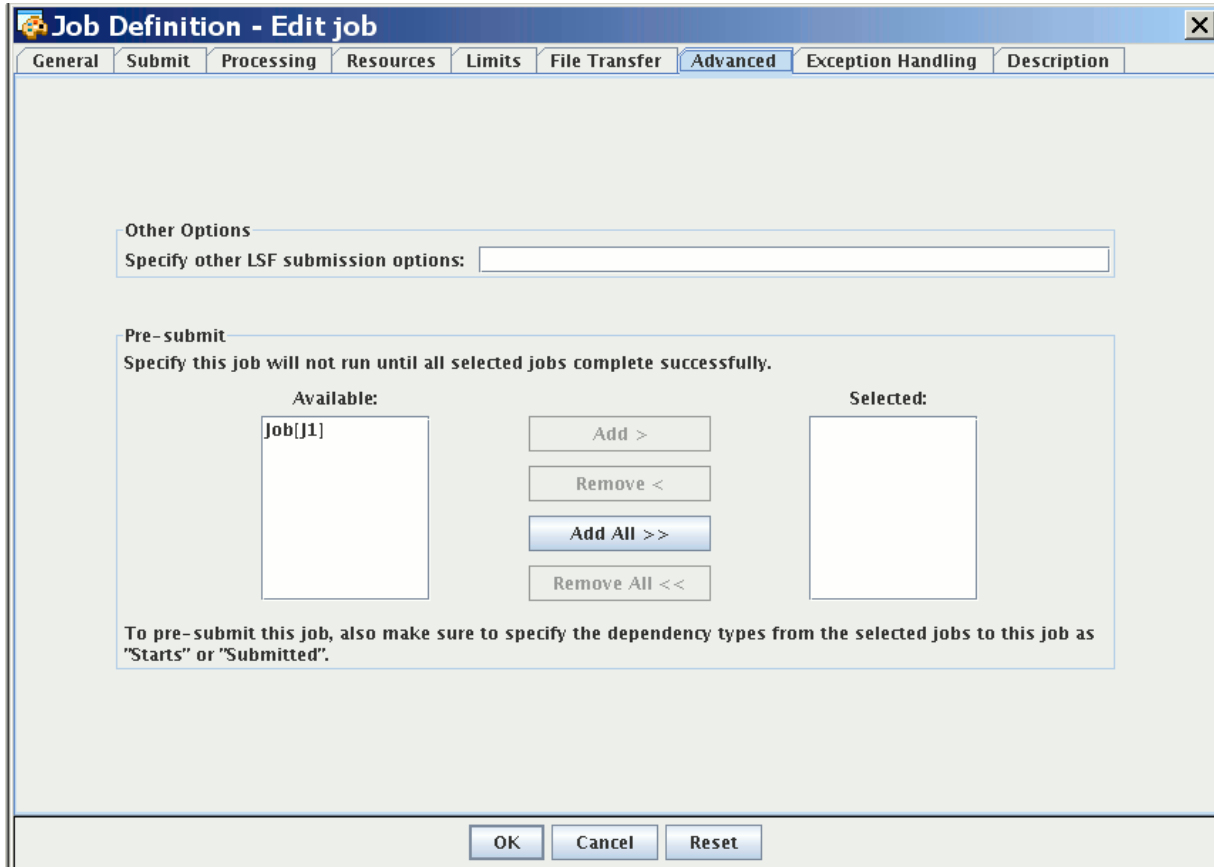
1. In the Local path including name field, specify the full path name where the output file is to be stored locally.
2. In the File on execution host field, specify the full path where the output file will be located when the job completes.
3. Select Copy file to local location after running job.
4. Click Add to add this operation to the list of operations to perform.
5. Repeat as required.

Append output to a local file after the job runs

If the job you are defining produces output files that must be transferred to another location after the job completes, and you want the output appended to a file that already exists, do the following:

1. In the Local path including name field, specify the full path name where the output file is to be appended.
2. In the File on execution host field, specify the full path where the output file will be located when the job completes.
3. Select Append file to local location after running job.
4. Click Add to add this operation to the list of operations to perform.
5. Repeat as required.

The Advanced tab



You use this tab to specify additional bsub submission options that are not available from the Definition dialog, and to select jobs upon which this job depends.

Other Options

You can specify additional bsub submission options for a job.

This allows you to use options that are not available from the job definition dialog. The options you specify are added to the bsub command when you submit the job or job array.

You can also specify user variables in the Other Options field.

Note:

The following options are not supported in the Other Options field: -l, -lp, -ls for interactive jobs, and -K for submitting a job and waiting for it to complete.

- Example: Specify License Scheduler options:

For example, if you use the esub feature and you want to display accounting statistics for jobs belonging to specific License Scheduler projects, specify in the Other Options field:

```
-a myesubapp -Lp mylsproject
```

- Example: Specify complex job dependencies

If you have complex dependencies between jobs, you can use the Other Options field with the built-in user variable `JS_FLOW_FULL_NAME`.

Note that you are limited to what the LSF `bsub -w` option supports. For more details, refer to the *LSF Command Reference*.

- To specify jobB depends on the completion of jobA, regardless of its exit code:

In jobB's job definition dialog, Advanced tab, Other Options field, specify:

```
-w "ended(#{JS_FLOW_FULL_NAME}:jobA)"
```

In this example, `JS_FLOW_FULL_NAME` is the full name of the subflow containing jobA and jobB.

- To specify jobB depends on a combination of dependencies, specify in Other Options:

```
-w "ended(#{JS_FLOW_FULL_NAME}:JobA) && done(#{JS_FLOW_FULL_NAME}:JobC)"
```

- To specify dependencies for jobs in flow array elements:

A job in a flow array element has a name in the format "11:usr1:F1:FA(1):J1".

Make sure you single quote the name. For example:

```
-w "exit('11:usr1:F1:FA(1):J1', > 2)"
```

Using the `JS_FLOW_FULL_NAME` variable:

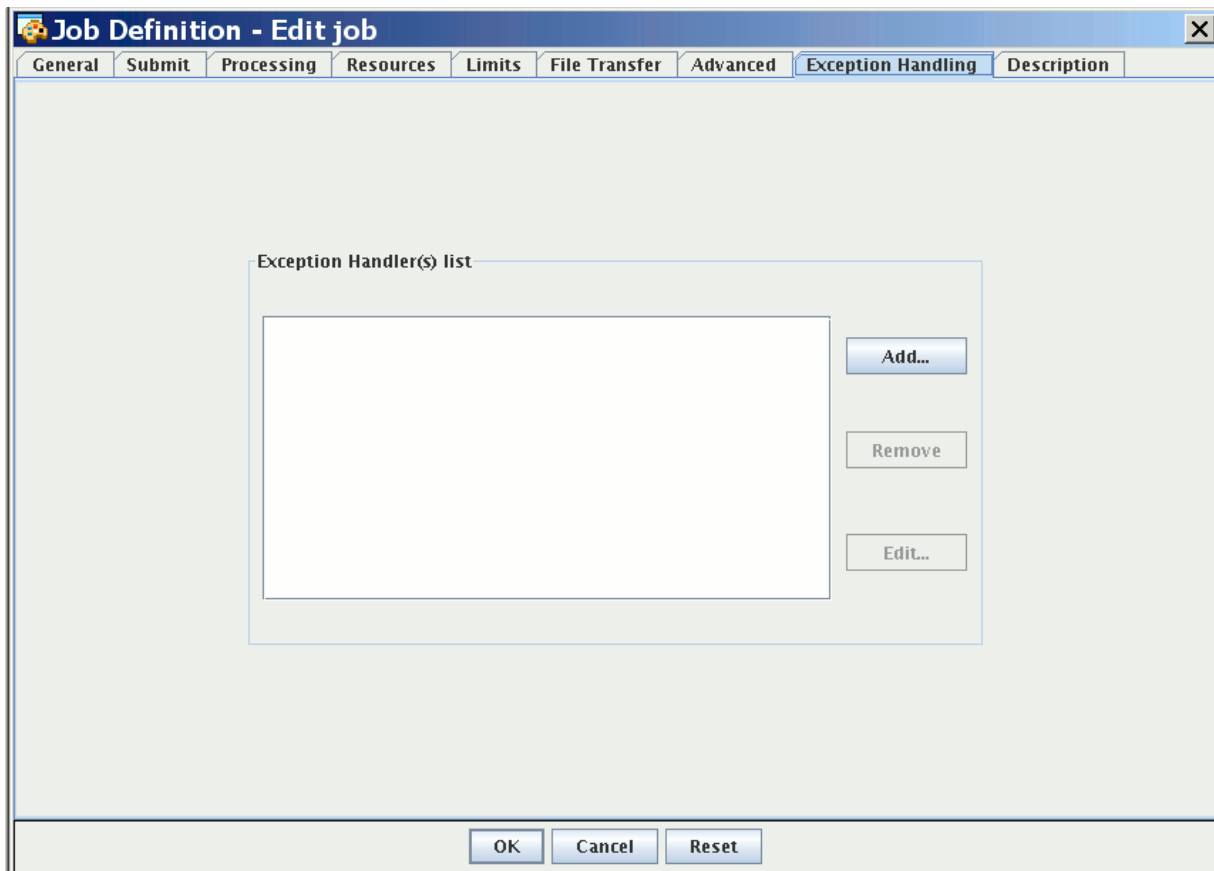
```
-w "exit(#{JS_FLOW_FULL_NAME}#{JS_FLOW_INDEX}:J1', > 2)"
```

Pre-submit

Select jobs upon the current job depends.

- Only LSF jobs, job arrays, job scripts, job array scripts, and template jobs can be pre-submitted.
- Only jobs, job scripts, job arrays, job array scripts, and template jobs can be preceding jobs to the dependent job to be pre-submitted.
- The jobs to be pre-submitted must be direct links. They cannot be more than one link away.
- The dependencies of all predecessors must be logically connected with AND.
- In Flow Editor, the Event type in the Job Event Definition for the preceding jobs to the other job must be set to Starts or Is Submitted.
- If you specify dependent jobs to be pre-submitted, and the condition is never met, it is possible for the flow to be "stuck". To handle this, define an overrun exception handler to kill the last job if it runs or pends for more than a certain period of time.

The Exception Handling tab



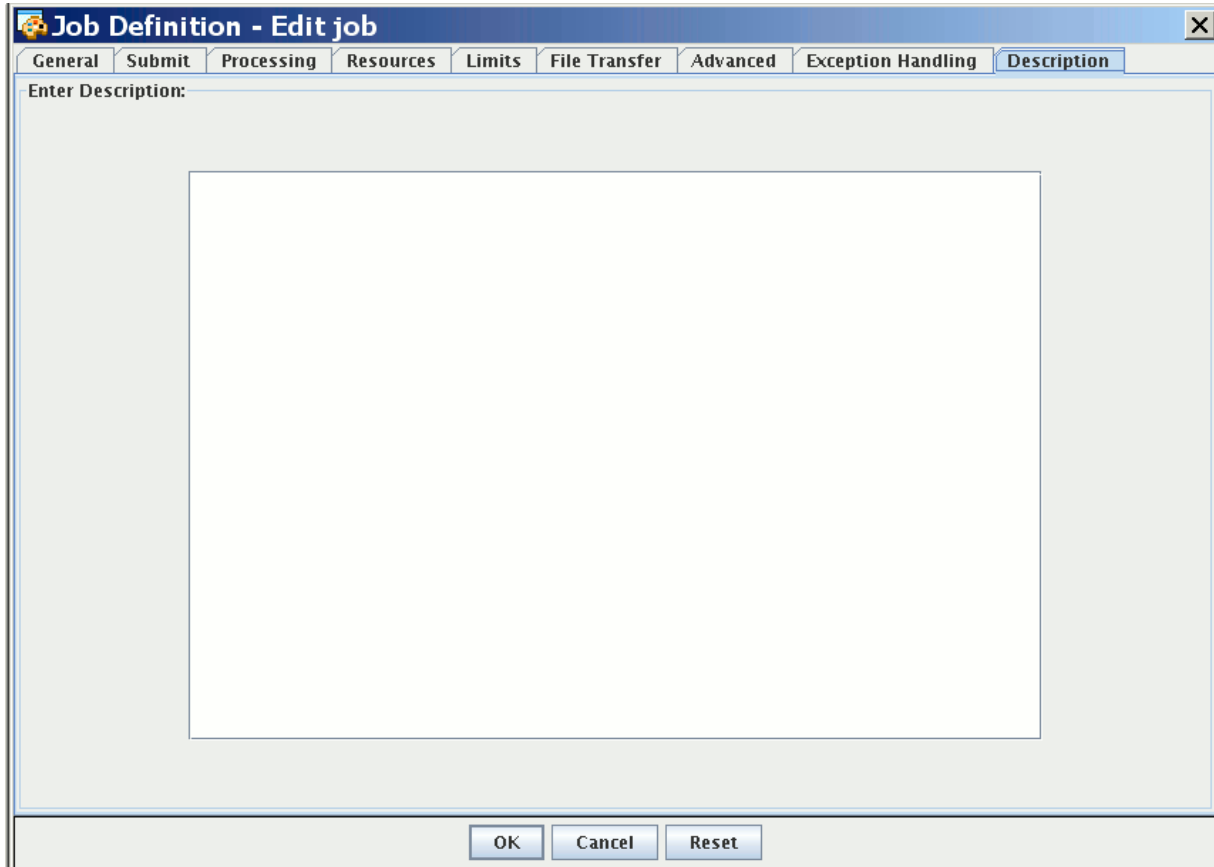
You use this tab to specify what action to take if a specific exception occurs while running this job or job array.

Exceptions and applicable handlers

In a...	If this exception occurs...	You can use this handler...
Job	Overrun	Kill
	Underrun	Rerun
	Specified exit code	Rerun
Job array	Overrun	Kill
	Underrun	Rerun
	Sum of exit codes	Rerun
	Number of unsuccessful jobs	Kill

The Description tab

In the input field, add any descriptive text that may be used for managing this job or job array within the flow. For example, if this job requires special instructions for operations staff, place those instructions here.



The image shows a screenshot of a software dialog box titled "Job Definition - Edit job". The dialog has a blue title bar with a close button (X) in the top right corner. Below the title bar is a tabbed interface with the following tabs: "General", "Submit", "Processing", "Resources", "Limits", "File Transfer", "Advanced", "Exception Handling", and "Description". The "Description" tab is currently selected and highlighted. The main area of the dialog is labeled "Enter Description:" and contains a large, empty white rectangular text input field. At the bottom of the dialog, there are three buttons: "OK", "Cancel", and "Reset".

About flow completion attributes

Because flows can be as individual as their creators, and may contain recovery jobs that run when another job fails, Platform Process Manager provides many options to choose from when defining your flow.

For example, you may require that every job in a flow complete successfully, and if any job fails, you may want to stop processing the flow immediately. In another case, you may want to process as many jobs as possible in a flow, and handle any exceptions on an individual basis. The first example is handled by the default behavior of Platform Process Manager, the latter by defining flow completion attributes.

You define flow completion attributes to a flow to describe the criteria the Platform Process Manager Server should use to determine when to assign a state to the flow—when it should be considered complete. You can also specify what the Platform Process Manager Server should do with any jobs that are running when it determines a flow is complete.

Default completion criteria of a flow

By default, Platform Process Manager considers a flow to be complete (Done or Exited) when:

- All work items in the flow have completed successfully. The flow is Done.
- or
- Any work item in the flow fails or is killed. The flow is Exited.

Alternative completion criteria

You can specify two alternatives to the default completion criteria for a flow:

1. Specify a list of work items that must end before the flow is considered to be complete, and ignore the other work items in the flow when determining the state of the flow
2. Specify a list of work items, any one of which must end before the flow is considered to be complete, and ignore the other work items in the flow when determining the state of the flow

Default completion behavior of a flow

By default, when a flow is considered complete and has been assigned a state, no new work is dispatched, unless it is within a subflow or job array that is still in progress. Any work that is currently processing completes, and the flow is stopped.

If, however, you have selected a list of work items, and specified that all must end before the flow is considered complete, even if a work item in the flow exits, the flow continues processing until all of the selected items have completed. At that time, any work that is currently processing completes, and the flow is stopped.

Conversely, if you have selected a list of work items, and specified that the flow is complete when any of the selected work items ends, the flow continues processing until one of the selected items ends, even if other work items exit. At that time, any work that is currently processing completes, and the flow is stopped.

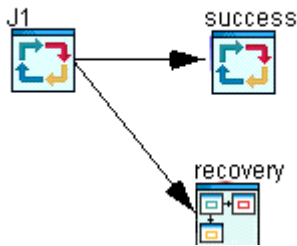
Alternative completion behavior

You can direct Platform Process Manager to continue processing work in a flow even after it is considered complete and has been assigned a state. In this case, Platform Process Manager continues to process the flow until it cannot run any more work, or until the remaining work is dependent on events or has dependencies that cannot be met, and then the flow is stopped.

If you use error recovery routines

You may choose to include error recovery routines within a flow that only run when a particular work item in the flow fails. Not only will you not want the flow to wait indefinitely for work that can never complete, you will also not want the flow to stop, preventing the error recovery routine from running.

In this case, you can select particular work items that must end before the flow should be considered complete. You can specify that all of the selected work items must end, or to consider the flow complete when any one or more of the selected work items end. In the case of the following flow with an error recovery routine, you want the flow to be considered complete when either success or recovery complete:



If you use multiple branches in a flow

You may define a flow that contains multiple branches. In this flow, if one branch fails, you may not want the flow to stop processing. Perhaps you want to let the flow to run as much as it can, and then you will perform some manual recovery and rerun the failed branch.

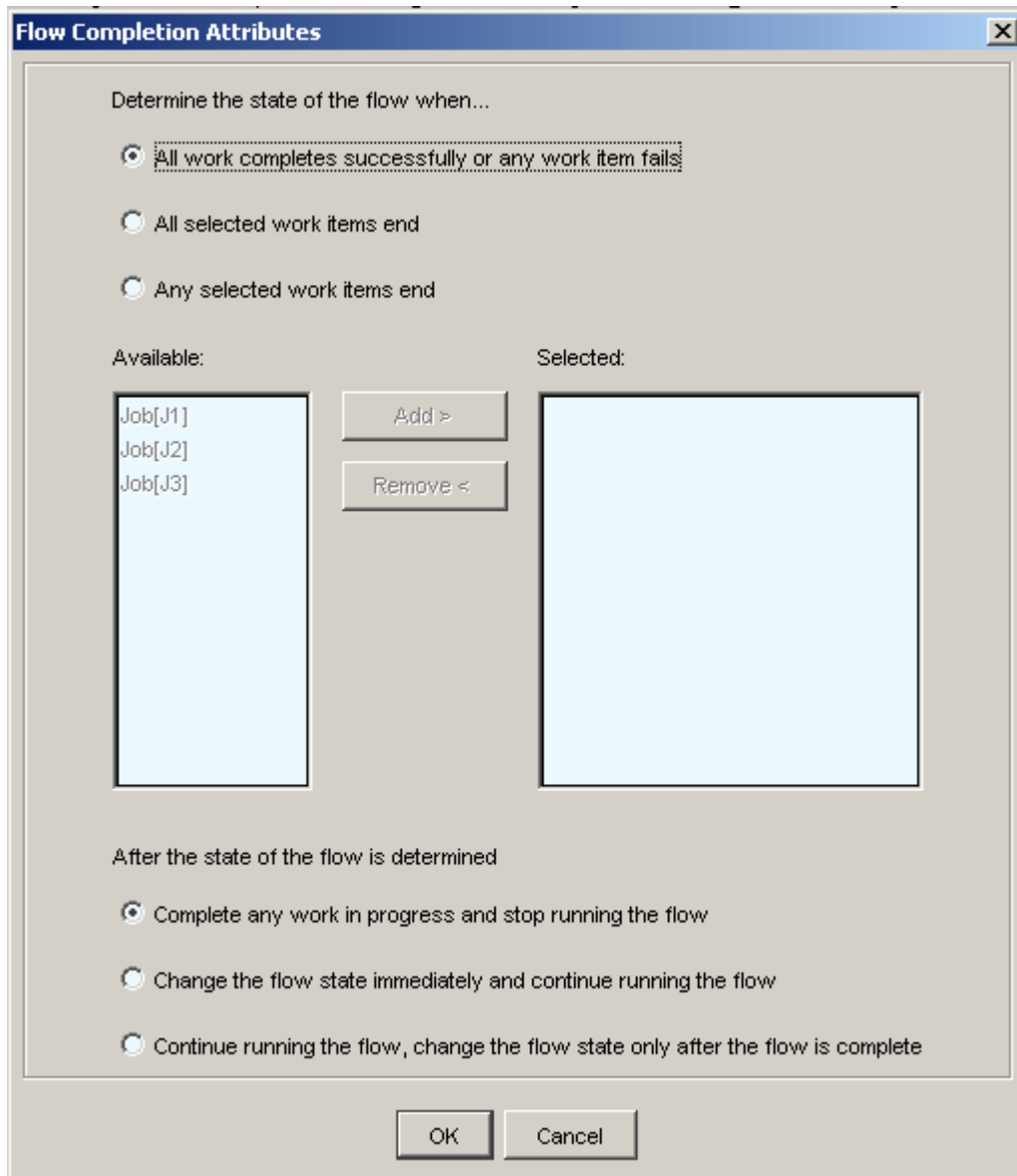
Specify flow completion attributes

You can specify the following flow completion criteria to specify when Platform Process Manager should consider the flow complete and assign it a state:

- All work completes successfully or any work item fails. This is the default.
- All selected work items end.
- Any selected work items end. You can also specify what Platform Process Manager should do when the state of the flow is determined
- Complete any work in progress and stop running the flow. This is the default.
- Change the flow state immediately but continue running the flow until any remaining work items that can complete, complete.
- Continue running the flow and only change the state when any remaining work items that can complete, complete.

Assign a state to a flow when all work items are done

1. From the Action menu, select Specify Flow Completion Attributes, or right-click in a blank section of the flow definition, and select Completion Attributes. The Flow Completion Attributes dialog box appears.



2. Leave the first option set to the default All work completes successfully or any work item fails.
3. If you want the flow to stop running if any work item exits, leave the option as the default Complete any work in progress and stop running the flow. If you want the work item to continue to run but change the flow state, select Change the flow state immediately and continue running the flow. This allows you to immediately trigger the next work item if there is one that is dependent on this flow or subflow. If you want to continue to process as many jobs in the flow as possible Continue running the flow, change the flow state only after the flow is complete.
4. Click OK. The flow will be assigned a state when all of the work items complete successfully or any work item fails or is killed.

Assign a state to a flow when all selected work items end

1. From the Action menu, select Specify Flow Completion Attributes, or right-click in a blank section of the flow definition, and select Completion Attributes. The Flow Completion Attributes dialog box appears.
2. Select All selected work items end.
3. From the list of available work items, select those that must process before the flow can be assigned a state. Select each item and click Add> to move it to the list of selected items, or double-click on an item to move it to the other list.
4. If you want the flow to stop running when the specified jobs end, leave the option as the default Complete any work in progress and stop running the flow. If you want the work item to continue to run but change the flow state, select Change the flow state immediately and continue running the flow. This allows you to immediately trigger the next work item if there is one that is dependent on this flow or subflow. If you want to continue to process as many jobs in the flow as possible Continue running the flow, change the flow state only after the flow is complete.
5. Click OK. The flow will be assigned a state when all of the selected work items end.

Assign a state to a flow when any selected work item ends

1. From the Action menu, select Specify Flow Completion Attributes, or right-click in a blank section of the flow definition, and select Completion Attributes. The Flow Completion Attributes dialog box appears.
2. Select Any selected work items end.
3. From the list of available work items, select those that may process before the flow can be assigned a state. Select each item and click Add> to move it to the list of selected items, or double-click on an item to move it to the other list. When one item in this list ends, the flow will be assigned a state.
4. If you want the flow to stop running when the specified jobs end, leave the option as the default Complete any work in progress and stop running the flow. If you want the work item to continue to run but change the flow state, select Change the flow state immediately and continue running the flow. This allows you to immediately trigger the next work item if there is one that is dependent on this flow or subflow. If you want to continue to process as many jobs in the flow as possible Continue running the flow, change the flow state only after the flow is complete.
5. Click OK. The flow will be considered complete when one of the selected work items ends.

Continue processing when the state of the flow is determined

1. From the Action menu, select Specify Flow Completion Attributes, or right-click in a blank section of the flow definition, and select Completion Attributes. The Flow Completion Attributes dialog box appears.
2. If you want the work item to continue to run but change the flow state, select Change the flow state immediately and continue running the flow. This allows you to immediately trigger the next work item if there is one that is dependent on this flow or subflow.

3. Click OK. When the flow is considered complete and assigned a state, any eligible work items in the flow will continue to process until Platform Process Manager cannot run any more work, or until the remaining work is dependent on events or has dependencies that cannot be met. The flow is then stopped.

Continue processing and only change the state after the flow is complete

1. From the Action menu, select Specify Flow Completion Attributes, or right-click in a blank section of the flow definition, and select Completion Attributes. The Flow Completion Attributes dialog box appears.
2. If you want to continue to process as many jobs in the flow as possible Continue running the flow, change the flow state only after the flow is complete.
3. Click OK. The flow will continue to run and only change state once Platform Process Manager cannot run any more work, or until the remaining work is dependent on events or has dependencies that cannot be met.

Configuring flow exit codes

By default, a Done job has an exit code of 0. As a result, a Done flow or subflow has an exit code of 0, since the default way that Process Manager determines the flow exit code is through the sum of all exit codes of all work items in the flow.

However, it is possible to specify custom success exit codes for LSF jobs, job scripts, local jobs, and manual jobs. As a result, if you specify custom success exit codes for these types of jobs, a Done flow can have an exit code other than 0.

If there is more than one Done job with an exit code other than 0 in a Done flow, or there are some jobs with Done or Exited states with codes other than 0 in a failed flow, the sum of all exit codes may not be meaningful to you.

For such cases, you can configure the flow to inherit the exit code of the last item that was successfully completed or that failed. You can do this in the Flow completion Attributes dialog, with the option Determine the flow exit code from the last finished work item in a successful flow, or the last failed work item in a failed flow.

How the system selects the last finished or failed work item:

- If more than one work item finishes or fails last and at the exact same time, the system picks an item at random to get the exit code.
- If you select Change the flow state immediately and continue running the flow the system does not consider jobs that finish or fail after the flow state was changed.

In combination with the other options in the Flow Attributes dialog, you can configure your flow to have an exit code that makes sense to you.

Configure flow exit code calculation

1. In Flow Editor, select Action > Specify Flow Completion Attributes.

The Flow Completion Attributes dialog is displayed.

2. Select all desired behavior for flow completion:

- Determine the state of the flow when...
- After the state of the flow is determined
- Determine the flow exit code from
 - Select The last finished work item in a successful flow, or the last failed work item in a failed flow.

3. Click OK.

Configure dependencies for subflows

If your flow has subflows, when creating your flow, you want to establish a dependency between the subflow and other work items to track when flow completes successfully with a specific exit code, or when a flow fails with a specific exit code.

1. In Flow Editor, draw a dependency from the subflow to the next work item.
2. Select the dependency, right-click and select Open Definition.

The Flow Definition is displayed.

3. Select the event type **The flow completes successfully with exit code...**, **The flow fails with exit code....**, or **The flow fails**.
4. Click **OK**.

Specify exception handling for a flow

You can use Platform Process Manager to monitor for specific exception conditions when a flow is run, and specify handlers to run automatically if those exceptions occur.

You can monitor a flow for the following exceptions:

- Overrun—the flow runs longer than it should
 - Underrun—the flow runs for an abnormally short time
 - The flow has exit code—the flow ends with a particular exit code
 - Number of unsuccessful jobs—a particular number of jobs in the flow are unsuccessful
1. From the Action menu, select Add Flow Attribute, or right-click in a blank section of the flow definition and select Flow Attribute, and select the Exception Handlers tab. The Flow Attribute dialog box appears.
 2. On the Exception Handling tab, click Add.
 3. In the Exception type field, select the exception you want to handle.
 4. If you chose *Runs more than...*, in the Expected run time field, specify the maximum time, in minutes, the flow can run before it should be killed.

If you chose *Runs less than...*, in the Expected run time field, specify the minimum time, in minutes, the flow can run before it should be rerun.

If you chose the *flow has exit code*, in the Value field, choose the operator and value that best define the exit code requirement. For example, greater than 5.

If you chose *number of unsuccessful jobs*, in the Value field, choose the operator and value that best define the requirement. For example, greater than 3.

5. In the Action field, select the appropriate exception handler. In most cases, however, the appropriate exception handler is selected for you, as follows:

If you monitor for this exception...	This handler is used...
Overrun	Kill
Underrun	Rerun
Exit code	Rerun
Number of unsuccessful jobs	Kill

6. Click OK. The exception handling specification is added to the list.
7. Repeat steps 2 through 6 until you have finished specifying exceptions to handle. Click OK.

Flow attributes

You can specify the following flow attributes:

- A description of the flow
- Email notification about the flow
- Preventing concurrent versions of the same flow
- Automatic exception handling of the flow

Flow description

You can use the description field of a flow to include any instructions regarding the flow, or to include general descriptions about what this flow does. This is especially useful if your site uses shared flows, that might be reused by another user.

Flow working directory

You can specify the working directory for the flow. 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.

You can use user variables when specifying the working directory.

Email notification for a flow

By default, Platform Process Manager notifies you by email only if your flow exits. You can set the notification options to send an email to you or another user when:

- The flow exits
- The flow ends, regardless of its success
- The flow starts
- The flow starts and exits
- The flow starts and ends, regardless of its success

If the flow exits, the email provides information about the jobs that caused the flow to exit. If you are using the default flow completion criteria, this is information about the job or job array that exited. If you specified flow completion criteria, this includes on the jobs specified in the flow completion criteria that exited.

You can also turn off flow email notification entirely.

At the system level, your Platform Process Manager administrator can turn off flow email notification, or limit the size of the emails you receive. If you are not receiving email notifications you requested, or if your email notifications are truncated, check with your administrator.

Prevent concurrent flows

When you create a flow definition, you can prevent multiple copies of the flow from running at the same time. This is useful when you need to run a flow repeatedly, but any occurrence of the flow must have exclusive access to a database, for example.

Specify flow attributes

1. From the Action menu, select Add Flow Attribute or right-click in a blank section of the flow definition and select Flow Attribute.
The Flow Attribute dialog box appears.
2. On the General tab, enter the description text in the field provided. When you have finished typing the description, click OK.
3. To specify user variables and environment variables, click Modify, which is located to the right of the Input Variables field.

The Flow Input Variables dialog displays a list of input variables that are currently defined in the flow, and the order in which they are defined.

You can specify variables on a per-flow basis, which allows multiple jobs or sub-flows within a flow to use the same variable; you can specify variables from the job definition dialog, which overrides these flow-level input variables; and you can specify built-in variables.

Input variables can only contain alphanumeric characters, underscores, and user variable definitions. No semicolons can be part of the value.

Important:

The order in which you define the variables is important. You must define one variable before you can use it in the next. For example, if you have an input variable named `MyVar_{Var 1}`, you must have defined the `Var 1` variable first. `Var 1` can either be defined in the current flow level (if it was defined before `MyVar_{Var 1}`) or at a parent or other upper level flow.

- To add an input variable, click New.

Specify a name for the variable. Unless you are explicitly specifying a built-in variable, do not use a variable name beginning with `JS_`.

To define a default value, select the Specify a Default Value field, then specifying a default value for the variable. You can include user variables when specifying a default value.

Important:

Do not specify a default value for a built-in variable.

- To modify an input variable, select the variable you want to edit and click Edit.
- To remove an input variable, select the variable you want to remove and click Remove.

The environment variables defined in the list can only be used in the current flow or sub-flow, though any environment variables listed at the parent flow level are propagated to its subflows. Variables defined in sub-flows will overwrite variables defined in the main flow or parent flows if they have the same name.

Changing environment variables in a job cannot affect other jobs. Therefore, if one job changes the value of an environment variable, the same environment variable is unaffected in the next job, even if it runs after the first job.

The names and values of local and environment variables that a job uses appear in the Runtime Attributes of that job.

4. Optional. To specify a working directory at the flow level, use the Working directory field.

Tip:

You can use user variables when specifying the working directory.

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.

5. In the Notify when flow field, select the appropriate notification option. To receive a notification only if a flow exits, leave the default at Notify when flow exits. Otherwise, leave Notify when flow checked, and select the desired option.
6. In the Email address field, specify the email address to be notified. The default email address is your user name.

Tip:

You can use user variables when specifying the email address.

7. To prevent concurrent versions of the same flow, in the Options box, check Allow only one flow to run at a time.
8. Click OK.

Turn off email notification for a flow

1. On the General tab, uncheck Notify when flow. This does not affect email notifications regarding job completion.
2. Click OK.

Save the flow definition

You can save a flow definition at any time, whether it is complete or not. You can save the flow definition locally or on a shared-file system.

When saving the flow definition, specify a unique file name using alphanumeric characters, periods (.), underscores (_) or dashes (-). You cannot use a colon (:), semicolon (;) or pound sign (#) in a job name.

The file name you assign is concatenated with your user ID to become the flow name.

If you plan to use this flow definition as a subflow within another flow definition, ensure you give it a meaningful name that will make it unique within the other flow definition.

Once you submit a flow definition, a copy of the flow definition is stored within the Platform Process Manager system. If you make a change to the flow definition, you need to submit the flow definition again before the changes take effect in Platform Process Manager.

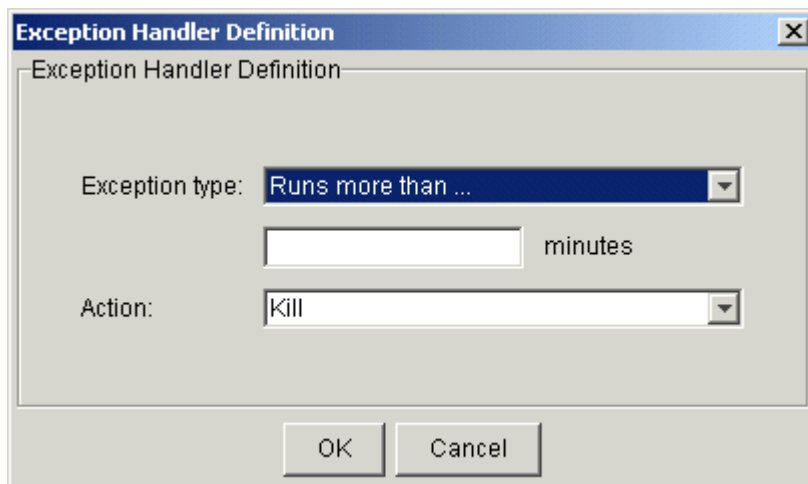
Loop a flow or subflow

You can define a flow or subflow that loops a specific number of times or loops until a specific condition is met. This is useful if you need to rerun a group of jobs until you achieve specific results.

In this example, John needs to run a series of three jobs that need to be repeated until the correct data results—an undetermined number of times. John created the following flow called `DataRefine`:



Platform Process Manager allows you to automatically rerun a flow or subflow whenever a particular work item in the flow has a specific exit code. This allows John to loop `DataRefine` as many times as required to complete refining the data results. In the script run by the job `Examine_data`, John sets the exit code of the job to be a particular value, such as 77, until such time as the data refinement is complete. Then the exit code of `Examine_data` is set to 0. John used the exception handling at the flow level to loop the flow, as follows:



If John requires it, he can use the number of times the flow is rerun in his job. This information is available through the built-in variable

```
JS_ITERATION_COUNTER
```

[*flow_name*], where *flowname* is the name of the flow, without the user name. For example:

```
JS_ITERATION_COUNTER
```

```
[myflow:subflow]
```

To loop a flow:

1. Define the jobs in the flow.
2. Ensure that one job in the flow sets a specific exit code in the circumstances under which you want to rerun the flow. Ensure that it sets a different value when the flow should stop rerunning.
3. Set the automatic rerun exception handler to rerun the flow under the correct circumstances:
 - a) Right-click in an empty space in the flow definition, and select Flow Attribute.

- b) Click Exception Handling.
- c) Click Add. The Exception Handler Definition dialog appears.
- d) In the Exception type field, select A work item has exit code...
- e) In the Work item field, ensure the correct work item is selected.
- f) In the Has exit code field, specify the exit code conditions required to loop the flow.
- g) In the Action field, ensure Rerun is specified.
- h) If applicable, in the After field, specify the number of minutes to delay the rerunning of the flow.
- i) In the Maximum number of reruns field, specify the maximum number of times you want the exception handler to rerun the flow.
- j) Click OK. The exception handling specification is added to the list. Click OK again.

The flow you define here will always loop under the specified circumstances, even when embedded in another flow as a subflow.

Loop a subflow that does not contain a loop definition

1. Add the subflow at the appropriate location in your flow.

Note: You still need to ensure that one job in the subflow sets a specific exit code in the circumstances under which you want to rerun the flow. Ensure that it sets a different value when the subflow should stop rerunning.

2. Right-click on the subflow icon, and select Attributes.
3. Click Exception Handling.
4. Click Add. The Exception Handler Definition dialog appears.
5. In the Exception type field, select A work item has exit code...
6. In the Work item field, ensure the correct work item is selected. By default, it is the last work item in the flow.
7. In the Has exit code field, specify the exit code conditions required to loop the flow.
8. In the Action field, ensure Rerun is specified.
9. If applicable, in the After field, specify the specify the number of minutes to delay the rerunning of the flow.
10. In the Maximum number of reruns field, specify the maximum number of times you want the exception handler to rerun the flow.
11. Click OK. The exception handling specification is added to the list. Click OK again. The subflow icon changes to indicate that this subflow will loop, as follows:



Note:

Any work item that depends on DataRefine cannot run until the looping completes.

About Platform Process Manager 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 exceptions

Platform Process Manager monitors for the following exceptions:

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

Misschedule

A *Misschedule* exception occurs when a job, job array, flow or subflow 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.

Note:

When a job depends on a time event, and you want to monitor for a misschedule of the job, ensure that the time event either directly precedes the job in the flow diagram, or precedes no more than one link (AND or OR) prior to the job in the flow diagram. Platform Process Manager is unable to process the misschedule exception if multiple links are used between the time event and the job depending on it.

Overrun

An *Overrun* exception occurs when a job, job array, flow or subflow exceeds its maximum allowable run time. You use this exception to detect run away or hung jobs. The time is calculated using wall-clock time,

from when the work item is first submitted to LSF until its status changes from Running to Exit or Done, or until the Overrun time is reached, whichever comes first.

Underrun

An *Underrun* exception occurs when a job, job array, flow or subflow finishes sooner than its minimum expected run time. You use this exception to detect when a job finishes prematurely. This exception is not raised when a job is killed by Platform Process Manager. The time is calculated using wall-clock time, from when the work item is first submitted to LSF until its status changes from Running to Exit or Done.

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 execution host, or a file system was not mounted properly.

Cannot Run

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

Behavior when an exception occurs

The following describes Platform Process Manager behavior when an exception occurs, and no automatic exception handling is used:

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.

When a ...	Experiences this exception ...	This happens ...
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.
	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 the when the job is successfully submitted until it reaches Exit or Done state.
Job array	Misschedule	The job array is not run.
	Cannot Run	The job array is not run.
	Start Failed	The job array is still waiting. Submission of the job array is retried until it runs.
	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 all elements in the array reach Exit or Done state.

User-specified conditions

In addition to the Platform Process Manager 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 in addition to the Platform Process Manager exceptions:

Work Item	Condition
Flow	An exit code of n (sum of all exit codes) n unsuccessful jobs
Subflow	An exit code of n n unsuccessful jobs
Job	An exit code of n

About Platform Process Manager exceptions

Work Item	Condition
Job array	An exit code of n n unsuccessful jobs

About exception handling

Platform Process Manager provides built-in exception handlers you can use to automatically take corrective action when certain exceptions occur, minimizing human intervention required. You can also define your own exception handlers for certain conditions.

Platform Process Manager built-in exception handlers

The built-in exception handlers are:

- Rerun
- Kill

Rerun

The *Rerun* exception handler reruns the entire job, job array, subflow or flow. 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.

Kill

The *Kill* exception handler kills the job, job array, subflow or flow. 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.

User-defined exception handlers

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

- Opening an alarm
- Running a recovery job
- Triggering another flow

Alarm

An *alarm* provides a visual, graphical cue that an exception has occurred, and either an email notification to one or more addresses, or the execution of 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.

When you are creating your flow definition, you can add a predefined alarm to the flow diagram, as you would another job. You create a dependency from the work item to the alarm, which can be opened by any of the exceptions available in the dependency definition. The alarm cannot precede another work item in the diagram—you cannot draw a dependency from an alarm to another work item in the flow.

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.

Valid alarm names are configured by the Platform Process Manager administrator.

Recovery job

You can use a job dependency in a flow diagram 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.

Behavior when exception handlers are used

The following describes Platform Process Manager 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'
	Underrun	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, or from any rerun starting points, as many times as required until the execution time exceeds the underrun time specified.
	An 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, or from any rerun starting points, as many times as required until an exit code other than n is reached.
	n unsuccessful jobs	Kill	The flow is killed. All incomplete jobs in the flow are killed. The flow status is 'Killed'

When a ...	Experiences this Exception ...	and the Handler Used is ...	This Happens ...
Subflow	Misschedule	Alarm	The alarm is opened. The subflow is not run. The flow continues execution as designed.
		Recovery job or flow	The subflow is not run. The flow continues execution as designed. The recovery job or flow is triggered.
	Overrun	Alarm	The alarm is opened. Both the flow and subflow continue execution as designed.
		Recovery job or flow	Both the flow and subflow continue execution as designed. The recovery job or flow is triggered.
		Kill	The subflow is killed. The flow behaves as designed.
	Underrun	Alarm	The alarm is opened. The flow continues execution as designed.
		Recovery job or flow	The subflow continues execution as designed. The recovery job or flow is triggered.
		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.
	An 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.
	n unsuccessful jobs	Kill	The subflow is killed. The flow behaves as designed.

About Platform Process Manager exceptions

When a ...	Experiences this Exception ...	and the Handler Used is ...	This Happens ...
Job or job array	Misschedule	Alarm	The alarm is opened. The job or job array is not run. The flow continues execution as designed.
		Recovery job or flow	The job or job array is not run. The flow continues execution as designed. The recovery job or flow is triggered.
	Overrun	Alarm	The alarm is opened. Both the flow and job or job array continue to execute as designed.
		Recovery job or flow	Both the flow and job or job array continue to execute as designed. The recovery job or flow is triggered.
		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.
	Underrun	Alarm	The alarm is opened. The flow continues execution as designed.
		Recovery job or flow	The flow continues execution as designed. The recovery job or flow is triggered.
		Rerun	Work items that have a dependency on this job or job array are not triggered. The job or job array is rerun as many times as required until the execution time exceeds the underrun time specified.
	Start Failed	Alarm	The alarm is opened. The flow continues execution as designed.
Recovery job or flow		The recovery job or flow is triggered.	
Rerun		The job or job array is rerun as many times as required until it starts successfully.	
Cannot Run	Alarm	The alarm is opened. The flow continues execution as designed.	
	Recovery job or flow	The recovery job or flow is triggered.	
An exit code of n	Rerun	The job or job array is rerun as many times as required until it starts successfully.	
n unsuccessful jobs	Kill	The job array is killed. The flow behaves as designed. The job array status is determined by its exit value.	

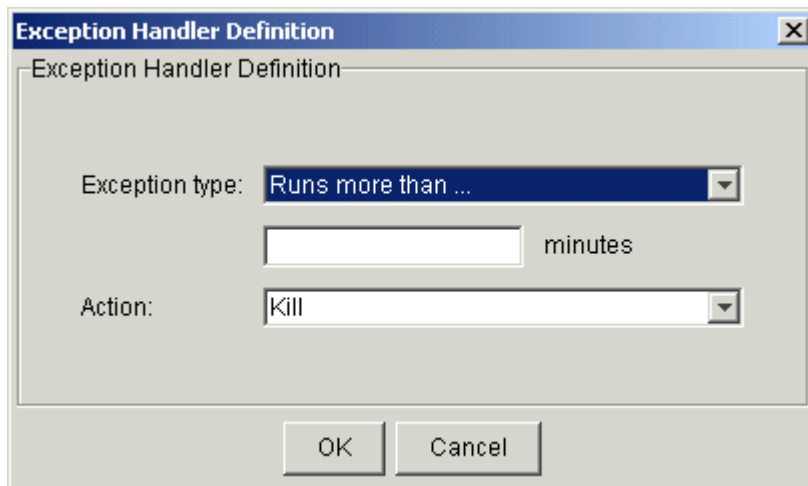
Handling exceptions

When you define a job, job array, flow or subflow, you can specify what exceptions you want Platform Process Manager to watch for, and how you want to handle the exceptions if they happen. You can specify as many exceptions and handlers as you want for any job, job array, flow or subflow. You can handle an exception automatically using the following:

- [Handle exceptions of a job or job array using built-in handlers](#) on page 143
- [Handle exceptions of a subflow using built-in handlers](#) on page 144
- [Handle exceptions with a recovery job](#) on page 145
- [Handle exceptions with a recovery flow](#) on page 146

Handle exceptions of a job or job array using built-in handlers

1. Within the flow definition in the Flow Editor, open the job or job array definition.
2. Click on the Exception Handling tab.
3. Click Add. The Exception Handler Definition dialog appears.



4. In the Exception type field, select the exception you want to handle.
5. If you chose *Runs more than...*, specify the maximum time, in minutes, the job or job array can run before it should be killed.
If you chose *Runs less than...*, specify the minimum time, in minutes, the job or job array can run before it should be rerun.
If you chose *Has exit code*, choose the operator and value that best define the exit code requirement. For example, greater than 5.
If you chose *Number of unsuccessful jobs is ...* choose the operator and value that best define the exit code requirement. For example, greater than 3.
6. In the Action field, select the appropriate exception handler. In most cases, however, the appropriate exception handler is selected for you, as follows:

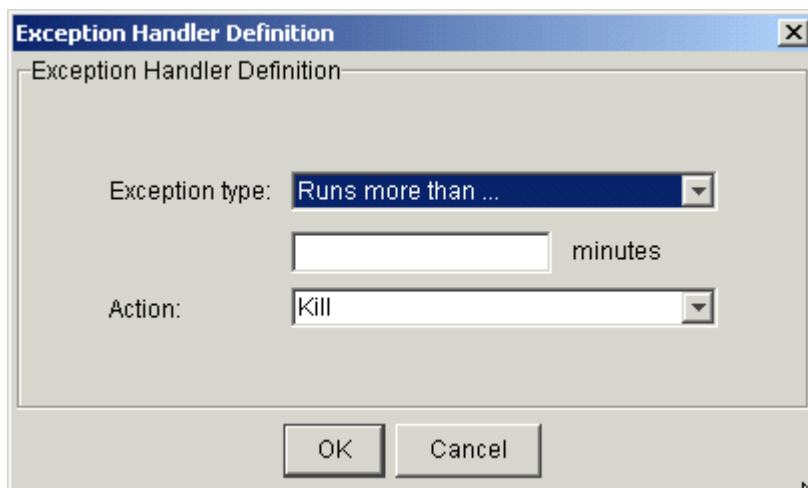
If you monitor for this exception...	This handler is used...
Overrun	Kill
Underrun	Rerun
Exit code	Rerun
Number of unsuccessful jobs is ...	Kill

If you specify a rerun exception, you can specify a number of minutes to delay before rerunning the subflow and the maximum number of times you want the exception handler to rerun the subflow.

7. Click OK. The exception handling specification is added to the list.
8. Repeat steps 3 through 7 until you have finished specifying exceptions to handle then click OK.

Handle exceptions of a subflow using built-in handlers

1. Within the flow definition in the Flow Editor, right-click on the subflow.
2. Select Attributes. The Subflow Attributes dialog appears.
3. Click on the Exception Handling tab.
4. Click Add. The Exception Handler Definition dialog appears.

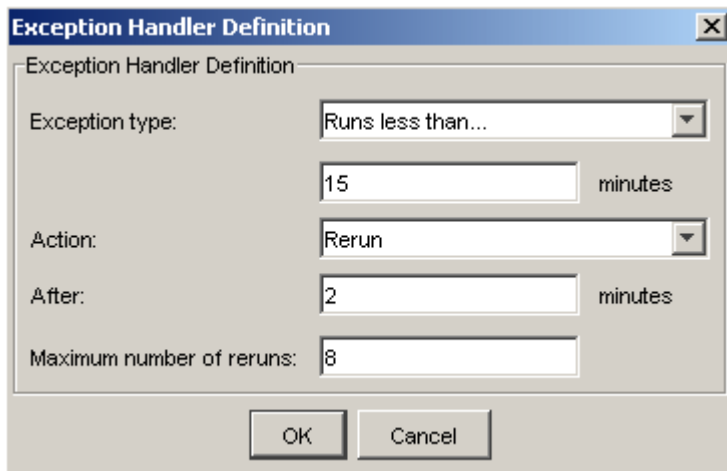


5. In the Exception type field, select the exception you want to handle.
6. Select the corresponding criteria for the Exception type that you specified.
 - If you chose Runs more than..., specify the maximum time, in minutes, the subflow can run before it should be killed or should trigger an alarm.
 - If you chose Runs less than..., specify the minimum time, in minutes, the subflow can run before it should be rerun or should trigger an alarm.
 - If you chose the Flow has exit code, choose the operator and value that best defines the exit code requirements before the subflow is rerun or triggers an alarm. For example, greater than 5.
 - If you chose Number of unsuccessful jobs, choose the operator and value that best defines the requirements before the subflow is killed or triggers an alarm. For example, greater than 3.
 - If you chose The work item has an exit code, choose the operator and value that best defines the exit code requirement before the subflow is rerun. For example, greater than 5.

- In the Action field, select the appropriate exception handler. In most cases, however, the appropriate exception handler is selected for you, as follows:

If you monitor for this exception...	This handler is used...
Overrun	Kill or Alarm
Underrun	Rerun or Alarm
Exit code	Rerun or Alarm
Number of unsuccessful jobs	Rerun or Alarm
The work item has an exit code	Rerun

If you choose to rerun the subflow when an exception occurs, you can delay the rerunning of the subflow by a specified number of minutes and specify the maximum number of times you want the exception handler to rerun the subflow, as shown:



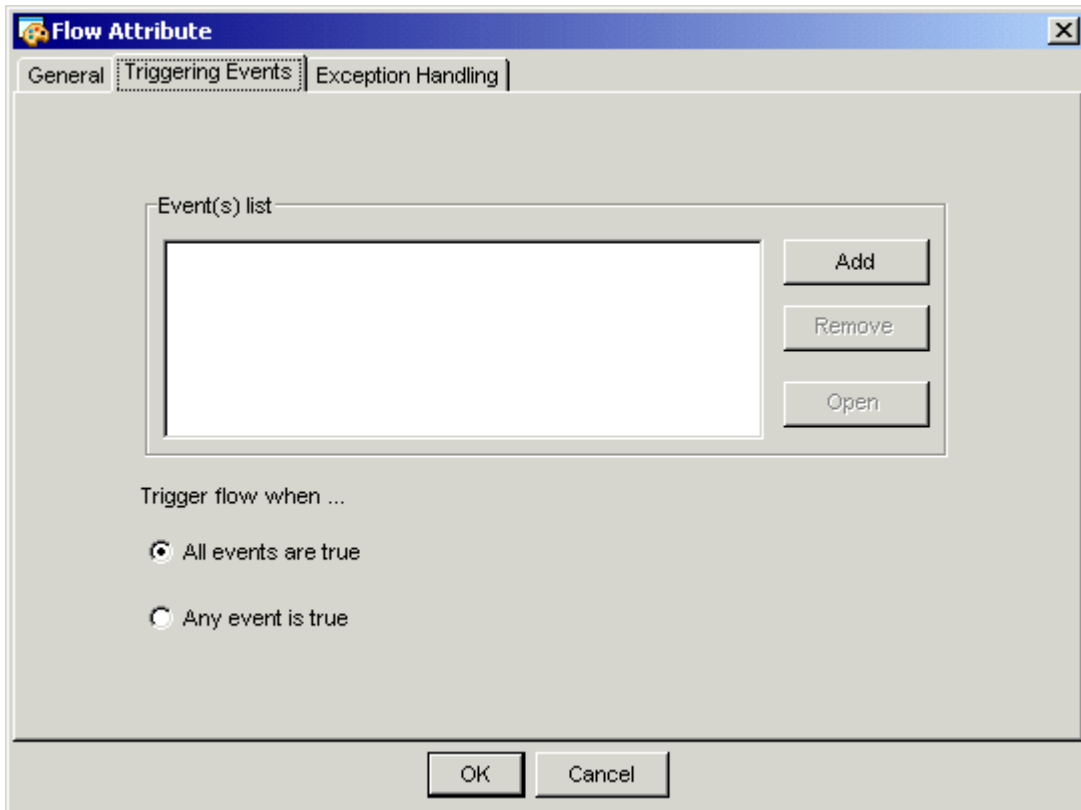
- Click OK. The exception handling specification is added to the list.
- Repeat steps 3 through 6 until you have finished specifying exceptions to handle, then click OK.

Handle exceptions with a recovery job

- Within the flow definition in the Flow Editor, draw both the predecessor and recovery jobs (or job arrays or subflows).
- Change to job dependency mode by clicking the Insert Dependency button.
- Draw job dependencies by left-clicking on the job that must run first, then left-clicking on the recovery job.
- Right-click on the dependency line and select Open Definition. The Event Definition dialog box appears.
- In the Event Type field, select the appropriate exception.
- Click OK.

Handle exceptions with a recovery flow

1. In the Flow Editor, define the recovery flow such that it performs the required functions.
2. When the recovery flow definition is complete, from the Action menu, select Add Flow Attribute...
3. Click the Triggering Events tab.



4. Click Add to define an event to trigger this flow. The Trigger Flow with Events dialog box appears.

Trigger Flow with Events

Select type of event: Proxy Event

Proxy Event

Create proxy for ...

Job Job Array Flow Subflow

Trigger event when ...

Flow name: testflow Owner: bhorner

Duration: 0 minutes

Event type: The flow ends with any exit code

Description: Trigger only when bhorner:testflow fails

OK Cancel

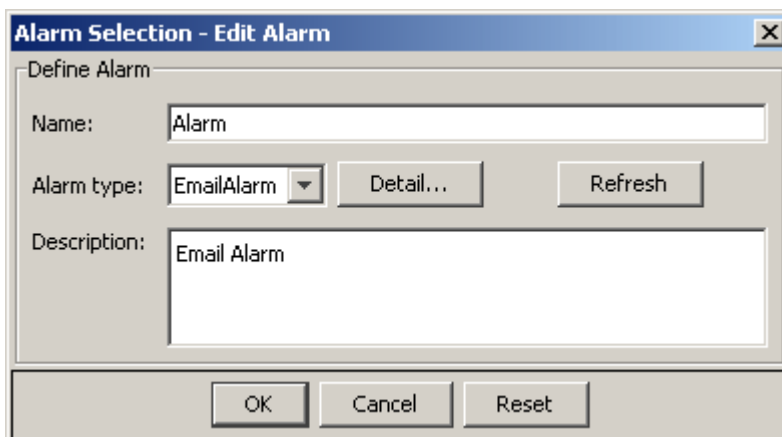
5. In the Select type of event field, select Proxy Event.
6. In the Create Proxy for... field, select Flow.
7. Optional. In the Owner field, specify the name of the user who owns the flow. If you own the flow, you do not need to specify a name—the user name will default to your own.
8. In the Flow name field, specify the name of the flow definition whose condition will trigger this flow. Ensure you specify the name of the flow definition, not its file name. The next occurrence of this flow will trigger the flow you are presently creating.
9. In the Event type field, select the exception condition under which you want this flow to trigger.
10. In the Description field, add any descriptive text that may be used for understanding this event. For example, if this event requires special instructions for operations staff, place those instructions here.
11. Click OK. The Flow-Triggering Event(s) dialog reappears, and the proxy event you defined appears in the list.
12. Click OK. The flow definition is submitted to the Platform Process Manager system, where it will await the appropriate conditions to be run.

Alarms

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. An alarm is opened as a result of the Alarm exception handler. Alarms are configured for your site by your Platform Process Manager administrator. Each alarm has a name and an email address to be notified.

Raise an alarm when an exception occurs within a flow

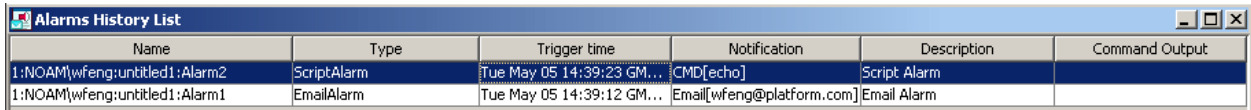
1. In the Flow Editor, with the flow definition opened, change to alarm mode by clicking the Insert Alarm button.
2. Drop the alarm icon in the appropriate place in the workspace.
3. Right-click on the alarm icon in the workspace, and select Open Definition. The Alarm Definition dialog box appears.



4. In the Name field, specify a unique name for the alarm. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_) in the job array name. A unique name is automatically assigned to the alarm, but you can change it to make it more meaningful.
5. In the Alarm type field, select the type of alarm you want to use from the list of configured types. Alarms are configured by your Platform Process Manager administrator. To see an updated list of alarms, click Refresh.
6. Optional. In the Description field, add any descriptive text that may be helpful for understanding this alarm. For example, if this alarm requires special instructions for operations staff, place those instructions here.
7. Click OK.
8. Draw the dependency line from the job or other work item whose exception opens this alarm to the alarm itself.
9. Right-click on the dependency line and select Open Definition. The Event Definition dialog box appears.
10. In the Event Type field, select the exception for which you want to open the alarm.
11. Click OK.

View the opened alarms

1. In the Flow Manager, from the View menu, select Alarms. The View Alarms dialog box appears. It lists all of the open alarms.



Name	Type	Trigger time	Notification	Description	Command Output
1:NOAM\wfeng:untitled1:Alarm2	ScriptAlarm	Tue May 05 14:39:23 GM...	CMD[echo]	Script Alarm	
1:NOAM\wfeng:untitled1:Alarm1	EmailAlarm	Tue May 05 14:39:12 GM...	Email[wfeng@platform.com]	Email Alarm	

Alarms stay in the list of open alarms until the history log file for that time period is archived or deleted. They do not disappear from the list when the problem is fixed.

Insert an alarm in a flow definition

You can use an alarm to send a notification when an exception occurs, or to notify a user when a particular condition is met. The alarm definition specifies how a notification should be sent if the alarm is opened.

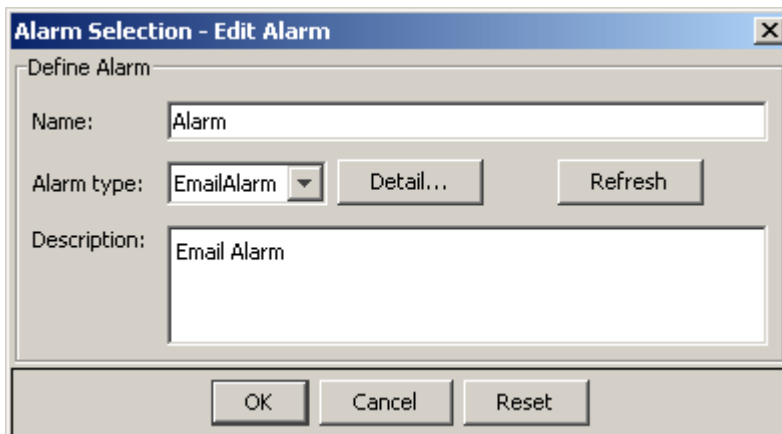
Alarms are configured for your site by your Platform Process Manager administrator. Each alarm has a name and an email address to be notified.

There are two ways to specify an alarm in a flow:

- By inserting an alarm as a successor to a work item in the flow, and specifying a dependency on the work item that opens the alarm when the dependency is met. This method is recommended when it is important to have a visual cue in the flow definition that an alarm is defined in a particular place. You can use this method when you want to notify a user of the successful completion of a work item.
- By specifying an alarm as an exception handler when a particular exception occurs. This method is recommended when you want to maintain an uncluttered view of the work items in your flow, and you are monitoring specifically for a particular exception.

To insert an alarm as a successor to a work item in a flow:

1. Click the Insert Alarm button in the design palette to change to alarm mode.
2. Drop the alarm icon in the appropriate place in the workspace.
3. Right-click on the alarm icon in the workspace, and select Open Definition. The Alarm Definition dialog box appears.



Alarm Selection - Edit Alarm

Define Alarm

Name:

Alarm type:

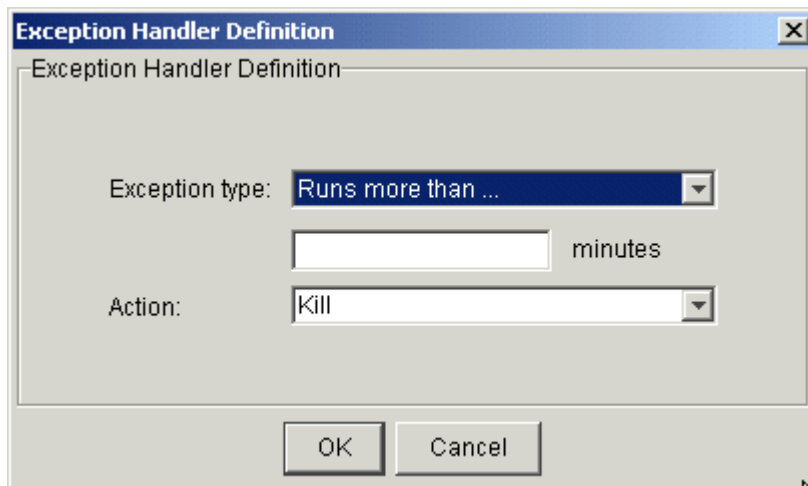
Description:

4. In the Name field, specify a unique name for the alarm. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_) in the alarm name. A unique name is automatically assigned to the alarm, but you can change it to make it more meaningful.
5. In the Alarm type field, select the type of alarm you want to use from the list of configured types. Alarms are configured by your Platform Process Manager administrator. To see an updated list of alarms, click Refresh.
6. Optional. In the Description field, add any descriptive text that may be helpful for understanding this alarm. For example, if this alarm requires special instructions for operations staff, place those instructions here. You can specify a user variable in this field.
7. Click OK.
8. Draw the dependency line from the job or other work item whose exception opens this alarm to the alarm itself.
9. Right-click on the dependency line and select Open Definition. The Event Definition dialog box appears.
10. In the Event Type field, select the exception for which you want to open the alarm.
11. Click OK.

Use an alarm as an exception handler

You can use an alarm as an exception handler, when it is not important to see that an alarm is opened at a particular point in a flow. If the visual cue is important, insert an alarm directly into the flow definition.

1. Open the definition for the work item you want to monitor for the exception.
2. Click on the Exception Handling tab.
3. Click Add. The Exception Handler Definition dialog appears.



4. In the Exception type field, select the exception you want to handle.
5. If you chose *Runs more than...*, specify the maximum time, in minutes, the work item can run before an action should be taken.

If you chose *Runs less than...*, specify the minimum time, in minutes, the work item can run before an action should be taken.

If you chose the *Flow has exit code*, choose the operator and value that best define the exit code requirement. For example, greater than 5.

If you chose *Number of unsuccessful jobs*, choose the operator and value that best define the requirement. For example, greater than 3.

6. In the Action field, select Alarm.
7. In the Alarm type field, select the type of alarm you want to use from the list of configured types. Alarms are configured by your Platform Process Manager administrator. To see an updated list of alarms, click Refresh.
8. Click OK. The exception handling specification is added to the list.

Run your flow

The attributes of a flow include what, if any, events trigger the flow to run, what constitutes successful completion of the flow, the type of email notification to implement regarding the flow, which flow exceptions to monitor for, and what to do if they occur.

When you create your flow definition, you need to know how and when you want the flow to run—will it run on a recurring basis, at a particular time? Or will it run when a file arrives in a particular location? Or a combination of the two? Provided that you want the flow to run under some specific conditions, you need to schedule the flow before you submit it to Platform Process Manager.

The first decision you need to make is how the flow will be triggered. (*Triggering a flow* is the act of telling Platform Process Manager to take a flow definition and create a flow from it.) A flow can be triggered manually or automatically by an event.

If you want to create a flow that can be run more than once, but want it to trigger it manually, see [Create a flow definition to be triggered manually](#) on page 155 for instructions.

If you can specify a recurring schedule for the flow, see [Run a flow at a specific time](#) on page 157 for instructions.

If you want to run a flow whenever something happens to a particular file, see [Run a flow based on file activity](#) on page 159 for instructions.

If the flow is to be triggered by one or more events, you need to specify each of the events that should trigger the flow, and then determine if the flow should trigger only when all events occur, or if any one of the events occurs.

If you want to run the flow only once, see [Run your flow once](#) on page 166 for instructions.

About manual triggers

When you want to create a flow that can be run more than once, but there is no schedule by which the flow should be run, you submit the flow to be triggered manually, and then trigger it manually as required.

You can explicitly trigger any submitted flow from within the Flow Manager at any time, even if the flow definition is on hold. By manually triggering a flow definition that is normally triggered by an event, you create an extra occurrence of the flow.

When you manually trigger a flow, you can pass values to the flow for user variables that are used within the flow.

A flow is also triggered implicitly when you run a flow immediately from the Flow Editor. However, in this case, the flow definition is not stored within Platform Process Manager, and you cannot trigger the flow later from the Flow Manager.

About automatic triggers

There are many ways to automatically trigger a flow:

- Using a time event, which triggers it at a certain time on the specified dates
- Using a file event, which triggers it when a certain file condition occurs
- Using a proxy event, which triggers it when another flow, or work item within another flow reaches a certain state
- Using an exception event, which triggers it when another flow generates a specific exception

Running a flow periodically

You can create a flow that runs on a recurring schedule, by specifying a time event to trigger the flow. The schedule can be as simple as running the flow daily at 9:00 a.m. or it can be as complex as running the flow on the second and fourth Mondays of the month, but not on a holiday. You use calendars to define the schedule criteria.

Running a flow multiple times on a date

You can define a flow to run on multiple dates by using a time event that references a calendar that resolves to multiple dates. However, if you want to run a flow multiple times on any of those dates, you need to define a time expression in the time event. You can do this with a calendar that resolves to one date or with a calendar that resolves to multiple dates.

Running a flow when a file...

You can define a flow that runs when something happens to a specified file by defining a file event to trigger the flow.

Running a flow when another flow...

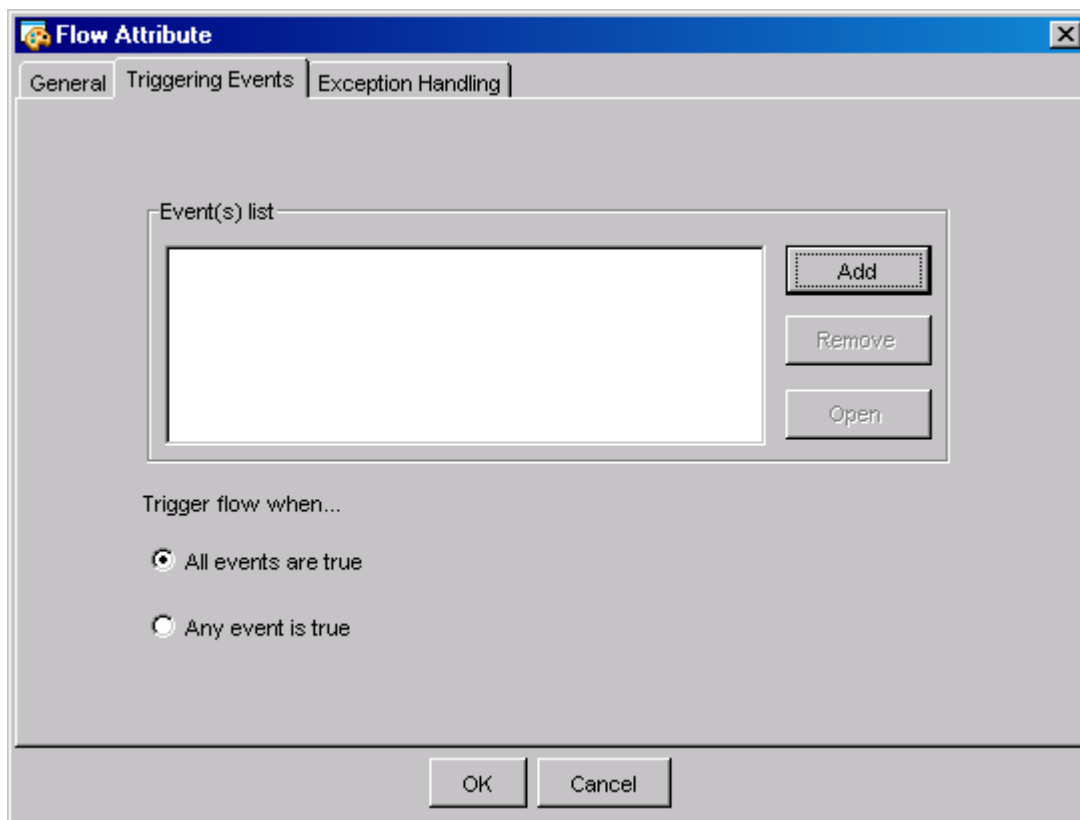
You can define a flow that runs when another flow or work item in another flow completes or reaches a certain condition.

Create a flow definition to be triggered manually

When you want to create a flow that can be run more than once, but there is no repeating schedule by which the flow should be run, you define the flow to be triggered manually. You can trigger it manually from the Flow Manager when it needs to be run. By default, unless you explicitly define an event to trigger a flow, a flow is designed to be triggered manually.

In the flow definition

1. When you have completed defining the flow, right-click in an empty space in the flow definition and select Flow Attribute. The Flow Attributes dialog box appears.
2. Click the Triggering Events tab. Ensure that the list of triggering events is blank.



3. Click OK.
4. From the Action menu, select Submit to submit the flow. The flow will be submitted on hold—you will have to manually trigger it. When you are ready to trigger the flow, open the Flow Manager and expand the tree until you see the flow definition you want to trigger.

From the command line

1. On the command line, type the following:

```
jsub flow_file_name
```

where *flow_file_name* is the full path name of the file containing the flow definition.

Run your flow

2. Press Enter.

Schedule your flow

You can schedule a flow to run at a particular date and time, when a file arrives, or a combination of these. You schedule a flow using an event.

Run a flow at a specific time

In the flow definition

1. In the Flow Editor, open the flow definition.
2. Right-click in an empty space in the flow definition and select Flow Attribute. The Flow Attributes dialog box displays.
3. Click the Triggering Events tab.
4. Click Add to define an event to trigger the flow. The Trigger Flow with Events dialog box displays.
5. In the Select type of event field, select Time Event.
6. In the Calendar name field, select the calendar that resolves to the dates on which you want this flow to run.
7. In the Time zone section, specify the time zone for this time event.
8. In the Hours and Minutes fields, specify the time when you want the flow to start running.

Note:

Do not schedule your flow to start between 2:00 a.m. and 3:00 a.m. on the day that daylight savings time begins (the second Sunday in March), as the flow will not run and any subflows that are scheduled to start after this flow will also not run.

This is because the 2:00 a.m. to 3:00 a.m. hour is removed to start daylight savings time in North America.

9. In the Duration of event field, specify the number of minutes after the specified time that the flow can start. This is useful if there is a time window in which the flow can start. If the flow must start exactly at the specified time, leave the duration at 1 minute.
10. Optional. In the End after ... occurrences field, specify the maximum number of occurrences of this time event before you want it to end.
11. Click OK. The Triggering Event(s) tab reappears, and the time event you defined appears in the list.
12. Click OK.
13. From the Action menu, select Submit to submit the flow. The flow definition is submitted to Platform Process Manager, where it will be scheduled at the specified time, on each day that the specified calendar is true.

From the command line

1. On the command line, type the following:

```
jsub -T time_event flow_file_name
```

where *time_event* is the definition of the time event that triggers this flow and *flow_file_name* is the full path name of the file containing the flow definition.

2. Press Enter.

Run a flow at multiple times on a single date

1. In the Flow Editor, open the flow definition.
2. Right-click in an empty space in the flow definition and select Flow Attribute. The Flow Attributes dialog box displays.
3. Click the Triggering Events tab.
4. Click Add to define an event to trigger the flow. The Trigger Flow with Events dialog box displays.
5. In the Select type of event field, select Time Event.
6. In the Calendar name field, select the calendar that resolves to the dates on which you want this flow to run.
7. In the Time zone section, specify the time zone for this time event.
8. In the Hours and Minutes fields, specify an expression that resolves to the times when you want the flow to start running. Be sure to specify the times as they appear on a 24-hour clock, where valid values for hours are from 0 to 23. For the syntax of the time expression, see [Specifying time expressions](#) on page 158.
9. In the Duration of event field, specify the number of minutes after the specified times that the flow can start. This is especially useful if the flow is triggered by multiple events, requiring that you define a time window in which the flow can start. If the flow must start exactly at the specified time, leave the duration at 1 minute.
10. Optional. In the End after ... occurrences field, specify the maximum number of occurrences of this time event before you want it to end.
11. Click OK. The Triggering Event(s) tab reappears, and the time event you defined appears in the list.
12. Click OK.
13. From the Action menu, select Submit to submit the flow. The flow definition is submitted to Platform Process Manager, where it will be scheduled at the specified times, each day the calendar is true.

Specifying time expressions

You can specify several times for the event to trigger. You can:

- Specify a list of times separated by commas. For example, to run the flow at 2:00 p.m., 3:00 p.m. and 5:00 p.m., specify the following in the Hours field:

```
14, 15, 17
```
- Specify a range of hours. For example, to run the flow every hour from 1:00 a.m. to 5:00 a.m., specify the following in the Hours field:

```
1-5
```
- Specify a combination of the above. For example, to run the flow at 2:00 p.m., 3:00 p.m., and every hour from 7:00 p.m. to 10:00 p.m., specify the following in the Hours field:

```
14, 15, 19-22
```
- Use the Minutes field to modify the value in the Hours field. For example, specify the following in the Hours field:

```
7, 9, 11-13
```

and the following in the Minutes field:

```
15, 30
```

to run the flow at 7:15, 7:30, 9:15, 9:30, 11:15, 11:30, 12:15, 12:30, 13:15 and 13:30.

- Use an asterisk (*) in the Hours field to specify every hour, or in the Minutes field to specify every minute. For example, to run a flow every hour, in the Hours field, specify an asterisk (*).

Run a flow based on file activity

In the flow definition

1. In the Flow Editor, open the flow definition.
2. Right-click in an empty space in the flow definition and select Flow Attribute. The Flow Attributes dialog box displays.
3. Click the Triggering Events tab.
4. Click Add to define an event to trigger the flow. The Trigger Flow with Events dialog box displays.
5. In the Select type of event field, select File Event.
6. In the File name field, specify the full path name of the file as the Platform Process Manager Server sees it, that is to be monitored for the activity, or click Browse to locate the file in the file system.

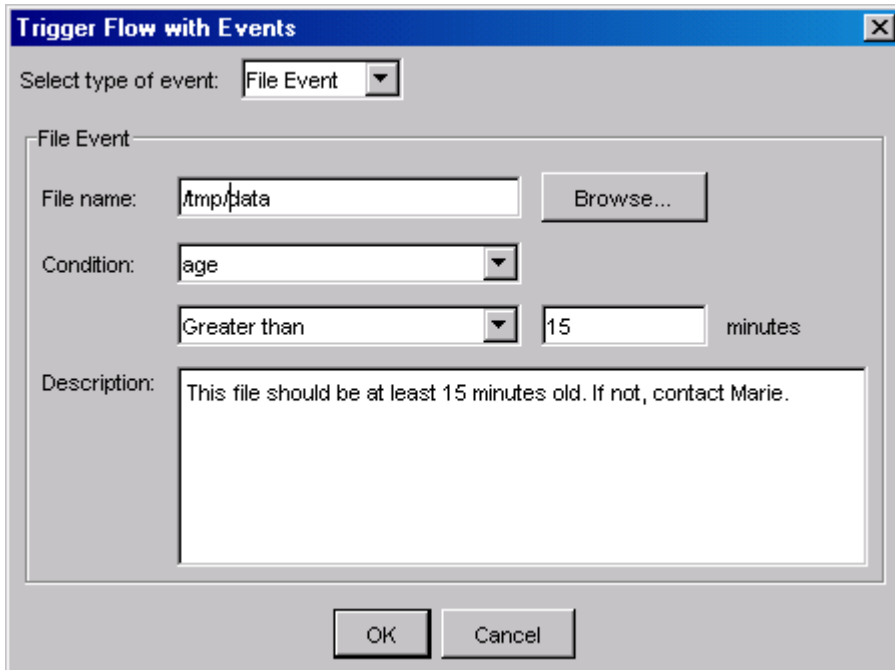
When specifying the file name, you can also specify wildcard characters: * to represent a string or ? to represent a single character. For example, *.dat* matches abc.dat, another.dat and abc.dat23. S??day* matches Sat days. tar and Sundays. dat.*e matches smil e.

For arrival/exist/size/age events, every matched file triggers the event. For example, if you specify a dependency on the arrival of *.tar, the dependency is met when 1. tar arrives, and again when 2. tar arrives.

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.

You can also specify a variable for the file name, provided your system is configured to support them. See [User variables within a flow definition](#) on page 81 for more information about user variables.

7. In the Condition field, specify the condition that matches the activity you want to monitor the file for. Choose from the following:
 - exists
 - does not exist
 - age
 - arrival
 - size
8. Depending on the condition you choose, you may need to further qualify the condition with the input fields that follow the condition. For example, when you choose size, you need to specify an operator (greater than, and so on) and the size, in bytes.



9. Click OK. The Triggering Event(s) tab reappears, and the file event you defined appears in the list.
10. Click OK.
11. From the Action menu, select Submit to submit the flow. The flow definition is submitted to Platform Process Manager, where it is triggered when the specified file event is true.

Examples

- Triggering when a file exists

The following file event triggers the flow when the file /tmp/core exists:



When triggering a flow when a file exists, keep the following in mind:

- Platform Process Manager polls periodically to see if the file exists. When it does, the flow is triggered. The default polling interval is 30 seconds. Check with your Platform Process Manager administrator to see what your polling interval is set to.
- Unless the file is deleted, after the flow is triggered, it will trigger again each time Platform Process Manager polls and finds the file exists, unless you combine this event with another such as a time event.
- Triggering when a file is deleted

The following file event triggers the flow when the file tmp/update is deleted:



After the flow is triggered, it will trigger again each time Platform Process Manager polls and finds the file does not exist, unless you combine this event with another such as a time event.

- Triggering when a file is more than 15 minutes old

The following file event triggers the flow when the file /tmp/data is more than 15 minutes old:

File name:

Condition:

minutes

- Triggering whenever a file arrives

The following file event triggers the flow every time a tar file arrives in the tmp directory:

File name:

Condition:

From the command line

1. On the command line, enter the following command:

```
jsub -F "file_event" flow_file_name
```

where *file_event* is the definition of the file event that triggers this flow and *flow_file_name* is the full path name of the file containing the flow definition. For example:

```
jsub -F "arrival(/tmp/*.tar)" testflow.xml
```

Run a flow when another flow...

You can run a flow when another flow reaches a certain condition, or you can run a flow when a work item in another flow reaches a certain condition. In either case, you use a proxy event to trigger the flow. As its name indicates, the proxy event acts as a proxy in the current flow for another flow or a work item that runs within another flow

Run a flow when another flow completes

1. In the Flow Editor, open the flow definition.
2. Right-click in an empty space in the flow definition and select Flow Attribute. The Flow Attributes dialog box displays.
3. Click the Triggering Events tab.
4. Click Add to define an event to trigger the flow. The Trigger Flow with Events dialog box displays.
5. In the Select type of event field, select Proxy Event.
6. In the Create proxy for... field, select Flow.
7. In the Flow name field, specify the name of the flow definition whose condition will trigger this flow. Ensure you specify the name of the flow definition, not its file name. The next occurrence of this flow will trigger the flow you are presently creating.
8. Optional. In the Owner field, specify the name of the user who owns the flow. If you own the flow, you do not need to specify a name—the name will default to your own.
9. In the Duration field, specify the number of minutes in the past to detect the proxy event.
10. In the Event type field, select The flow completes successfully.
11. In the Description field, add any descriptive text that may be used for understanding this event. For example, if this event requires special instructions for operations staff, place those instructions here.

Trigger Flow with Events

Select type of event: Proxy Event

Proxy Event

Create proxy for...

Job Job Array Flow Subflow

Trigger event when...

Flow name: testflow Owner: NOAM\jheys

Duration: 0 minutes

Event type: The flow completes successfully

Description: Trigger only when jheys:testflow is successful.

OK Cancel

12. Click OK. The Triggering Event(s) tab reappears, and the proxy event you defined appears in the list.

13. Click OK.

14. From the Action menu, select Submit to submit the flow. The flow definition is submitted to Platform Process Manager, where it is triggered when the specified file event is true.

Examples

- Triggering when a flow has exit code greater than 3:

The following proxy event triggers the flow when the flow testflow exits with an exit code greater than 3:

Trigger event when...

Flow name: testflow Owner: NOAM\jheys

Duration: 0 minutes

Event type: The flow has exit code...

Greater than 3

- Triggering when 5 or more jobs in a flow fail

The following proxy event triggers the flow when 5 or more jobs in the flow testflow fail:

From the command line

1. On the command line, to achieve the same results, type the following:
`jsub -p "flow(numexit(bhorner:testflow)>=5)"`
2. Press Enter.

Calculation of number of jobs in a flow

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 work items in the flow, such as events or alarms.

Run a flow when a proxy job completes

1. In the Flow Editor, open the flow definition.
2. Right-click in an empty space in the flow definition and select Flow Attribute. The Flow Attributes dialog box displays.
3. Click the Triggering Events tab.
4. Click Add to define an event to trigger the flow. The Trigger Flow with Events dialog box displays.
5. In the Select type of event field, select Proxy Event.
6. In the Create proxy for... box, leave the default at Job.
7. In the Job name field, specify the fully qualified name of the job, in the following format:

flow_name.subflow_name.job_name

If the job is not defined within a subflow, simply specify the flow name and the job name, separated by a colon.

Note:

You cannot specify a proxy for a manual job.

8. If the flow containing the job is not owned by your user ID, in the Owner field, specify the user ID that owns the flow containing the proxy job.
9. In the Duration field, specify the number of minutes in the past to detect the proxy event.
10. In the Event type field, select the type of dependency you want to use to trigger the flow, and the appropriate operator and values.
11. In the Description field, add any descriptive text that may be used for understanding this event.

12. Click OK. The Triggering Event(s) tab reappears, and the proxy event you defined appears in the list.
13. Click OK.
14. From the Action menu, select Submit to submit the flow. The flow definition is submitted to Platform Process Manager, where it is triggered when the specified file event is true.

From the command line—trigger when job fails

1. On the command line, to trigger when the job fails, type the following:

```
jsub -p "job(exit(bhorne: testflow:J2))"
```
2. Press Enter.

Run your flow once

When you have finished creating a flow definition, you can run the flow immediately from the Flow Editor. You may want to do this to test the job sequence in a flow, or when the flow is to be run only once, and not on a recurring schedule. If you plan to run a flow again, or on a recurring basis, ensure that you submit the flow definition.

From the Flow Editor

1. Ensure that the Platform Process Manager Server is up and running.
2. When you have completed the flow definition, from the Action menu, select Run Now.
3. In the Run Flow Confirmation dialog, click Yes. The flow will run once. A copy of the flow definition is not retained in the Flow Manager. You can view the flow from your adhoc folder in the Flow Manager.

From the command line

1. On the command line, type the following:

```
jrun flow_file_name
```

where *flow_file_name* is the full path name of the file containing the flow definition.

2. Press Enter.

Submit your flow definition

Until you submit a flow definition, Platform Process Manager is not aware of it. Submitting a flow definition places it under the control of Platform Process Manager. Once a flow definition is submitted, Platform Process Manager determines when the flow is to run, and triggers it as appropriate.

When you have completed defining your flow, it is recommended that you save the flow before you submit it.

You can optionally submit your flow with versioning comments, which makes it easier to track different versions of the flow.

- [Submit your flow without version comments](#) on page 167.
- [Submit your flow with version comments](#) on page 167.

When the flow is submitted successfully, you will receive a confirmation message with the version number of the flow. New flows are submitted as version 1.0.

After submitting the flow to Platform Process Manager, it is not published by default. To publish a flow, run the Flow Manager as an administrator, right-click the flow, and select Publish. To unpublish the flow, right-click the published flow and select Unpublish.

Submit your flow without version comments

1. In the flow editor, select the flow definition that you want to submit.
2. From the Action menu, select Submit.
3. If the flow editor displays a Flow exists dialog, specify the method of flow submission.
 - To assign the flow as a new version, click Update.
The new flow exists as a new version of the existing flow.
 - To assign the flow as a duplicate, click Duplicate.
The new flow exists as a separate, independent flow.

Note:

If you delete a flow, then later add a flow with the same name as the deleted flow, the new flow is treated as a new flow rather than a new version of the previous flow.

Submit your flow with version comments

1. In the flow editor, select the flow definition that you want to submit.
2. From the Action menu, select Submit with comment...
The Set Comments window displays.
3. In the Comments for the flow field, specify the comments for the flow version.
4. Click OK to submit the flow.
If there is a flow with the same name, the flow editor displays a Flow exists dialog.
5. If the flow editor displays a Flow exists dialog, specify the method of flow submission.

Run your flow

- To assign the flow as a new version, click Update.
The new flow exists as a new version of the existing flow.
- To assign the flow as a duplicate, click Duplicate.
The new flow exists as a separate, independent flow.

Note:

If you delete a flow, then later add a flow with the same name as the deleted flow, the new flow is treated as a new flow rather than a new version of the previous flow.

Control a Flow

When you have created a flow definition and scheduled it, or submitted it to be triggered manually, a copy of the flow definition is stored within the Process Manager system. You can trigger a flow at any time once its definition is known to Process Manager. You trigger the flow using Flow Manager or the command line interface.

When you trigger a flow definition manually, when you run a flow definition immediately, or when a flow definition is triggered automatically via an event, a flow is created. You can view and control these flows from within the Flow Manager.

About the Flow Manager

You use the Flow Manager to view the status of flows, jobs, job arrays and subflows that are currently in the system, or have run recently. You also use the Flow Manager to:

- Trigger a flow
- Place a flow definition on hold, or release it from hold
- Kill, suspend, resume or rerun a flow
- Kill, run or rerun a job
- Force a job complete

The Platform Process Manager Server must be running before you can open the Flow Manager.

About Flow Manager views

The Flow Manager user interface consists of two panes:

The left-hand pane controls the flow data that is displayed in the right-hand pane. You can look at the data in the following views:

- By Definition—Displays flow definitions organized by the user who submitted them.

You can see flow definitions in the tree in the form:

- *flow_definition_submitter:flow_name* [(On Hold)][(Published)]
 - *flow_id (flow_owner)(flow_state)*
- By Flow User—flow definitions and flows are sorted by user who owns them: the user who triggered the flow.

When the By Flow User view is selected, the left-hand pane lists all the flow definitions known to the Platform Process Manager Server, and any running flows. They are grouped by user, in an expandable tree structure, similar to Windows Explorer.

You can see flows in the tree in the form:

- *flow_owner*
 - *flow_id (flow_definition_submitter:flow_name)(flow_state)*
- By State—flows are sorted by their current state.

When the By State view is selected, the left-hand pane lists all the flows in the system, grouped by state. This allows you to look only at Exited flows, for example. The states are listed in a tree structure, similar to Windows Explorer.

You can see flows in the tree in the form:

flow_state

- *flow_owner*
 - *flow_id (flow_definition_submitter:flow_name)*
- By Event—flows are sorted by their triggering events.

When the By Event view is selected, the left-hand pane lists all the flows in the system, grouped by triggering event. This allows you to see all flows that are triggered at a particular time, or all flows that are waiting for a particular file to arrive. The events are listed in a tree structure, similar to Windows Explorer.

You can see flows in the tree in the form:

event_name

- *flow_definition_submitter:flow_name [(On Hold)] [(Published)]*
- • *flow_id (flow_owner) (flow_state)*

When a view is selected, the right-hand pane shows the graphical illustration of the currently selected flow definition or flow.

The left-hand pane also contains an optional legend, which displays the meaning of each of the states you may see in the left pane.

- You can also view the following information by selecting it in the left-hand pane:
 - Alarms—the current list of alarms that have been opened
 - Manual jobs—the list of manual jobs requiring acknowledgement

System status

You can view the current status of the Platform Process Manager system from the View menu, by selecting System Status. The System Status view displays the status of the Platform Process Manager agents—the hosts that run the jobs.

Service	Status	Run on Host	Host type	Description
Agent	ok	curie	SOL732	

List of alarms

When you choose to view the alarms (from the View menu, select Alarms), a window shows all of the open alarms in the system. Open alarms remain in the list until the history log file containing the alarm is archived or deleted.

Name	Type	Trigger time	Notification	Description	Command Output
1:NOAM\wfeng:untitled1:Alarm2	ScriptAlarm	Tue May 05 14:39:23 GM...	CMD[echo]	Script Alarm	
1:NOAM\wfeng:untitled1:Alarm1	EmailAlarm	Tue May 05 14:39:12 GM...	Email[wfeng@platform.com]	Email Alarm	

List of manual jobs

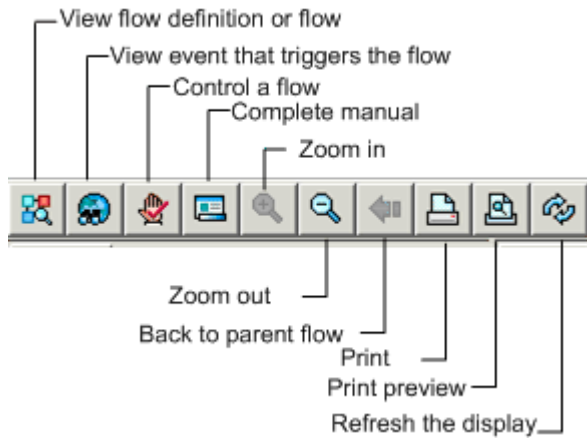
When you choose to view manual jobs (from the View menu, select Manual Jobs), a window displays all of the manual jobs in the Platform Process Manager system. Those that are waiting for acknowledgement now have a check mark in the Completion Required field.

ID	User	Name	Completion Required	Description
82	bhorner	printtest:M1	<input type="checkbox"/>	
81	bhorner	printrpt:M1	<input checked="" type="checkbox"/>	

About the toolbar

The Flow Manager toolbar looks like this:

Control a Flow



Real-time data

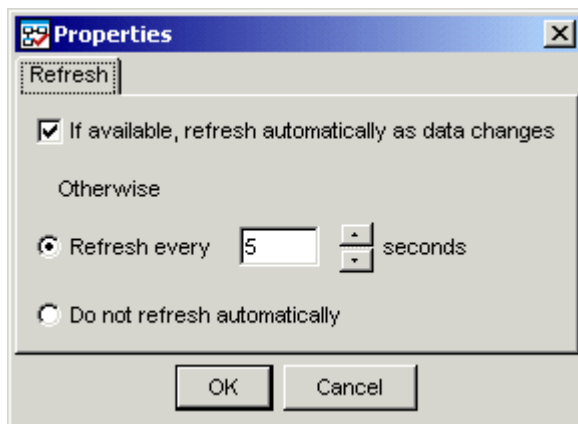
The data displayed in the Flow Manager is intended to reflect real-time status of the flows in the system. The Flow Manager display is set to refresh automatically every 5 minutes. Depending on the number of flows in the system, you may want to change that value, or turn off the automatic refresh entirely.

Refresh the data displayed manually

1. Click the refresh button or from the View menu, select Refresh.

Change the automatic refresh option

1. From the File menu, select Properties. The Set Refresh Rate dialog appears.



2. If you want the data to refresh automatically, leave Refresh automatically checked.
3. Specify the number of seconds between refreshes of the data.
4. Click OK.

Print data

From any view in the Flow Manager, you can print the data displayed. Particularly when viewing a flow or flow definition, you may want to preview the data to be printed.

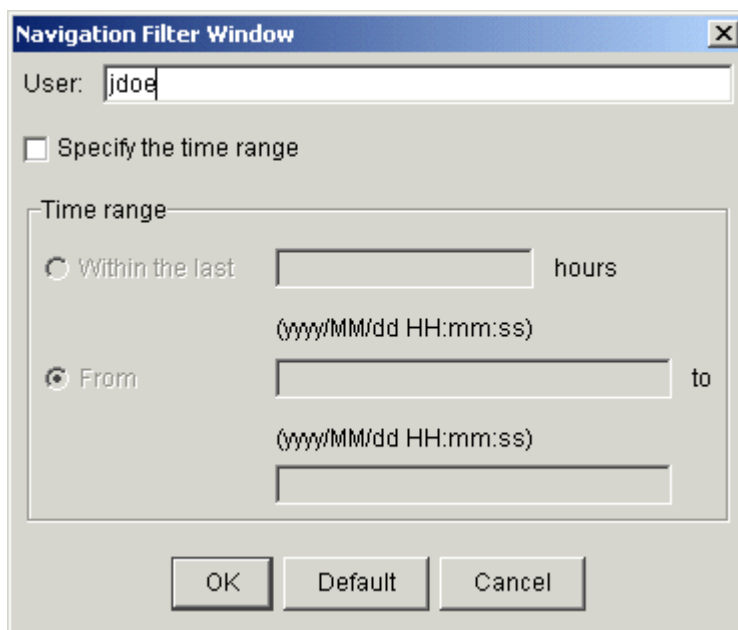
1. From the File menu, select Print Preview to see how your flow will look on paper.
You can adjust the spacing in your flow to avoid breaking icons at a page boundary.
2. From the File menu, select Print and click OK.

Filter the data displayed in the tree view

You can filter the data displayed in the tree view, to limit the data to that which meets your needs. This is especially useful when your Platform Process Manager system runs many hundreds of flows and you do not want to download unnecessary amounts of data to your client machine. The following are some of the ways you may want to filter the data: When viewing flows by state, and you want to limit the flows displayed to those owned by a particular user, when you want to limit the flows displayed to those that ran during the last hour or two, or when you want to limit the flows displayed to those that ran within a particular time window.

Limit the flows displayed to those owned by a user

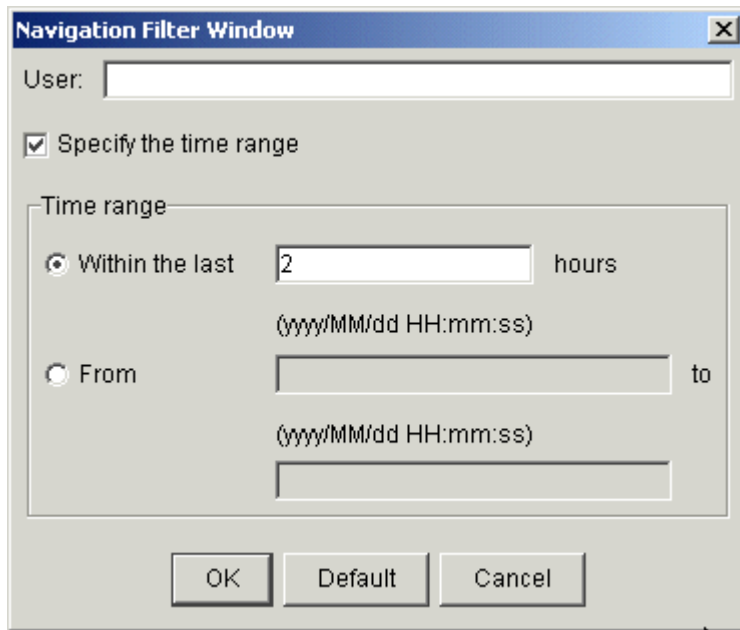
1. In the Flow Manager, select the view of the data you want—by state or by event by clicking the appropriate tab at the top of the left pane.
2. From the View menu, select Set Filter....
3. In the User field, specify the name of the user whose flows you want to see.



4. Click OK. The view of the flows is refreshed with the new filter applied.

Limit the flows displayed to last x hours

1. In the Flow Manager, select the view of the data you want—by state or by event by clicking the appropriate tab at the top of the left pane.
2. From the View menu, select Set Filter....
3. Click Specify the time range.
4. In the Within the last field, specify the number of hours of flow data to include.



5. Click OK. The view of the flows is refreshed with the new filter applied.

Limit the flows displayed to a time period

1. In the Flow Manager, select the view of the data you want—by state or by event by clicking the appropriate tab at the top of the left pane.
2. From the View menu, select Set Filter....
3. Click Specify the time range.
4. Select From.
5. In the first input field, specify the starting date and time of the time period for which you want to display flows.
6. In the second input field, specify the ending date and time of the time period for which you want to display flows.

The screenshot shows a dialog box titled "Navigation Filter Window" with a close button (X) in the top right corner. It contains a "User:" label followed by an empty text input field. Below this is a checked checkbox labeled "Specify the time range". Underneath the checkbox is a section titled "Time range" which contains two radio button options. The first option is "Within the last" followed by an empty text input field and the word "hours", with the format "(yyy/MM/dd HH:mm:ss)" below it. The second option is "From" followed by a text input field containing "2002/07/01 12:00:00", the word "to", another text input field containing "2002/07/31 12:00:00", and the format "(yyy/MM/dd HH:mm:ss)" below it. At the bottom of the dialog are three buttons: "OK", "Default", and "Cancel".

7. Click OK. The view of the flows is refreshed with the new filter applied.

Trigger a flow

When you create a flow definition that is not triggered automatically by an event, it needs to be triggered explicitly before it can run. You can trigger it manually from the Flow Editor by specifying Run Now. However, the flow runs only once, and the definition is not stored in the Platform Process Manager system where it can be run again. If you want to be able to run a flow more than once, but to trigger it manually as required, you submit the flow definition, specifying that it will be triggered manually. The flow definition is submitted to Platform Process Manager, where it awaits a manual trigger. When you trigger a flow, you can pass parameters to the flow using user variable and value pairs. The values are available to any job in the flow, for the life of the flow. For example, you can use this to specify the path to the data files to be processed.

Trigger a flow

From the Flow Manager

1. In the Flow Manager, select By User.
2. In the tree view of the Flow Manager, expand the tree until you see the flow definition you want to trigger.
3. Right-click on the flow definition, and select Trigger. A flow is created and run.

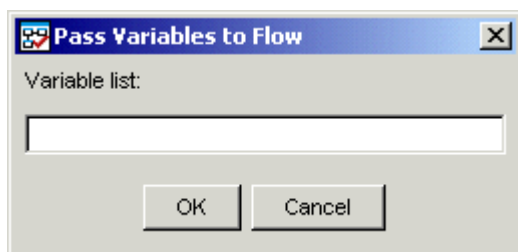
From the command line

1. On the command line, type the following:
`jtrigger flow_definition_name`
 where *flow_definition_name* is the name of the flow definition you want to trigger.
2. Press Enter.

Trigger a flow, passing it values for variables

From the Flow Manager

1. In the tree view of the Flow Manager, expand the tree until you see the flow definition you want to trigger.
2. Right-click on the flow definition, and select Trigger, then select With Variables. The Pass Variables to Flow dialog box appears.

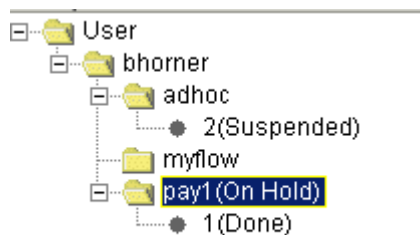


3. Specify the parameters to pass in the following format:
`variable=value;variable=value...`
4. Click OK. A flow is created and run.

View a flow definition and specify versioning options

When working within the Flow Manager, you are not limited to working with flows—you can also view the definition of a flow.

1. In the Flow Manager, select By Flow User.
2. Under the appropriate user ID in the tree view, locate the flow definition you want to view. It is listed by name, above every occurrence of the flow that is in the system:



In the above example, *pay1* is the flow definition. Below it is *1*, the ID of the flow that just completed.

3. Right-click on the definition name, and select View Flow. The flow definition is displayed in the right-hand pane. You cannot edit the definition here—you can only change the definition in the Flow Editor.

View Version

You can view the version history of the flow to see the different versions of the flow that are submitted.

You may also set any eligible flows to be the default version. 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.

1. In the Flow Manager, select By User.
2. Under the appropriate user ID in the tree view, locate the flow definition you want to view.
3. Right-click on the definition name, and select View Version.

The Flow Version window displays. This window initially displays the version of the corresponding flow definition when the flow instance is instantiated (Current Version), and the latest version of the corresponding flow definition (Latest Version).

4. Click View history to expand the version history list.

A scroll panel with the flow version, submission time, and comments is displayed. The comments shows any comments made with the Submit with comments... option for submitting a flow.

5. Optional. Specify the default version options.
 - In a static subflow, to set a flow version to be the default version, select a version in the expanded version history list and click Set Default Version.

Note:

The **Set Default Version** might not be available under certain circumstances. For example, you will not be able to set a default version when viewing the history of a flow instance from the tree view.

- In a dynamic subflow, specify the default version update options in Update to default version.

You can select automatic or manual updates, and you can choose to manually update the dynamic subflow now.

View Statistics

You can view relevant flow instance information summary:

1. In the Flow Manager, select By User.
2. Under the appropriate user ID in the tree view, locate the flow definition you want to view.
3. Right-click on the definition name, and select View Statistics.

View inter-flow relationships

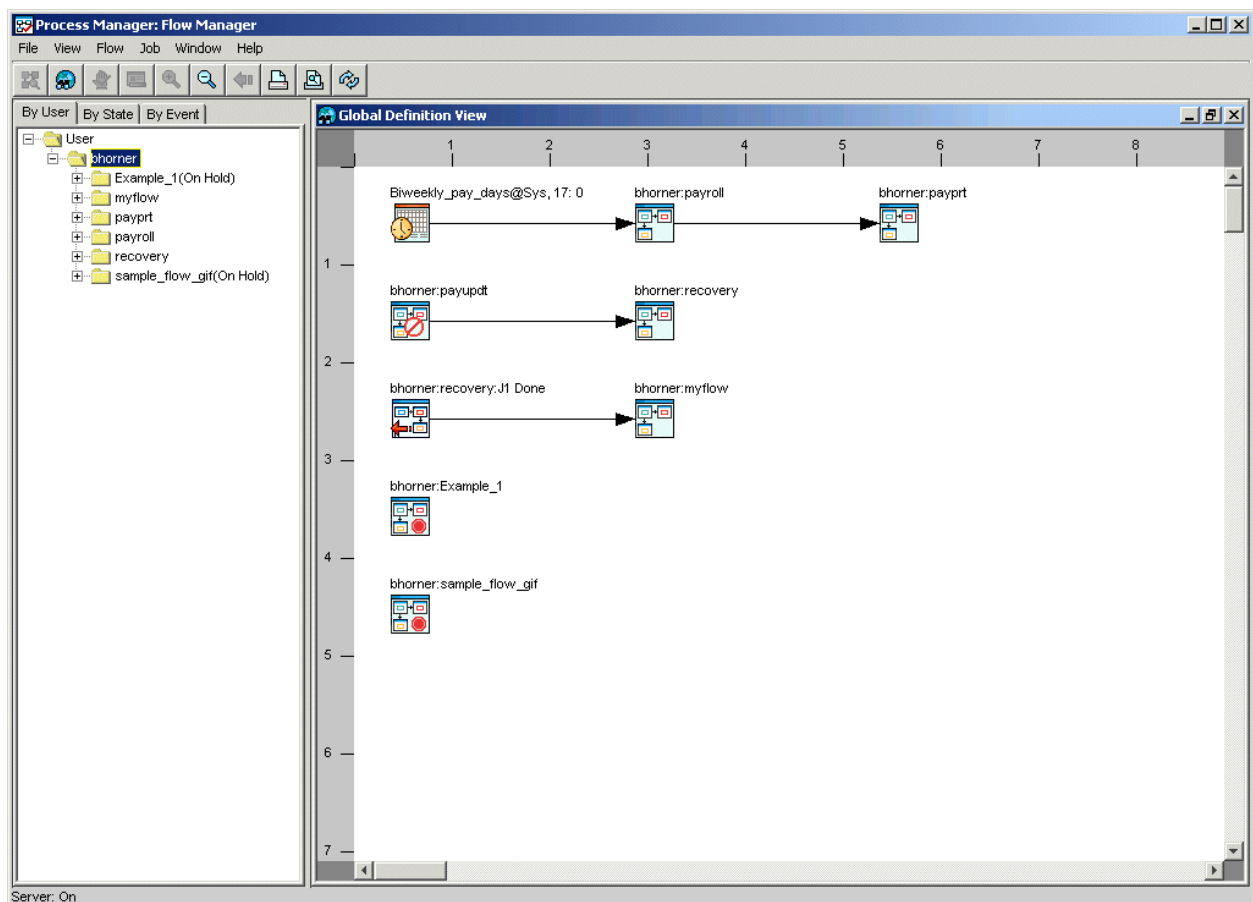
You can view all of your flow definitions in Platform Process Manager using the Global View option in the Flow Manager. This allows you to see proxy dependencies between flows.

The global view displays inter-flow relationships in a graphical format: it shows a graphical representation of each of the events that trigger a flow, it shows dependency lines between flows, and it shows curved, dotted dependency lines that represent dependencies on proxies within a flow.

You use the global view to see how your flow definitions relate to each other, and to see what triggers each flow. From the global view, you can:

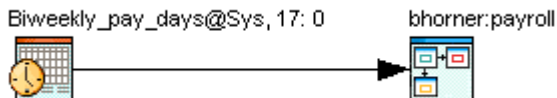
- See the entire business process
- See what events are used to trigger each flow
- See proxy dependencies between flows
- Manually complete an inter-flow dependency
- See flows that other flows are dependent upon that have not yet been submitted to Platform Process Manager, or are on hold, waiting for a manual trigger

The following is an example of a global view:



View the events that trigger a flow:

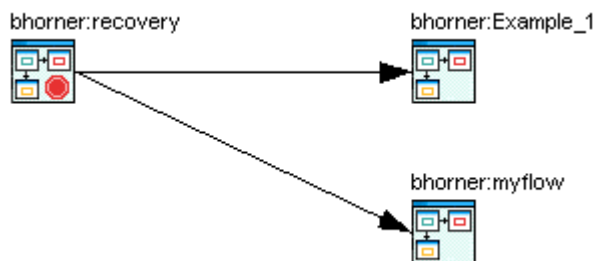
1. Open the Flow Manager.
2. From the View menu, select Global View. The Global Definition View appears, where the flow definitions are shown graphically. An event that triggers a flow is displayed to the left of the flow, with a dependency arrow drawn between them. In the following example, the flow is triggered by a time event:



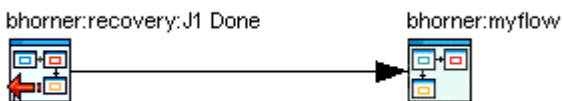
View proxy dependencies that trigger flows

1. Open the Flow Manager.
2. From the View menu, select Global View. The flow definitions are shown graphically in the right-hand pane. An event that triggers a flow is displayed to the left of the flow, with a dependency arrow drawn between them, as shown:

A proxy event that triggers a flow is displayed to the left of the flow, with a dependency arrow drawn between them. In the following example, both Example_1 and myflow are dependent on the completion of recovery. These dependencies were defined as proxy events on the flow recovery.

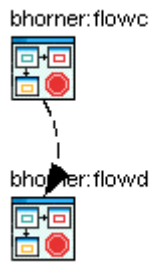


In the following example, myflow triggers when J1 in recovery completes. This dependency was defined as a proxy event on recovery: J1.



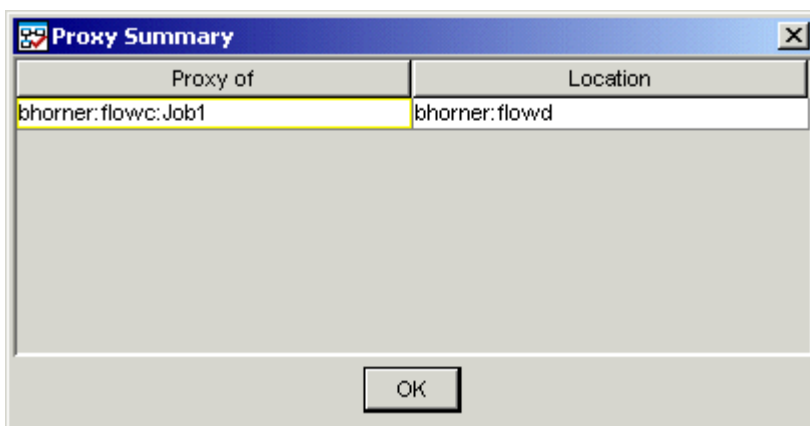
View proxy dependencies within a flow

1. Open the Flow Manager.
2. From the View menu, select Global View. The flow definitions are shown graphically in the right-hand pane. A proxy dependency on a work item within a flow is represented by a curved, dotted line running from the flow containing the actual work item to another flow containing the proxy to the work item. Refer to the following example:



In the above example, flowd has a dependency on a work item in flowc.

- To see a description of the proxy dependency, double-click on the curved arrow. The Proxy Summary dialog appears, displaying information about the proxy event:



Manually complete an inter-flow dependency

- Open the Flow Manager.
- From the View menu, select Global View. The flow definitions are shown graphically in the right-hand pane. An event that triggers a flow is displayed to the left of the flow, with a dependency arrow drawn between them.
- Right-click on the dependency line representing the dependency you want to complete.
- Select Complete Dependency. The dependency is completed, removing the dependency only for this occurrence of the flow. Completing a dependency has no impact on the flow definition.

Note:

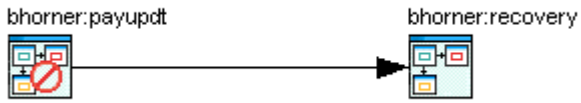
Removing a dependency does not automatically make a flow eligible to run—if it has other dependencies, it will wait for those to be met, unless you complete them also.

View dependencies on flows that do not exist or are on hold

- Open the Flow Manager.

- From the View menu, select Global View. The flow definitions are shown graphically in the right-hand pane. An event that triggers a flow is displayed to the left of the flow, with a dependency arrow drawn between them.

When a flow has a dependency on another flow whose definition does not yet exist in Platform Process Manager, the global view looks like this example, where `recovery` has been submitted, but `payupdt` has not:



When a flow has a dependency on another flow that is on hold, awaiting a manual trigger, the global view looks like this example, where `payprt` depends on `Example_1`, which must be triggered manually:



Determine the status of jobs in a flow

When you view a running flow, you can see the progress as a job runs. There are multiple ways to determine the current status of a job:

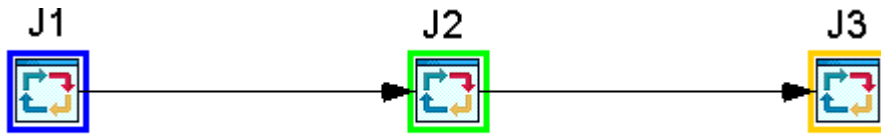
- By the color of the box around the work item icon
- By the text displayed when you place your mouse over the work item icon
- By the state shown in the Runtime Attributes dialog

Colored border around the icon

The Flow Manager uses a colored border around the job, job array, subflow and manual job icons to indicate their current state.

When the Color is ...	The State is ...	Which means the Work Item ...
Blue	Done	Completed successfully
Red	Exit	Failed
	Killed	Was killed when the flow was killed
Green	Running	Is currently running
	Waiting for completion	Manual Job completion required
Brown	Pending in LSF	Is pending in LSF—the job is waiting in a queue for scheduling and dispatch.
		Platform Process Manager considers these jobs to be the same as Running. Unless otherwise specified, anything that applies to Running jobs also applies to jobs that are Pending in LSF.
Yellow/orange	Waiting	Is waiting to be dispatched, or was suspended while it was waiting
	Initializing	Is still initializing
Black	Suspended	Was suspended after the flow started to run
	Initializing Suspended	Was suspended while the flow was initializing
	Waiting Suspended	Was suspended while the job was waiting to be dispatched
Gray	On hold	Is held in the flow definition—it cannot be run

In the following example, the flow testflow was running. J1 with a blue border completed successfully. J2 with a green border is currently running. J3 has a yellow border—it is waiting.



Fly-over mouse text

When you place your mouse over a work item within a flow, a popup window appears for a short period of time that describes the state of the work item. For example:



Runtime attributes

When you view the runtime attributes of a work item, its state is displayed, with other information about the work item. For example:

Runtime Attributes	
[Job name]	: J1
[Job ID]	: 1456
[Submitter]	: bhorne
[Command]	: sleep 5
[State]	: Done
[Exit code]	: 0
[CPU usage]	: 0.23
[Start time]	: Wed Aug 14 03:45:11 GMT 2005
[Finish time]	: Thu Aug 15 17:39:39 GMT 2005

Manually complete a dependency

You can manually complete a dependency, so that a work item no longer needs to wait for that dependency to be met. You can select any dependency within a flow and complete it. This is useful if the duration on a file event has expired, and the file is now available, or if you determine that the dependency can never be met, and there is a case for running the work item anyway. You can also manually complete a dependency that triggers a flow.

1. To manually complete a dependency within a flow, open the flow in the Flow Manager. To manually complete a dependency that triggers a flow, from the View menu, select Global View to display all of the flows.
2. Right-click on the dependency line representing the dependency you want to complete.
3. Select Complete Dependency. The dependency is completed, removing the dependency only for this occurrence of the flow. Completing a dependency has no impact on the flow definition.

Note:

Removing a dependency does not automatically make a work item eligible to run—if it has other dependencies, it will wait for those to be met, unless you complete them also.

Kill a running job

You can kill an individual job that is running, without killing the entire flow.

From the Flow Manager

1. In the Flow Manager, locate the flow containing the job you want to kill, and open the flow.
2. Locate the job you want to kill, and right click on it.
3. From the menu, select Kill.

From the command line

1. On the command line, specify the following:

```
jjob -i flow_id -k flow_name[:subflow_name]:job_name
```

where *flow_ID* is the unique flow ID containing the job you want to kill, *flow_name* is the name of your flow, *subflow_name* is the name of your subflow (if you have one), and *job_name* is the name of your job.

2. Press Enter.

Stop a flow at a specific point by putting a job on hold

In some cases, you may want to stop a flow at a specific point so that you can fix problems. You can do this by putting a job in the flow on hold.

When you put a job in the flow on hold:

- The job receives the status On Hold. The status of the flow is not affected.
- The flow pauses at that specific job.

Only the branch of the flow that contains the job that is On Hold pauses. Other branches of the flow continue to run.

Only jobs in the Waiting state can be put on hold. When desired, you can then release the job that you have put on hold. The flow instance continues to run and the job receives the status Waiting.

You can put on hold LSF jobs, job submission scripts, local jobs, job arrays, and job array scripts.

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

Only users who own the flow, the Process Manager administrator, or Process Manager control administrator can hold and release LSF jobs, job submission scripts, local jobs, job arrays, and job array scripts.

1. Select the By State tab, and double-click a flow to display it.
2. Select the desired job in the Waiting state in the flow, right-click and choose Hold.

From the command line

1. On the command line, specify the following for a job in the Waiting state:

```
jjob -i flow_id -p flow_name[:subflow_name]:job_name
```

where *flow_ID* is the unique flow ID containing the job you want to put on hold, *flow_name* is the name of your flow, *subflow_name* is the name of your subflow (if you have one), and *job_name* is the name of your job.

2. Press Enter.

The job will be put on hold.

Note:

To release the job at a later time, use the `-g` option on the command line.

Run or rerun a single job

You can run or rerun a single job directly from the Flow Manager. You may want to use this option to debug a flow, or to run a single job to fix a flow. You can use this option to rerun a job, regardless of whether the job failed or completed successfully.

You can run or rerun a job in a suspended flow, but the state of the flow does not change. If you specify a rerun exception handler to rerun a job, and the flow is suspended, the job does not run until the flow has been resumed.

When you rerun a job in a flow that is running, successor work items run as designed.

When you rerun a job in a flow that is Exited or Killed, only the job runs—its successors do not. If you want to rerun more than one job, you must wait until one job completes before rerunning the next—you rerun them one at a time. If you want to rerun multiple jobs, or if you want the successor jobs to run, you need to rerun the flow.

1. Locate the flow containing the job you want to run, and open the flow.
2. Locate the job you want to run, and right click on it.
3. From the menu, select Run. The job will be run, but its successors may not, depending on the state of the flow.

From the command line

1. On the command line, specify the following:

```
jjob -i flow_id -r flow_name[: subflow_name]: job_name
```

where *flow_ID* is the unique flow ID containing the job you want to run, *flow_name* is the name of your flow, *subflow_name* is the name of your subflow (if you have one), and *job_name* is the name of your job.

2. Press Enter.

Mark a job complete

You can mark a job complete without actually running the job. Use this option when you want a flow to continue running, even though the job failed or did not run. Marking a job complete does not actually run the job—it just changes its state. You mark a job complete so that its successor jobs can run when you rerun the flow. You can only complete a job in a flow that has exited.

1. Locate the flow containing the job you want to mark complete, and open the flow.
2. Locate the job you want to mark complete, and right click on it.
3. From the menu, select Complete Job.

From the command line

1. On the command line, specify the following:

```
jjob -i flow_ID -c flow_name[:subflow_name]:job_name
```

where *flow_ID* is the unique flow ID containing the job you want to mark complete, *flow_name* is the name of your flow, *.subflow_name* is the name of your subflow (if you have one) and *job_name* is the name of your job.

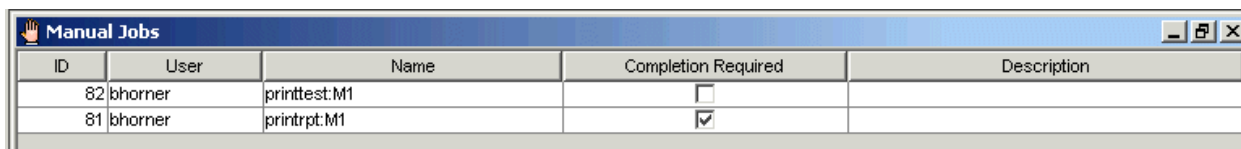
2. Press Enter.

Work with manual jobs

A flow containing a manual job cannot complete its processing until the manual job has been explicitly completed. When a flow progresses to the point where the manual job is next in the work flow, that branch of the flow (or the entire flow) halts. A notification is sent to a specified email address, indicating that the manual job is awaiting completion. Generally, this requires completing the actual task associated with the manual job and then completing the manual job to indicate that the task is complete. You can complete a manual job using the Flow Manager or using the command line interface.

View the manual jobs awaiting for completion

1. In the Flow Manager, from the View menu, select Manual Jobs. The Manual Jobs window appears, listing the manual jobs that are not yet completed.



ID	User	Name	Completion Required	Description
82	bhorner	printtest:M1	<input type="checkbox"/>	
81	bhorner	printrpt:M1	<input checked="" type="checkbox"/>	

Note that not all manual jobs in this list are ready to be completed. Those manual jobs that are awaiting completion have a check mark in the Completion Required column.

From the command line

1. On the command line, type the following:
jmanuals
2. Press Enter.

Complete a manual job

1. In the Flow Manager, from the View menu, select Manual Jobs. The Manual Jobs window appears.
2. Locate the manual job in the list—it will have a check mark in the Completion Required column.
3. Ensure that the manual task associated with this manual job has been completed, and complete the manual job. Left-click or right-click on the job to select it.
4. Click Complete Manual Job or click the Complete the Manual Job button. The Complete manual job dialog appears.
5. If applicable, in the Description field, specify any comments required to describe what happened. For example:



Complete manual job

Name: testflow:M1

Description: Completed by J. Doe.

Complete Manual Job Cancel

The description you enter here appears in the Runtime Attributes of the manual job.

6. Click Complete Manual Job.

From the command line

1. On the command line, type the following:

```
jcomplete -i flow_id flow_name[:subflow_name]:job_name
```

where *flow_id* is the unique ID of the flow containing the manual job, and *flow_name* [:*subflow_name*]:*job_name* is the fully qualified name of the manual job to complete.

2. Press Enter.

Completing manual jobs with exit codes

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

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

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

Complete a manual job with an exit code

1. In Flow Manager, select the manual job, right-click and select Complete Manual Job.
2. In the Exit code field, enter the exit code for the job.
3. Click the Complete Manual Job button.

You can view the exit code you entered by selecting the manual job, right-clicking and choosing View Runtime Attributes.

From the command-line

Use the `j complete` command with the `-e` option.

Example: Complete a manual job with exit code 4

```
jcomplete -d "printed check numbers 4002 to 4532" -e 4 -i 42 payprt:checkprinter
```

completes the manual job `checkprinter` with exit code 4 in the flow `payprt` with flow ID 42, and adds the comment "printed check numbers 4002 to 4532".

Work with proxies

Proxies are used to represent work items that run within another flow, or to represent another flow. Another work item can depend on the success or failure of a proxy. A proxy event can be used to trigger a flow, or to trigger a work item within a flow.

Using the Flow Manager, you can do the following with proxies:

- See if any proxies of a work item exist.
- If proxies to a work item exist, see a list of those work items that depend on them. This allows you to determine the impact that a work item has on other flows.
- Locate a proxy dependant
- Manually complete a proxy dependency
- View the inter-flow relationships established by defining proxies, using the global view.

Flow ID	Work Item	Owner	Condition
	flowB:A1	bhorner	Completes successfully
	flowB:Hello_World	bhorner	Completes successfully

Proxy dependants—those work items that depend on a proxy— are listed in the Show Proxy Dependants dialog. The list of proxy dependants includes every location where dependants to the proxy are defined, including both flow definitions and flow instances. If no flow ID is listed in the Flow ID column, the item listed is a flow definition.

The Show Proxy Dependants dialog shows the flow ID where the dependant runs, if applicable, the name of the dependant, including flow and subflow names, if applicable, the owner of the flow and the condition under which the dependant runs.

If you double-click on a work item listed in the Show Proxy Dependants dialog, the flow definition or flow containing that work item is opened. However, you cannot change the definition here—you need to change it in the Flow Editor and resubmit it.

To see if any proxies of a work item in a flow exist:

1. Open the flow containing the work item.
2. Right-click on the work item for which you want to check for proxies.
3. Select Show Proxy Dependants. The Show Proxy Dependants dialog appears. If any proxies to this work item exist, the work items that depend on them are listed here.

See if any proxies of a flow exist

1. In the tree view, right-click on the flow definition name.
2. Select Show Proxy Dependants. The Show Proxy Dependants dialog appears. If any proxies to this flow exist, the work items that depend on them are listed here.

Navigate to a proxy dependant

1. Open the flow containing the work item.
2. Right-click on the work item for which you want to check for proxies.
3. Select Show Proxy Dependants. The Show Proxy Dependants dialog appears. If any proxies to this work item exist, the work items that depend on them are listed here.
4. In the list of proxy dependants, double-click on the work item you want to locate. The flow or flow definition containing the work item is opened in the right-hand pane.

Manually complete a proxy dependency

1. Open the flow containing the work item.
2. Right-click on the work item for which you want to check for proxies.
3. Select Show Proxy Dependants. The Show Proxy Dependants dialog appears. If any proxies to this work item exist, the work items that depend on them are listed here.
4. In the list of proxy dependants, double-click on the work item you want to locate. The flow containing the work item is opened in the right-hand pane.
5. Right-click on the dependency line running from the proxy to the proxy dependant.
6. Select Complete Dependency. The dependency is completed, removing the dependency only for this occurrence of the flow. Completing a dependency has no impact on the flow definition.

Note:

Removing a dependency does not automatically make a work item eligible to run—if it has other dependencies, it will wait for those to be met, unless you complete them also.

Kill a running flow

You can kill a flow any time after it has started running. Killing a flow kills any work items within the flow that have not yet completed.

1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow you want to kill.
3. Right-click on the flow and select Kill. All incomplete or waiting jobs in the flow are killed.

From the command line

1. On the command line, type the following:

```
jkil flow_id
```

where *flow_id* is the unique ID of the flow you want to kill.

2. Press Enter.

Suspend a running flow

You can suspend a flow after it has started running. Suspending a flow suspends all jobs, job arrays and subflows within the flow that have not yet completed. Any jobs that were already completed before the flow was suspended are not affected by either suspending or resuming the flow.

1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow you want to suspend.
3. Right-click on the flow and select Suspend. All incomplete and waiting jobs in the flow are suspended until they are explicitly resumed.

From the command line

1. On the command line, specify the following:

```
jstop flow_id
```

where *flow_id* is the unique ID of the flow you want to suspend.

2. Press Enter.

Resume a suspended flow

You can resume a flow after it has been suspended. Resuming a flow resumes all suspended jobs, job arrays and subflows within the flow. Any jobs that were already completed before the flow was suspended are not affected by either suspending or resuming the flow.

1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow you want to resume.
3. Right-click on the flow and select Resume. All suspended jobs in the flow are now resumed.

From the command line

1. On the command line, specify the following:

```
jresume flow_id
```

where *flow_id* is the unique ID of the flow you want to resume.

2. Press Enter.

Rerun an exited flow

You can rerun a flow that has exited, provided that the flow was not killed.

When you rerun a flow, jobs are rerun as follows:

- If the flow uses the default completion criteria (the flow exits when a job exits), the flow runs again, beginning with the job that exited. Only exited jobs are rerun.
- If the flow uses completion criteria (the flow is complete when one or more specified jobs in the flow complete), the flow runs again, beginning with the jobs that exited and any jobs set as rerun starting points, but all successor jobs are also rerun, even if they are Done.

If you need to rerun a flow that was killed, retrigger the flow.

Note:

Rerunning a flow that contains an alarm will not reopen a previously opened alarm. Similarly, rerunning a flow that contains a manual job that was already marked complete will not reset the state of the manual job. If the flow contains a manual job that was already marked complete, the state of the manual job is reset to waiting, but the manual job will not require completion again—the remainder of the flow may not run as designed.

1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow you want to rerun.
3. Ensure that no jobs are still running within the flow—sometimes elements of a job array or jobs in a subflow may continue to run after a flow exits.
4. Right-click on the flow and select Rerun.

The Rerun Flow dialog is displayed.

5. Select whether to rerun the flow from exited items and starting points, or from starting points only and click OK.

The flow is rerun, beginning with any jobs that exited, were killed, or were set as rerun starting points.

From the command line

1. On the command line, specify the following:

```
jrerun flow_id
```

where *flow_id* is the unique ID of the flow you want to rerun.

2. Press Enter.

Set a starting point to rerun a flow

By default, when you rerun a flow, the flow continues from exited jobs. Under certain situations, the root cause of a job failing may be from conditions set by a previous job, in which case, you will need to rerun the flow from a job that is before the exited job. To address this situation, you can use the flow manager to 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.

You can only set a rerun starting point for flows that are in an Exited state. In order to be set as a rerun starting point, the item in the flow must meet the following requirements:

- The item must be in a Done or Exited state.
- The item must be one of the following types of work items:
 - Job
 - Job array
 - Job submission script
 - Job array submission script
 - Local job
 - Template job

Tip:

You can set multiple work items as rerun starting points. In addition, all exited jobs in a flow automatically become rerun starting points. This is the default behavior and remains unchanged even when you set other work items as rerun starting points.

1. In the Flow Manager, select the most appropriate view for finding the points to rerun the flow.
2. Right-click the work item and select Set as starting point to rerun flow.

When you set a work item to be the starting point to rerun a flow, the work item will have a green circle in the top-right corner to indicate that it is the rerun starting point.

Remove the starting point to rerun a flow

To remove a work item as a rerun starting point, use the flow manager to unset a work item as the rerun starting flow.

1. In the Flow Manager, select the appropriate work item to find the points to rerun the flow.
2. Right-click the work item and select Unset as starting point to rerun flow.

Rerun a flow while a job is still running

You can rerun a flow that has the Running, Exited, or Done state.

This is useful for flows that have several branches. When one branch fails, you can rerun the branch without waiting for other branches of the flow to complete.

You can:

- Set or unset starting points when there are still jobs running in the flow.
- Choose whether to rerun the flow from:
 - Exited items and starting points. The flow will rerun from any starting points, exited work items, and, from the item following any manually completed jobs provided dependencies are met.
 - Starting points only. The flow will rerun only from starting points.

Note that you can only rerun a running flow if the part of the flow to be rerun does not overlap with items that are currently running.

1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow you want to rerun.
3. Right-click on the flow and select Rerun.

The Rerun Flow dialog is displayed.

4. Select whether to rerun the flow from exited items and starting points, or from starting points only and click OK.

The flow is rerun, beginning with any jobs that exited, were killed, or were set as rerun starting points.

Rerun an exited job array

You can rerun a job array that has exited. You can rerun the entire job array, or only those elements of the array that exited. When you rerun a job array, the job array has a new ID.

Note:

Rerunning a job array that triggers an alarm will not reopen a previously opened alarm.

1. In the Flow Manager, locate the job array you want to rerun.
2. Right-click on the job array and select Rerun.

Hold a flow definition

You can hold a flow definition that has been submitted to the Platform Process Manager system. You do this when it has been scheduled to trigger automatically, but you do not want that automatic trigger to happen for some period of time. For example, you may do this when you first submit the flow definition but are not quite ready to put it into production, or when you require a maintenance window. The flow definition remains on hold until it is explicitly released.

When a flow definition is on hold, it cannot be triggered automatically, but can still be triggered manually.

1. In the Flow Manager, select By Definition.
2. Expand the tree view until you see the flow definition you want to hold.
3. Right-click on the flow definition and select Hold. The status of the flow definition changes to On Hold.

From the command line

1. On the command line, type the following:

```
jhold flow_definition_name
```

where *flow_definition_name* is the name of the flow definition you want to place on hold.

2. Press Enter.

Releasing a flow definition from hold

When a flow definition is placed on hold, it cannot be triggered automatically until it has been explicitly released.

1. In the Flow Manager, select By Definition.
2. Expand the tree view until you see the flow you want to release.
3. Right-click on the flow definition and select Release. The status of the flow definition changes to Released.

From the command line

1. On the command line, type the following:

```
jrelease flow_name
```

where *flow_name* is the name of the flow definition you want to release.

2. Press Enter.

Remove a flow definition

When you no longer require a flow definition, you can remove it from the list of flows the Platform Process Manager system knows about. If you remove a flow definition, and some flows belonging to the flow definition are still in the Platform Process Manager system, they appear in the Flow Manager in the adhoc folder.

1. In the Flow Manager, select By Definition.
2. In the tree view, locate the flow definition you want to remove.
3. Right-click on the flow definition and select Remove.
4. Confirm that you want to remove this definition. The flow definition is removed from the system.

From the command line

1. On the command line, type the following:
`jremove flow_name`
where *flow_name* is the name of the flow definition you want to remove.
2. Press Enter.

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.

Using mainframe

To use the mainframe support, you must:

1. Copy the template file `z/OS_Templ ate. xml` from `JS_TOP/8.1/examples` to `JS_TOP/work/templates`.
2. Define your template job in Flow Editor.

Define your job

Use the template job feature to define your mainframe job.

1. Make sure you have copied the `zOS_Templ ate. xml` file from `JS_TOP/8.1/examp les` to `JS_TOP/work/templ ates`.
2. Select the Insert Application button from the design palette.

The Insert Application window displays.

3. Select zOS Job from the list and click OK.
4. Click anywhere on your flow page.

A zOS job is added to your flow.

5. Right-click your zOS job and select Open Definition.

The Application Definition window displays:

Application Definition - Edit Application

General | Execution Environment | Exception Handling | About

Identification

Name: J5

Description:

Application Parameters

Parameter Name	Parameter Value
z/OS host name*	devsol01.lsf.platform.com
Login user ID*	iheys
Password*	*****
Is your JCL file located on z/OS host?*	Yes, my JCL file is located on z/OS host
JCL file*	./usr/jobs/entry.jcl
Output file*	./usr/jobs/entry.jcl.out
Estimated run time (in minutes)	1
Check interval (in minutes)	1
Time out (in minutes)	0

OK Cancel

On the **General** tab

Field	Description
z/OS host name	The full host name where the mainframe job is submitted to.
Login User ID	The mainframe log in ID.
Password	The mainframe log in password.
Is JCL file located on z/OS host?	Location of the JCL file (either on the z/OS host or LSF execution host).
JCL File	Full path to the JCL file to submit with the job.
Output file	Full path to the file to receive the mainframe job output. Note: Any existing output file will be overwritten without a warning.
Estimated run time (in minutes)	(Optional) The estimated run time of the job. This value informs the system when to begin checking the job status. Specifying this value reduces system overhead for long jobs.
Check interval (in minutes)	(Optional) How often the status of the job is checked by the system.

Field	Description
Time out (in minutes)	(Optional) The number of minutes before the job times out. If the job times out, the job exits with exit status 237. A time out period of 0 means no time out (the job runs until it finishes).

On the Execution Environment tab

Field	Description
Submit to queue/ partition	(Required for Windows only) Specify a queue created by the Administrator to be able to kill the job if necessary. Contact your Administrator for the mainframe queue name.
Run on host	(Optional) Specify the LSF host name where the proxy job will run.
Run as user	(Optional) Specify an LSF user name.
File Transfer	(Optional) Specify file transfers between Platform Process Manager Server and LSF execution host.
Log File	(Optional) Full path to the log file that contains the stdout of the LSF job. Includes FTP messages. Use for troubleshooting.

Status of jobs

The status of your mainframe jobs is displayed in Flow Manager just like any other job.

Killing a job (Windows)

To kill a job in a Windows environment, the Administrator must create a queue specifically for mainframe jobs. For jobs to be eligible to be killed, you must submit the mainframe job to that special queue. Contact your Administrator for more information.

Killing a job (UNIX)

You can kill a mainframe job regularly if you are on a UNIX platform.

Exit codes

The following exit codes may occur if there is a problem between your LSF proxy job and the mainframe job:

Exit Code	Failure Reason
230	Failed to connect to mainframe via FTP.
231	Failed to log in to mainframe.
232	Command <i>site filetype=jes</i> or <i>site filetype=seq</i> failed.
233	Failed to retrieve job ID. <i>put</i> or <i>get</i> command failed.
234	Failed to retrieve job output.
235	Failed to match Dir Header. <i>Dir</i> command failed.
236	Failed to get job status. <i>Dir</i> command failed.
237	Timeout checking mainframe job status.
238	Failed to delete a mainframe job.
239	Encryption error.
240	Environment variable not found.
241	Script error.
242	Platform Process Manager configuration file not found.
243	FTP configuration file not found.
244	Incomplete system output file: IEF142I and IEF472I not found.
245	System output file not found.
246	<i>bpost</i> command failed.
247	<i>bread</i> command failed.
248	Failed to get mainframe job ID from <i>bread</i> output.
249	Failed to transfer JCL file from z/OS host to LSF host.
250	Failed to modify job name in JCL file.
255	Mainframe job has an ABEND status.

Mainframe support

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:

- `cal editor`—to start the Calendar Editor graphical user interface
- `j cadd`—to create a calendar
- `j cal s`—to display a list of calendars
- `j cdel`—to delete a calendar
- `j cmod`—to edit a calendar

Flow definition commands

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

- `fl oweditor`—to start the Flow Editor graphical user interface
- `j run`—to submit and run a flow immediately, without storing the flow definition in Platform Process Manager
- `j sub`—to submit a flow definition to Platform Process Manager
- `j trigger`—to trigger the creation of a flow
- `j hold`—to place a flow definition on hold, preventing automatic triggering of the flow
- `j release`—to release a flow definition from hold, enabling automatic triggering of the flow
- `j defs`—to display information about flow definitions
- `j remove`—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
- `jobs`—to list open alarms
- `complete`—to complete a manual job
- `flows`—to display information about a flow
- `job`—to kill or run a job, or to mark a job complete
- `kill`—to kill a flow
- `manuals`—to list all manual jobs waiting for completion
- `publish`—to publish target flows for use by dynamic flows and flow arrays
- `rerun`—to rerun an exited flow
- `resume`—to resume a suspended flow
- `setvars`—to change the value of a local or global variable while a flow is running
- `stop`—to suspend a flow
- `unpublish`—to unpublish target flows and remove them from the list for use by dynamic flows and flow arrays

Other commands

- `id`—to verify the connection between the Platform Process Manager Client and the Platform Process Manager Server
- `admin`—to control the Platform Process Manager daemon on Unix
- `hist`—to view the historic information about server, flow definitions, flows, and jobs.
- `reloadalarm`—to reload the alarm definitions.

caleditor

starts the Calendar Editor.

Synopsis

caleditor

You use the `cal edit or` 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 as follows:

start_time,end_time|start_time|,end_time|start_time

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

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

Where:

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

start_time,end_time

Specifies both the start and end times of the interval.

start_time,

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

,end_time

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

start_time

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

Absolute time examples

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

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

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

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

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

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

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

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

Commands

`..` = 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 `j cadd` or `j cmod` 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 | | Fri days@Sys && !Hol i days@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_Hol i day`. This calendar resolves to any Monday that is not a holiday, as defined in the `Hol i days` system calendar.

```
jcadd -d "Mondays, Wednesdays and Fridays" -t "Mondays@Sys || Wednesdays@Sys || Fridays@Sys" Everyot herday
```

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

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

Creates a calendar called `Short week` 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`

jcal s

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

Synopsis

```
jcal s [-l] [-u user_name|-u all] [cal_name]
```

```
jcal s [-h][[-V]
```

Description

You use the `j cal s` 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

```
jcal s -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" Runday2001
```

Deletes the calendar Runday2001 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 | | Fri days@Sys && !Hol i days@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 Special Days. 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] [-e exit_code]-i flow_id flow_name  
[:subflow_name]:manual_job_name
```

```
jcomplete [-h][[-V]
```

Description

You use the `jcomplete` command to mark a manual job complete, to tell 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.

-e *exit_code*

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

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

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

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

-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 `jmanual s` 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 `j defs` 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.
<i>flow_id</i>	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 <code>stderr</code> and exits.
-v	Prints the Platform Process Manager release version to <code>stderr</code> 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 `hist` or the `-u root` option is used.

-C *flowdef*

Displays information about when a flow definition state is:

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

-C flow

Displays information about when a flow state is:

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

-C job

Displays information about when a job or job array is:

- Started
- Killed
- Suspended
- Resumed
- Finished

Time interval format

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

Specify the time interval as follows:

start_time,end_time|start_time|,end_time|start_time

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

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

Where:

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

start_time,end_time

Specifies both the start and end times of the interval.

start_time,

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

,end_time

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

start_timeStarts 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,.
```

Commands

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 | -p | -g | -l flow_name[:subflow_name]:job_name
```

Flow arrays in UNIX:

```
jjob [-u user_name] -i flow_id -c | -k | -r | -p | -g | -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.

-p

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

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

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

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

-g

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

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

-l

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

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

Kill a specific flow

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

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

Hold and release a job

- Hold a job

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


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

- Release the job

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

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

Hold and release a job array

- Hold a job array

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

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

- Release the job array

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

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

Hold and release a job in a flow array

- Hold a job in a flow array

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

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

- Release the job in the flow array

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

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

- Hold all jobs in all elements in the flow array

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

- Release all jobs in all elements in the flow array

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

See Also

[jmanu](#) [als](#)

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

```
kill -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

`j unpubl i sh`

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 `j r e l e a s e` 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

`j h o l d`

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 `j remove` 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 `hold` 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

`j sub`, `j hold`

jrerun

reruns an exited, done, or running flow.

Synopsis

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

```
jrerun [-h][|-V]
```

Description

You use the `jrerun` command to rerun a flow. The flow must have a state of Exit, Done, or Running.

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

If the flow contains multiple branches, the flow is rerun from the first exited jobs or starting points in each branch and continues to process as designed.

You must be the owner of a flow or a 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 `j resume` 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
```

Commands

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

[j stop](#)

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 `j run` 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 `j setvars` 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 `j set versi on` command to set the default version of the specified flow. The default version of the flow is the version set to be effective at the current time. If you trigger this flow, Process Manager will instantiate the flow instance with the default version.

Options

-v *default_version*

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

-u *user_name*

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

flow_name

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

-h

Prints the command usage to `stderr` and exits.

-V

Prints the 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.errfile` 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 `j sub` 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 `j sub` 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. `j sub` 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]][#occurrences][+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, *.dat* matches abc. dat, another. dat and abc. dat 23. S??day* matches Sat days. tar and Sundays. dat. *e matches smi l e.

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 ==.

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

Note:

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

- Example: on successful completion of J1:
-p "job(done(jdoe:myflow:J1))"
- Example: if payjob exits with an exit code greater than 5:
-p "job(exit(jdoe:myflow:testflow:payjob)>5)"
- Example: if payjob ends with any of the following exit codes: 5, 10, 12, or 14:
-p "job(exit(jdoe:myflow:testflow:payjob)==5 10 12 14)"
- Example: if payjob does NOT end with any of the following exit codes: 7, 9, 11:
-p "job(exit(jdoe:myflow:testflow:payjob)!=7 9 11)"

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

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
```

Commands

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
```

Commands

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

`jr run`

junpublish

unpublishes a target flow from Platform Process Manager.

Synopsis

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

```
junpublish [-h][[-V]]
```

Description

You use the `junpublish` command to unpublish a target flow from Platform Process Manager. Unpublished 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

`junpublish`

