

IBM Spectrum LSF Process Manager  
Version 10 Release 2

*Using the IBM Spectrum LSF Process  
Manager Clients*



**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 207.](#)

This edition applies to version 10, release 2 of IBM Spectrum LSF Process Manager (product number 5725G82) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 1992, 2018.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Chapter 1. Introduction to Process Manager.....</b>	<b>1</b>
About Process Manager.....	1
About Process Manager terms.....	2
Change your server.....	7
About flow definitions and flows.....	8
Actions you can perform against flow definitions.....	8
What can I do with a flow definition?.....	9
What can I do with a flow?.....	9
What can I do with a job?.....	9
Where do I store my flow definitions?.....	10
What makes a flow Done?.....	10
What happens if a job exits?.....	10
How does Process Manager know when my flow is complete?.....	10
Flow definition examples.....	10
<b>Chapter 2. Process Manager Calendars.....</b>	<b>11</b>
About the calendar editor.....	12
Creating a calendar with specific dates.....	13
Create a calendar using an expression.....	14
Create a calendar with a complex expression.....	16
Calendar examples.....	18
Edit an existing calendar.....	19
Edit a calendar to change the pattern.....	19
Edit a calendar to change calendar combinations.....	20
Delete a calendar.....	20
Updating time zone data.....	20
<b>Chapter 3. Define your flow.....</b>	<b>25</b>
Ways to create a flow definition.....	25
How do I know if a job or job array is undefined?.....	26
Using the example flows.....	26
View the sample flow definition.....	27
Use a sample flow definition.....	27
Create a flow diagram.....	27
Create a simple flow diagram.....	27
Other things you can do.....	28
Include a job array in the flow diagram.....	28
Insert a job array.....	28
Define job array details.....	29
Preparing job array input files.....	30
Include a job submission script in the flow diagram.....	30
Define job details.....	31
Include a job array submission script in the flow diagram.....	31
Define job array details.....	31
Content of the job/job array submission script.....	32
Include a static subflow in the flow diagram.....	32
Include a static flow array in the flow diagram.....	33
Flow array element names.....	34
Viewing a static flow array.....	34
Include a dynamic subflow in the flow diagram.....	35

Include a dynamic flow array in the flow diagram.....	35
Dynamic flow array element names.....	36
Viewing a dynamic flow array.....	37
Include a manual job in the flow diagram.....	37
Define manual job details.....	37
Specifying custom exit codes for successful job completion.....	38
From Flow Editor.....	38
Include a local job in the flow diagram.....	39
Define local job details.....	40
Running a local job.....	40
Insert a job to another batch system.....	41
About Other Batch Jobs.....	41
Include a job for another Batch System in the flow.....	42
Output and error file generation for work items in a flow.....	42
Default location of output and error files.....	43
Override order for output and error file generation settings.....	43
Configuring output and error file generation for work items in a flow.....	43
About variables in Process Manager.....	44
Types of variables .....	45
Scope of variables and variable override order.....	45
List of Process Manager built-in variables.....	47
Where you can use variables.....	52
Include the variable evaluator to run jobs based on decision branches.....	54
Use global variables.....	55
Setting user variables within a flow definition.....	56
Set a flow variable using the flow manager.....	62
Dependencies.....	62
Job dependencies.....	62
Specify dependency on the start or submission of specific jobs.....	64
Running a job in a flow based on file activity.....	67
Running a flow based on file activity.....	68
Change the label displayed for an event.....	69
Dependency on a date and time.....	69
Specify dependencies on a job array.....	71
Define dependencies between elements in job arrays .....	74
Specify dependencies on a subflow.....	75
Specify dependencies on an unconnected work item.....	77
Specifying multiple dependencies.....	80
Details of a job.....	81
The General tab.....	81
The Submit tab.....	83
The Processing tab.....	85
The Resources tab.....	86
The Limits tab.....	87
The File Transfer tab.....	88
The Advanced tab.....	89
The Exception Handling tab.....	91
The Description tab.....	91
Determine the flow state with Flow Completion Attributes.....	92
About completion attributes.....	92
Defining when a flow is successful or failed when using branches in a flow.....	93
Specify flow completion attributes.....	94
Configuring flow exit codes.....	97
Configure flow exit code calculation.....	98
Configure dependencies for subflows.....	98
Specify exception handling for a flow.....	98
Flow attributes.....	99
Specify flow attributes.....	101

Turn off email notification for a flow.....	102
Save the flow definition.....	102
Loop a flow or subflow.....	102
Loop a subflow that does not contain a loop definition.....	103
<b>Chapter 4. About Process Manager exceptions.....</b>	<b>105</b>
About exception handling.....	108
Process Manager built-in exception handlers.....	108
User-defined exception handlers.....	108
Behavior when exception handlers are used.....	109
Handling exceptions.....	112
Handle exceptions of a job or job array using built-in handlers.....	112
Handle exceptions of a subflow using built-in handlers.....	113
Handle exceptions with a recovery job.....	114
Handle exceptions with a recovery flow.....	114
Alarms.....	115
Raise an alarm when an exception occurs within a flow.....	115
View the opened alarms.....	116
Insert an alarm in a flow definition.....	116
Use an alarm as an exception handler.....	117
<b>Chapter 5. Run your flow.....</b>	<b>119</b>
Create a flow definition to be triggered manually.....	120
In the flow definition.....	120
From the command line.....	121
Schedule your flow.....	121
Run a flow at a specific time.....	121
Run a flow at multiple times on a single date.....	122
Specifying time expressions.....	122
Run a flow based on file activity.....	123
Run a flow when another flow.....	125
Run a flow when another flow completes.....	125
Run a flow when a proxy job completes.....	126
Run your flow once.....	127
From the Flow Editor.....	127
From the command line.....	127
Submit your flow definition.....	128
Submit your flow without version comments.....	128
Submit your flow with version comments.....	128
<b>Chapter 6. Control a Flow .....</b>	<b>131</b>
About the Flow Manager.....	131
Real-time data.....	133
Refresh the data displayed manually.....	133
Change the automatic refresh interval.....	133
Print data.....	133
Filter the data displayed in the tree view.....	133
Limit the flows displayed to those owned by a user.....	134
Limit the flows displayed to last x hours.....	134
Limit the flows displayed to a time period.....	134
Trigger a flow.....	135
Trigger a flow.....	135
Trigger a flow, passing it values for variables.....	135
View a flow definition and specify versioning options.....	136
View Version.....	136
View Statistics.....	137
View inter-flow relationships.....	137

View proxy dependencies that trigger flows.....	138
View proxy dependencies within a flow.....	138
Manually complete an inter-flow dependency.....	138
View dependencies on flows that do not exist or are on hold.....	139
Determine the status of jobs in a flow.....	139
Manually complete a dependency.....	141
Kill a running job.....	141
From the Flow Manager.....	141
Run or rerun a single job.....	142
From the command line.....	142
Stop a flow at a specific point by putting a job on hold.....	142
From the command line.....	143
Mark a job complete.....	143
From the command line.....	143
Work with manual jobs.....	144
View the manual jobs awaiting for completion.....	144
Complete a manual job.....	144
Completing manual jobs with exit codes.....	145
Complete a manual job with an exit code.....	145
From the command-line .....	145
Work with proxies.....	145
See if any proxies of a flow exist.....	146
Navigate to a proxy dependant.....	146
Manually complete a proxy dependency.....	147
Kill a running flow.....	147
From the command line.....	147
Suspend a running flow.....	147
From the command line.....	148
Resume a suspended flow.....	148
From the command line.....	148
Rerun an exited flow.....	148
From the command line.....	149
Set a starting point to rerun a flow.....	149
Rerun a flow while a job is still running.....	150
Rerun an exited job array.....	150
Hold a flow definition.....	151
From the command line.....	151
Releasing a flow definition from hold.....	151
From the command line.....	152
View a flow definition and specify versioning options.....	152
View Version.....	152
View Statistics.....	153
Remove a flow definition.....	153
From the command line.....	153
<b>Chapter 7. Mainframe support.....</b>	<b>155</b>
Using mainframe.....	155
Exit codes.....	158
<b>Chapter 8. Commands.....</b>	<b>161</b>
caleditor.....	162
floweditor.....	162
flowmanager.....	163
jadmin.....	163
jalarms.....	164
jcadd.....	166
jcals.....	170

jcdel.....	171
jcmmod.....	172
jcommit.....	176
jcomplete.....	180
jdefs.....	181
jexport.....	183
jflows.....	184
jhold.....	185
jid.....	186
jjob.....	186
jkill.....	189
jlicenseupdate.....	190
jmanuals.....	190
jpublish.....	191
jreconfigalarm.....	192
jrelease.....	192
jremove.....	193
jrerun.....	194
jresume.....	195
jrun.....	196
jsetvars.....	197
jsetversion.....	199
jsinstall.....	199
jstop.....	200
jsub.....	201
jsubmit.....	201
jtrigger.....	201
junpublish.....	203
licenseinfo.....	204
ppmsetvar.....	205
<b>Notices.....</b>	<b>207</b>
Trademarks.....	208





---

# Chapter 1. Introduction to Process Manager

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

## About Process Manager

---

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

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

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

### Tip:

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

### 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 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 Process Manager. If the flow definition is to be triggered by an event, 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 Process Manager until you trigger it, creating a flow. If the flow definition is run immediately, 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 **IBM > IBM Spectrum LSF Process Manager > Flow Editor**, or by running **floweditor** on UNIX.

### The Calendar Editor

You use the Calendar Editor to define calendars, which 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.

Process Manager calendars are independent of jobs, flow definitions and flows, so that they can be reused. The Process Manager administrator can create calendars that can be used by any user of Process

Manager. These are referred to as *system* calendars. Process Manager includes a number of built-in system calendars so you do not need to define some of the more commonly used expressions.

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 **IBM> IBM Spectrum LSF Process Manager > > Calendar Editor**, or by running **caleditor** 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 **IBM> IBM Spectrum LSF Process Manager > > Flow Manager**, or by running **flowmanager** on UNIX.

## About 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. Process Manager schedules and manages jobs that run on Process Manager hosts. 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 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 input1, input2, input3 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. Process Manager schedules and manages job submission scripts that run on the 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 `script1`, `script2`, `script3`, 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.

## Local jobs



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.

## Other Batch jobs

An Other Batch job is a job that you run on a remote batch system or workload manager that is not LSF.

When Process Manager is configured to communicate with another batch system, you will see the Other Batch Job work item enabled in Flow Editor:



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

## 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 embed 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 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

Since version 10.1, adhoc flows have been removed from the By Definition view tab in Flow Manager. However, users can still view adhoc flow status from the By User tab or the By State tab in Flow Manager.

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

- In the **By User** view.
- In the **By Event** view.

**Note:** Since version 10.1, adhoc flows have been removed from the **By Definition** view.

### Static subflows



A *static subflow* is a flow definition that has been copied into another flow definition. Using a static subflow within a flow is a simple method to share and reuse common routines.

### Dynamic subflows



A *dynamic subflow* is a reference to another flow definition. The flow definition that is referenced is called a target flow. The flow that contains the dynamic subflow is called the parent flow.

You maintain the target flow separately from the parent flow. Changes made to the target flow are automatically updated in the parent flow.

Only target flows that have been submitted to Process Manager and published can be selected in a dynamic subflow.

### Static flow array



A *static flow array* is a way to specify to run a static subflow as an array. The subflow will run as many times as the size of the array.

In addition, in the array attributes, you can define whether to run the subflow in parallel or sequentially.

### Dynamic flow array



A *dynamic flow array* is a way to specify to run a dynamic subflow as an array. The subflow will run as many times as the size of the array.

In addition, in the array attributes, you can define whether to run the subflow in parallel or sequentially.

You maintain the target flow separately from the parent flow. Changes made to the target flow are automatically updated in the parent flow.

Only target flows that have been submitted to Process Manager and published can be selected for the dynamic flow array.

## 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. 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 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 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. 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 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 Process Manager to pass variables to and from scripts. 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 Process Manager Server.
- Environment variables, that allow you to submit a job with an environment variable, including a user or local variable.

## Variable evaluator



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.

## File naming conventions

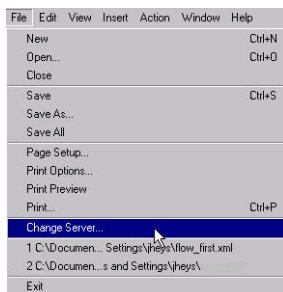
This guide uses UNIX file naming conventions to illustrate file names. However, if the file you are referencing is on a Windows file system, use Windows file naming conventions where applicable.

## 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.

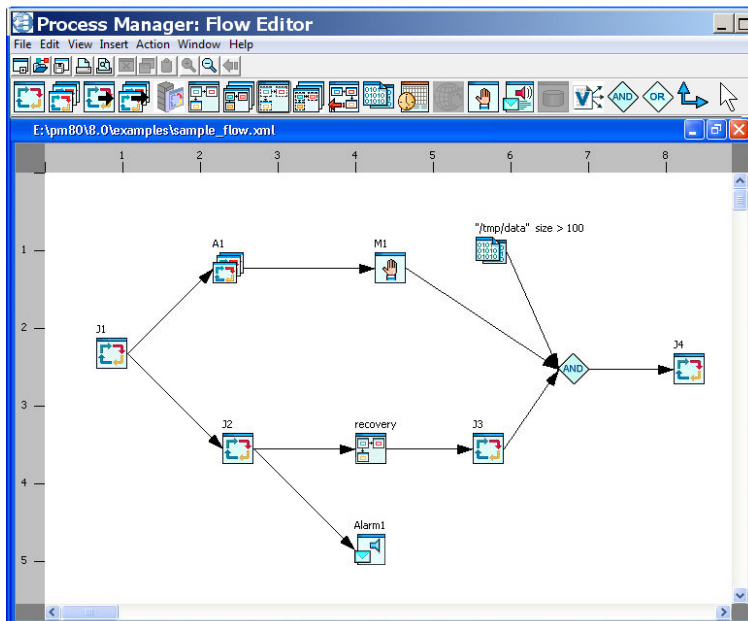
4. Check you server host name and port number in the lower left hand corner of your client application. In Flow Manager, status is also available.



You have changed your server.

## About flow definitions and flows

A flow definition is a collection of Process Manager work items (jobs, job arrays and subflows) and their relationships. These 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 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 Process Manager from running a flow, even though the flow definition has been submitted to Process Manager and is recognized by Process Manager. Holding a flow definition essentially causes 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 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 Process Manager system. After a flow definition is removed, 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 Process Manager system. 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 or a local job
- Hold a waiting job in a running flow

- Rerun a Running, Exit, or Done 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, 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 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. 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 or to ignore work items in the waiting state that never run.

## Flow definition examples

---

To help you build your own flows, you can refer to the examples installed with Flow Editor. You can access the examples through **Help > Flow Examples**:

- `Sample.xml`: Demonstrates a simple flow with three jobs and the dependency **Completes successfully**.
- `jobs_and_dependencies.xml`: Demonstrates a flow with three separate branches that run according to whether the first job in the flow completed successfully or ended with a specific exit code or exit codes.
- `event_and_alarm.xml`: Demonstrates how to use time of day and arrival of a file in a specific directory to trigger a subflow. Also uses an alarm to send an email to the administrator.
- `dynamic_flow.xml` and `target_flow.xml`: Demonstrates how to use a subflow by reference, how to pass input variables from a main flow to a subflow, and how to pass a variable from a subflow to the parent flow and use it in a subsequent job in the main flow.
- `flowarray_eval.xml`: Demonstrates how to use the variable evaluator to decide which branch of a flow to run, how to run flow arrays in parallel or sequentially, and how to use a variable to set the array size. This flow contains two subflows to be run as arrays, and a condition evaluator that decides whether to run the arrays in parallel or sequentially.

---

# Chapter 2. Process Manager Calendars

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.

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 Process Manager calendars using the Calendar Editor.

### About calendars

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

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 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

---

Types of Calendars	Calendar Names
Monthly calendars	First_monday_of_month 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 Process Manager. However, it must be edited by a Process Manager administrator to update the holidays for your company for each year.	

## About the calendar editor

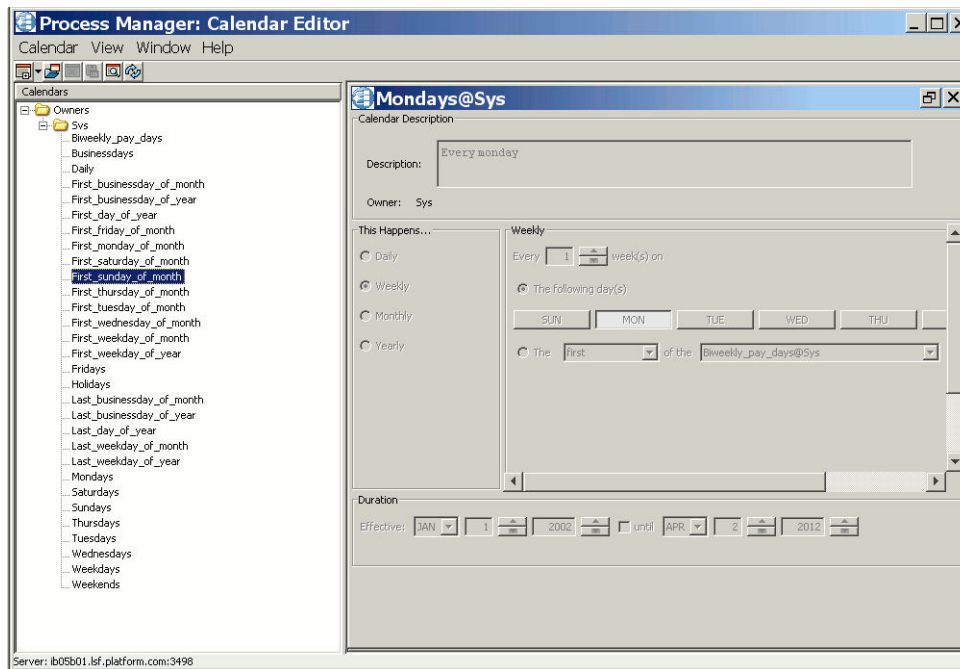
You use the Calendar Editor to create calendars that define the dates on which you want some action to take place. Calendars are required to create time events to trigger flows or dispatch jobs at a particular time. The 6 Server must be running before you can use the Calendar Editor.

### 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:

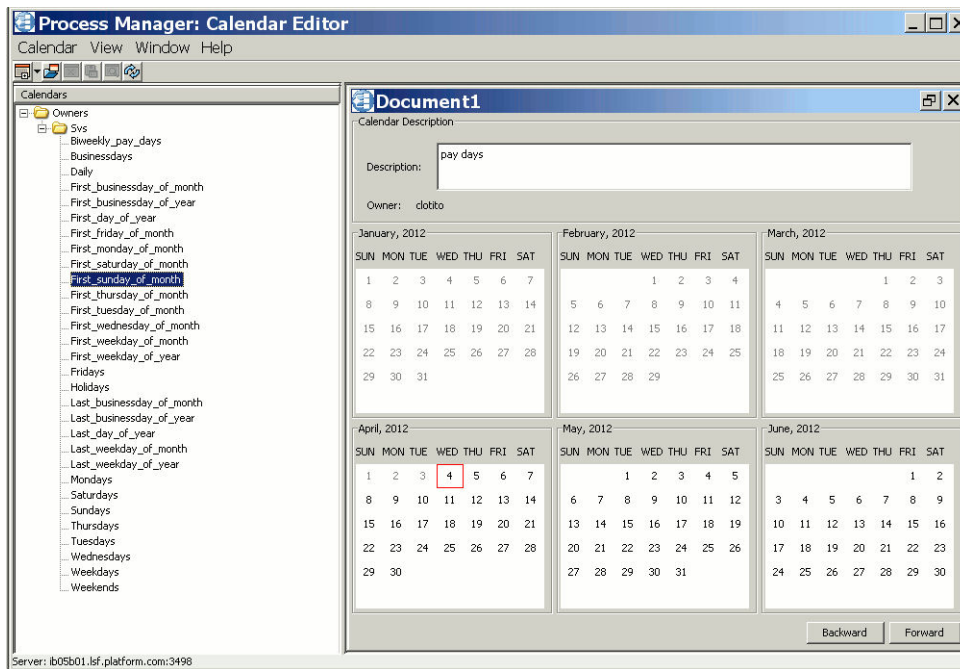
- Calendar names can contain the digits 0 to 9, the characters a to z and A to Z, underscore (\_), and dash (-)
- Calendar names cannot begin with a number

## Creating a calendar with specific dates

You can create a custom calendar with specific dates when the calendar needs to be valid for only one or two dates or the calendar needs to be valid for specific, random dates that do not repeat with any pattern.

### Procedure

1. Ensure that 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.

**Note:** The calendar **Description** field does not support multi-line when creating a new calendar or modifying an existing calendar.

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

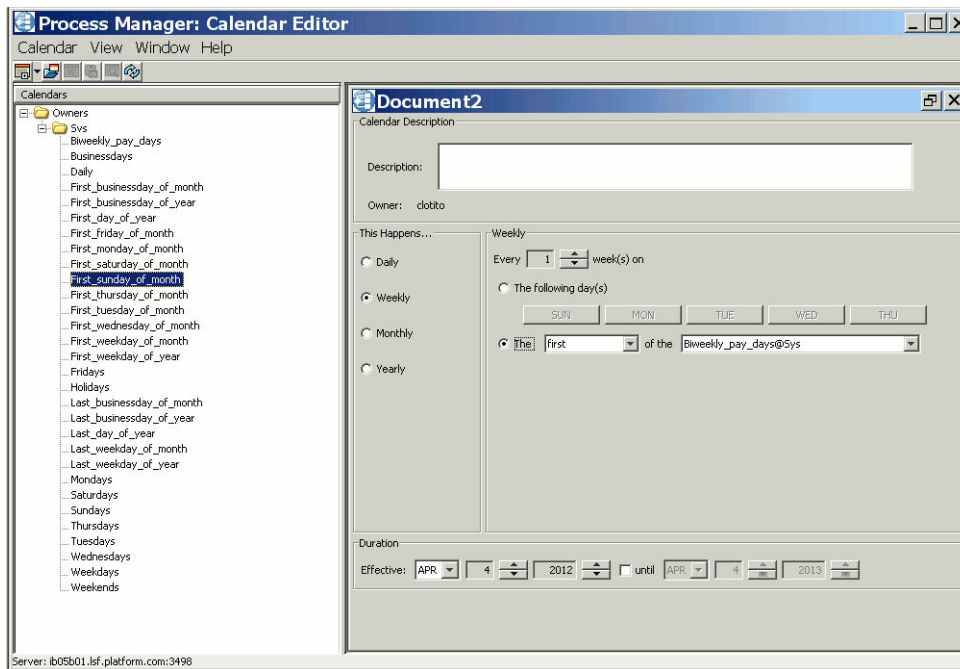
### About this task

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

### Procedure

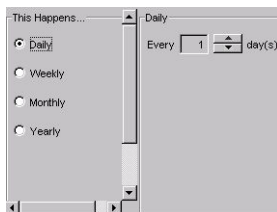
1. Open the Calendar Editor.
2. From the **Calendar** menu, select **New Calendar** and select **Specify Pattern**. The calendar expression dialog box appears. The dialog box is shown here with the list of calendars expanded on the left.



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.



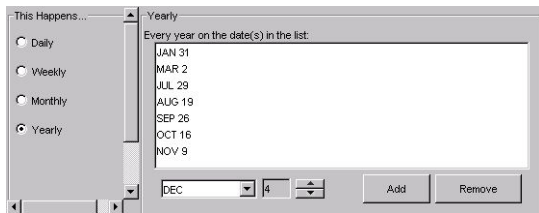
- If the expression should be true on specific days every month or every  $n$  months, in the **This Happens ...** field, select **Monthly**. Then specify how frequently this occurs, in the **Every  $n$  month(s)** field. Then click on the appropriate dates or days of the week. For example, if the expression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

### About this task

Use this method to create a calendar when:

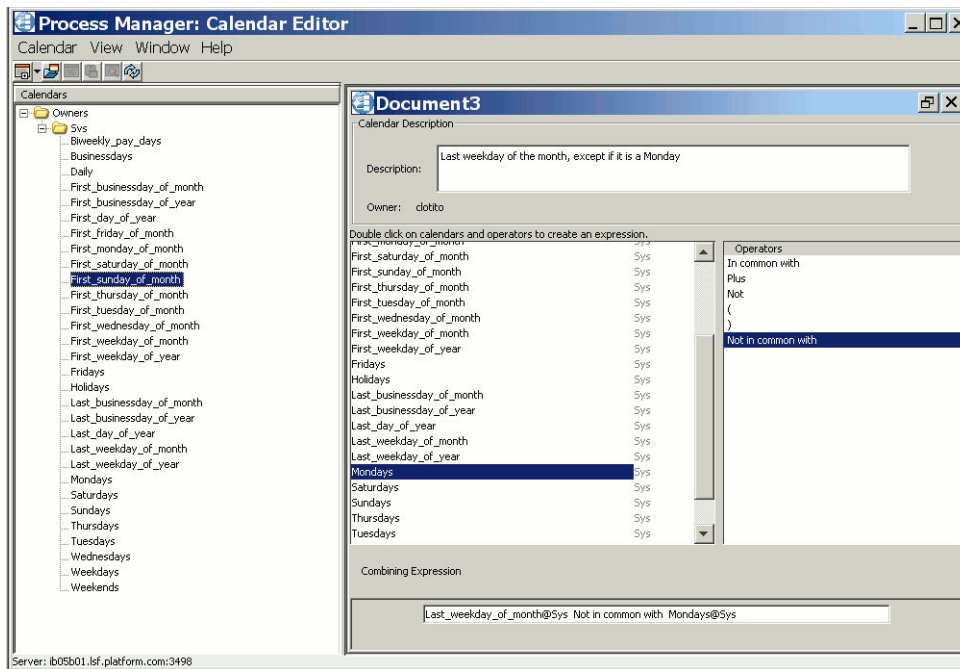
- The calendar needs to be valid on certain days, but must exclude other days, such as holidays
- The calendar needs to be valid on dates that are already defined in one calendar, and also on dates already defined in another calendar

### Procedure

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

Operator	Description
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 the 'This Happens...' dialog box with the 'Weekly' tab selected. Under 'This Happens...', the 'Weekly' radio button is chosen. The 'Every' field is set to '2' and 'week(s) on'. Under 'The following day(s)', the 'FRI' button is selected. The 'The' field is set to 'first' and the 'of the' field is set to '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**

---

### **About this task**

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:

### **Procedure**

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**

### **About this task**

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

### Procedure

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

### About this task

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

### Procedure

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

---

### About this task

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

### Procedure

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**.

## Updating time zone data

---

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

### About this task

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

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

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

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

## Procedure

1. Download the ICU source code.

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

The following packages are required:

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

2. Download the latest ICU data files.

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

For example, go to the directory 2018i/44/1e (where **2018i** is the latest ICU data for the year 2018, **44** is the directory for updates to ICU version 4.4 and newer, and **1e** is the directory for Little Endian processors, including all Intel processors).

The following ICU data files are required:

- zoneinfo64.res
- windowsZones.res
- timeZoneTypes.res
- metaZones.res

3. Build the ICU data library for Windows.

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

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

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

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

- 1) Choose **Build**. Select **Configuration Manager**.

- 2) Select **Release** or **Debug** for the **Active Solution Configuration**.

- 3) Select **Win32** or **x64** for the **Active Solution Platform**.

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

- e) Update the ICU data files:

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

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

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

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

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

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

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

4. Build the ICU data library for UNIX.

- a) Compile the ICU tool and libraries.

Execute the following commands to compile the ICU:

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

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

b) Install the ICU tool.

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

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

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

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

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

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

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

c) Update the ICU data files.

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

Execute the following commands to update the ICU data file:

```
cp ./source/data/in/icudt57l.dat ./source/data/in/icudt57l.dat.bak
icupkg -a zoneinfo64.res icudt57l.dat
icupkg -a windowsZones.res icudt57l.dat
icupkg -a timeZoneTypes.res icudt57l.dat
icupkg -a metaZones.res icudt57l.dat
```

d) Compile the ICU data library.

Execute the following commands to compile ICU data:

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

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

5. Apply the ICU data library.

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

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

c) Restart the LSF Process Manager server.

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

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

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

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

b) Update the package properties files for LSF Application Center

Update the following files with the latest time zones:

- `export PPM_DIR=$GUI_TOP/3.0/wlp/usr/servers/platform/apps/platform.war/WEB-INF/classes/com/platform/gui/ppm/`
- `$PPM_DIR/view/package_zh_CN.properties`

- \$PPM\_DIR/view/package.properties
- \$PPM\_DIR/view/package\_en.properties

c) Restart the application.

```
Restart LSF Application Center:  
pmcadmin stop;pmcadmin start
```

```
Restart LSF Process Manager:  
jadmin stop;jadmin start
```

## What to do next

### Configure JS\_TIME\_EVENT\_OFFSET

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

**JS\_TIME\_EVENT\_OFFSET** parameter as a temporary measure.

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

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





---

## Chapter 3. 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.

### Tip:

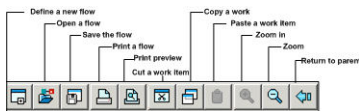
Flow Editor may not be installed if you purchased the Platform Suite for SAS. For more information, contact your sales representative.

### 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.

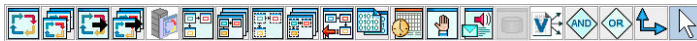
### About the toolbar

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

### About this task

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 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.

### Procedure

1. One job at a time
  - a. Draw the first job on the workspace
  - b. Define the details of the job
  - c. Draw the next job
  - d. Define the details of that job

- e. Draw the dependency line between the two jobs
- f. Continue with the next job in the flow, and repeat
2. Draw the flow, then fill in the details
  - a. Draw all of the jobs, flows and events on the workspace
  - b. Draw the dependency lines between the work items, establishing the relationships between them
  - c. Define the details for each job and job array, one at a time
3. Combine methods 1 and 2
  - a. Draw a group of jobs, job arrays and flows on the workspace
  - b. Draw the dependency lines between those work items
  - c. Define the details for each job and job array, one at a time
  - d. 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?

### About this task

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

### Procedure

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 Process Manager Client package includes sample flow definitions that you can use, to test your 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"> <li>A simple job dependency, where one job runs when its predecessor completes successfully</li> </ul>
Example_2	<ul style="list-style-type: none"> <li>The use of a recovery job</li> </ul>
Example_3	<ul style="list-style-type: none"> <li>The use of a manual task and a job array</li> </ul>
Example_4	<ul style="list-style-type: none"> <li>The use of a file event to trigger some processing</li> </ul>
Example_5	<ul style="list-style-type: none"> <li>The use of a time event to run a job array, which raises an alarm if it misses its schedule*</li> </ul>
*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 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

### Procedure

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

### Procedure

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

### Procedure

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:

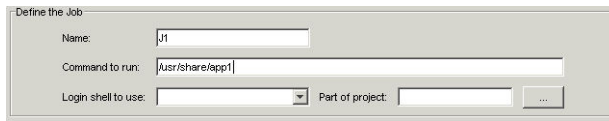
Define the Job

Name:

Command to run:

Login shell to use:  Part of project:

or on UNIX:



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:

### Procedure

- Copy a job
- Print the flow definition

### Copy a job

#### Procedure

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

#### Procedure

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 IBM Spectrum LSF* for more information. The maximum number of jobs in a job array is defined with the LSF MAX\_JOB\_ARRAY\_SIZE parameter in the LSF configuration file `lsb.params`. When a job array index is larger than the value defined by MAX\_JOB\_ARRAY\_SIZE in `lsb.params`, the job array submission is rejected.

## Insert a job array

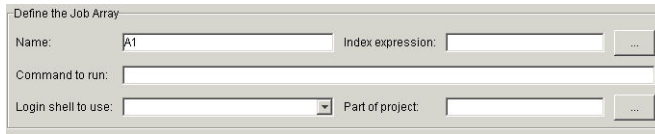
### Procedure

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

### Procedure

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

The dialog box is titled "Define the Job Array". It contains four fields: "Name:" with a text input field containing "A1", "Index expression:" with a text input field and a "... button", "Command to run:" with a text input field, and "Login shell to use:" with a dropdown menu. There is also a "Part of project:" checkbox and a text input field with a "... button.

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.

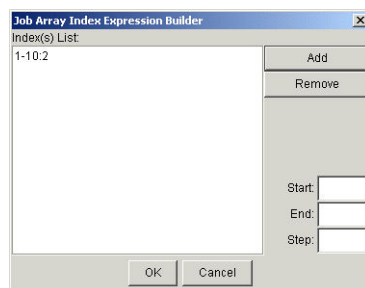
You can also specify a user variable for any of the values in the index expression—*start*, *end* or *step*. For example:

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.

The dialog box is titled "Job Array Index Expression Builder". It has a list box on the left labeled "Index(s) List" containing the entry "1-10:2". To the right of the list box are "Add" and "Remove" buttons. Below these are input fields for "Start:", "End:", and "Step:". At the bottom are "OK" and "Cancel" buttons.

- 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: `#{$step}`
- 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:  
  
`input.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:  
  
`output.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

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, 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 `input` and integers that correspond to job array indices from 1 to 1000:

`input.1, input.2, input.3, ..., input.1000`

## Include a job submission script in the flow diagram

---

### About this task

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

### Procedure

1. Click the **Insert Job 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

### Procedure

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 Process Manager Daemon jfd. Make sure that your script meets the conditions as explained in [Content of job/job array submission script](#).
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](#).
5. Click **OK**.

## Include a job array submission script in the flow diagram

---

### About this task

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.

### Procedure

1. Click the **Insert Job Array 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

### Procedure

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 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 Process Manager Daemon jfd. Make sure that your script meets the conditions as explained in [Content of job/job array submission script](#).
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](#).
7. Click **OK**.

## Content of the job/job array submission script

### About this task

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

### Procedure

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

---

### About this task

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.

### Procedure

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 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](#).

- 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

---

### About this task

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.

### Procedure

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, 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](#).

- a. Right-click the static flow array and select **Expand** to enter the subflow level.
- b. Right-click on the workspace and select **Attribute**.
- c. Click the **Exception Handling** tab and specify the exceptions and handlers.

**Tip:**

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

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:usr1:FA, and its index is 1 to 3, the three elements have the name of 1:usr1:FA(1), 1:usr1:FA(2), and 1:usr1:FA(3).

## Viewing a static flow array

### Procedure

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

---

### About this task

At any time, you can include an existing flow definition as a dynamic subflow within a flow diagram. A dynamic subflow, also called a subflow by reference, refers to a target flow that has already been submitted to Process Manager. Only flows that have been submitted to Process Manager and published are eligible target flows for dynamic subflows.

When you modify the definition of the target flow, all the dynamic subflows that reference this target flow can obtain the latest version.

### Procedure

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 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 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 Process Manager to update the dynamic subflow when the main flow is instantiated.
    - **Use the default setting specified in the main flow**
    - **Automatically update to the default version**
    - **Manually update to the default version**
  - e) 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

---

### About this task

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.

## Procedure

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 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, Process Manager replaces the variables with values, which is usually set by 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 Process Manager expands dynamic flow arrays, these input variables are set at the subflow level.

- e) Select how you want Process Manager to update the dynamic flow array when the main flow is instantiated.

- Use the default setting specified in the 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:usr1:FA`, and its index is 1 to 3, the three elements have the name of `1:usr1:FA(1)`, `1:usr1:FA(2)`, and `1:usr1:FA(3)`.

## Viewing a dynamic flow array

### Procedure

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

---

### About this task

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.

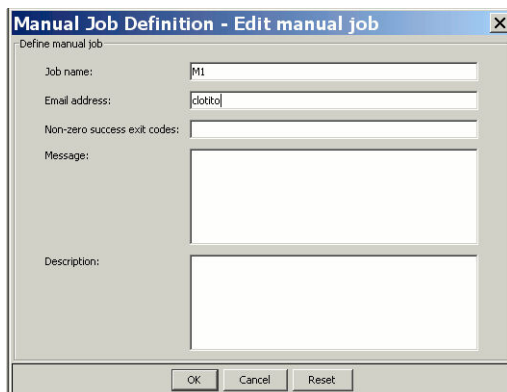
### Procedure

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

### Procedure

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



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 **Non-zero success exit codes** field, specify which numbers in addition to 0 represent success for the job. Specify a list of space-separated numbers from 1 to 255.
5. 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}.
```

6. 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.
7. 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

### Procedure

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

### About this task

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	<p>You cannot directly kill a local job in the same way as you kill any other job.</p> <p>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.</p>	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>jhist -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.

### Procedure

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

### About this task

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

### Procedure

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 Process Manager administrators, you must specify your own user name. The default is your own user name.

To set variables for local jobs, use variable files. You cannot use the command **ppmsetvar** to set variables for local jobs.

The following fields are optional. If undefined, 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 **Disable suspension for this job** field, select it in cases where you do not want to suspend a running local job when the flow is suspended. If this field is selected, when the flow is suspended, the running local job will not be suspended - it will continue to run. Only available for local jobs on UNIX/Linux.
- 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 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.

## Insert a job to another batch system

---

### About Other Batch Jobs


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

#### Limitations

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

#### Job Definition

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

Batch Job work item enabled in Flow Editor: 

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

#### Job monitoring and control

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

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

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

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

#### Dependencies

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

- **Completes successfully:** The job completes with exit code 0. If a job state is Done, it is considered as "Completes Successfully", otherwise it is considered as "Fails".

- **Fails:** The job fails.
- **Ends with any exit code:** The job exits with any exit code, including 0.
- **Ends with exit code:** The job exits with a particular exit code pattern. For example, not-equal-to, equal-to.

### Failover and Other Batch jobs

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

## Include a job for another Batch System in the flow

### About this task

Available only when Process Manager server is running on UNIX/Linux.

An Other Batch job is a job that is submitted to a remote batch system or workload manager that is not LSF. You can track and control the job like any other job in your flow.

### Procedure

1. Click the **Other Batch Job** button:



**Note:** The Other Batch Job icon will only be enabled if there are Other Batch Systems configured.

2. Left-click in the workspace in the location where you want to insert the job.

The **Other Batch Job** icon is inserted in your flow definition.

3. Draw the lines describing any dependencies the job has on other work items in the flow definition.

### Define Other Batch System job details

### About this task

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

### Procedure

1. Double-click on the Other Batch job. The Job Definition dialog is displayed.
2. Specify the job details.
3. Click **OK** to save your changes.

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

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

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

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

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

- Jobs
- Local jobs

- Job arrays
- Static subflows
- Static flow arrays
- Dynamic subflows
- Dynamic flow arrays

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

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

## Default location of output and error files

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

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

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

You can define a default working directory for flows with the parameter `JS_DEFAULT_FLOW_WORKING_DIR` in `js.conf`. Users can override the default working directory in individual work items.

## Override order for output and error file generation settings



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

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

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

### Procedure

1. In Flow Editor, from the menu select **Action > Add Flow Attribute**, and define output and error file settings.

Field	Description
<b>Create output files for jobs and job arrays</b>  <b>Create error files for jobs and job arrays</b>	Select <b>Yes</b> to configure output and error file generation and naming pattern.  <b>Important:</b> If you select to only generate output files, no error files are generated but the content of the error file is appended to the output file(same behavior as the LSF <b>bsub -o</b> option).
<b>Directory</b>	Specify a directory name or path relative to the flow's working directory. Process Manager creates specified subdirectories in the working directory if the directories do not exist.
<b>File Name</b>	Specify the naming pattern for files.  Built-in variables you can use: <ul style="list-style-type: none"> <li>• %u for user name</li> <li>• %t for time stamp</li> <li>• %J for job ID</li> <li>• %I for job array element</li> </ul> <b>Note:</b> If you specify the file naming pattern to start with a path and use a slash (/), this is interpreted as an absolute path by Process Manager. For example: if your working directory is /home/user1, and you specify the file naming pattern for output files to be /test/output/output.#{JS_FLOW_FULL_NAME}.%J, the output file will be created outside of the working directory in the directory /test/output, not within the working directory.  Default file naming pattern for output files: <ul style="list-style-type: none"> <li>• Job: output.#{JS_FLOW_FULL_NAME}.%J</li> <li>• Local job: output.#{JS_FLOW_FULL_NAME}</li> <li>• Job array element: output.#{JS_FLOW_FULL_NAME}.%J[%I]</li> </ul> Default file naming pattern for error files: <ul style="list-style-type: none"> <li>• Job: error.#{JS_FLOW_FULL_NAME}.%J</li> <li>• Local job: error.#{JS_FLOW_FULL_NAME}</li> <li>• Job array element: error.#{JS_FLOW_FULL_NAME}.%J[%I]</li> </ul>

2. Click **OK** to save your changes.

## About variables in Process Manager

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

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

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

## Types of variables

Process Manager supports three types of variables:

- Built-in variables
- User variables
- Environment variables

### Built-in variables

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

### User variables

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

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

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

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

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

User variables can also be used inside environment variables.

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

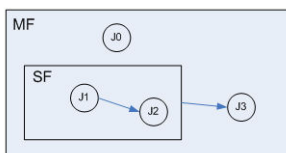
### Environment variables

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

## Scope of variables and variable override order

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

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



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

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

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

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

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

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

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

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

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

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

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

## Dynamic subflows

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

## List of Process Manager built-in variables

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

Built-in variable	Description	Usage
%I	<p>Index of a job array element.</p> <p>You use the built-in variable <b>%I</b> to obtain the index of a job array element. You can use <b>%I</b> in the Job Array Definition—Edit Job dialog, General tab, in the following fields:</p> <ul style="list-style-type: none"><li>• Input file</li><li>• Output file</li><li>• Error file</li></ul>	<p>Specify the variable as follows when you want to use its current value:</p> <p><i>file_name.%I</i></p> <p>Do not specify brace brackets and # sign.</p>
%J	<p>Job ID of a job array.</p> <p>You use the built-in variable <b>%J</b> to obtain the job ID of a job array. You can use <b>%J</b> on the Job Array Definition—Edit Job dialog, General tab, in the following fields:</p> <ul style="list-style-type: none"><li>• Output file</li><li>• Error file</li></ul>	<p>Specify the variable as follows when you want to use its current value:</p> <p><i>file_name.%J</i></p> <p>Do not specify # sign and brace brackets.</p>
JS_FLOW_ID	<p>Unique ID number of a flow.</p> <p>You use the built-in variable JS_FLOW_ID when you need to use the unique ID number of a flow.</p>	<p>Specify the variable as follows when you want to use its current value:</p> <p><i>#{JS_FLOW_ID}</i></p>
JS_FLOW_NAME	<p>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:</p> <p><i>flow_ID:username:flowname</i></p>	<p>Specify the variable as follows when you want to use its current value:</p> <p><i>#{JS_FLOW_NAME}</i></p>

Built-in variable	Description	Usage
JS_FLOW_FULL_NAME	<p>You use the built-in user variable JS_FLOW_FULL_NAME when you need to use the long version of a subflow name.</p> <p>For example,</p> <ul style="list-style-type: none"> <li>For a subflow named 11:usr1:F1:SF1:SSF1, this variable is set to 11:usr1:F1:SF1:SSF1.</li> <li>For a main flow named 11:usr1:F1, this variable is set to 11:usr1:F1.</li> <li>For a flow array element named 11:usr1:F1:FA(1), this variable is set to 11:usr1: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.</li> </ul>	<pre>#{JS_FLOW_FULL_NAME} (#{JS_FLOW_INDEX})</pre>



Built-in variable	Description	Usage
JS_FLOW_SHORT_NAME	<p>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.</p> <p>For example, there are two dynamic subflows (DSF1 and DSF2) in a main flow (11:usr1:F1), that both refer to the same target flow (TF). If the target flow sets a parent flow variable myvar, both dynamic subflows will overwrite each other's value of the myvar variable.</p> <p>To prevent this issue, for all subflows and flow arrays in a triggered flow, use the JS_FLOW_SHORT_NAME variable to indicate the name of the subflow.</p> <p>For example,</p> <ul style="list-style-type: none"> <li>• For a subflow named 11:usr1:F1:SF1:SSF1, this variable is set to SSF1.</li> <li>• For a main flow named 11:usr1:F1, this variable is set to F1.</li> <li>• For a flow array element named 11:usr1:F1:FA(1), this variable is set to FA. Note that this does not include the array index. If you need to differentiate between array elements, you must use the JS_FLOW_INDEX built-in user variable.</li> </ul>	<p>Set the parent flow variable to <i>variable_name_#JS_FLOW_SHORT_NAME</i> in the job.</p> <p>For example, for the myvar variable name, the DSF1 dynamic subflow will set <i>#myvar_DSF1</i> and the DSF2 dynamic subflow will set <i>#myvar_DSF2</i>.</p> <p>This allows both dynamic subflows to avoid variable name conflicts even though both dynamic subflows use the same target flow.</p> <p>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).</p>
JS_FLOW_WORKING_DIR	<p>Specifies the default work directory that is used if a work directory is not specified at the flow or job level.</p> <p>This parameter supports the built-in user variables <i>#JS_FLOW_NAME</i>, <i>#JS_FLOW_ID</i>, <i>%t</i> (for time stamp) and <i>%u</i> (for user name).</p> <p>The default value is the execution user's home directory:</p> <ul style="list-style-type: none"> <li>• Unix: <b>\$HOME</b></li> <li>• Windows: <b>%HOMEDRIVE% %HOMEPATH%</b></li> </ul>	<p>Specify the variable as follows when you want to use its current value:</p> <p><i>#JS_FLOW_WORKING_DIR</i></p>

Built-in variable	Description	Usage
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.	Specify the variable as follows when you want to use its current value:  #{JS_ITERATION_COUNTER}

Built-in variable	Description	Usage
JS_EVENT[n]_FILENAME	<p>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.</p> <p>If a flow is triggered by multiple files, multiple variables are created, each with a different value for <i>n</i>. The value of <i>n</i> is determined by the position of the triggering event in the list of possible flow-triggering events, including all types of events.</p> <p>Consider the following examples, where a flow definition is submitted with multiple events that can trigger the flow.</p> <p>Example: one event at a time:</p> <p>In this example, myflow is triggered to run under either of the following conditions:</p> <ul style="list-style-type: none"> <li>• At 5:00 p.m. on the first Thursday of the month (a time event)</li> <li>• If a file called payupdt arrives in the tmp directory</li> </ul> <p>Flow Attribute triggering events:</p> <p>First_thursday_of_month@Sys, 17:0</p> <p>"/tmp/payupdt" arrival</p> <p>Trigger flow when any event is true</p> <p>When the file /tmp/payupdt arrives, the name and value of the built-in variable are as follows:</p> <p>JS_EVENT[2]_FILENAME=/tmp/payupdt</p> <p>If the flow is triggered by the time event, no value is set for JS_EVENT[2]_FILENAME.</p> <p>Note that the value of <i>n</i> in the name of the variable corresponds to the position of the file event in the list of events.</p>	<p>Specify the variable as follows when you want to use its current value:</p> <p><code>#{JS_EVENT[n]_FILENAME}</code></p> <p>where <i>n</i> is the position of the triggering event in the list of possible events.</p>

Built-in variable	Description	Usage
	<p>Example: multiple events</p> <p>In this example, myflow is triggered to run when all of the following are met:</p> <ul style="list-style-type: none"> <li>• A file called payupdt arrives in the tmp directory</li> <li>• Today is Wednesday</li> <li>• The file /tmp/dbupdt exists</li> </ul> <p>Flow Attribute triggering events::</p> <p>"/tmp/payupdt" arrival</p> <p>Wednesdays@Sys, 0:0</p> <p>"/tmp/dbupdt" exist</p> <p>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:</p> <pre>JS_EVENT[1]_FILENAME=/tmp/ payupdt JS_EVENT[3]_FILENAME=/tmp/ dbupdt</pre> <p>Note that the value of <i>n</i> in the name of the variable corresponds to the position of the file event in the list of events.</p>	
JS_EVENT_ <i>n</i> _FILENAME_BASE	Base file name, part of JS_EVENT[ <i>n</i> ]._FILENAME.	<p>Specify the variable as follows when you want to use its current value:</p> <pre>#{\$JS_EVENT_<i>n</i>_FILENAME_BASE}</pre>

## Where you can use variables

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
- Project name

- Input file
- Output file
- Error file
- Email address
- Run as user name
- On the Job/Job Array Definition—Edit Job dialogs, Submit tab, in the following fields:
  - Queue name
  - Application profile name
  - Service level agreement name
- On the Job Definition—Edit Job dialog, Processing tab, in the following fields:
  - Host Requirements
    - Run on hosts
  - Number of Processors for Parallel Jobs
    - Minimum
    - Maximum
    - User Priority
      - Priority
    - User Group
      - Associate job with user group
    - Before Execution
      - Run command
- On the Job Definition - Edit Job dialog, Limits tab, in the following fields:
 

Limits tab

  - Host Limits
    - Maximum CPU time
    - Maximum run time
  - Job Limits
    - Maximum file size
    - Maximum core file size
    - Maximum memory size
    - Maximum data size
    - Maximum stack size
- 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

## Include the variable evaluator to run jobs based on decision branches

### About this task

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.

Display or hide variable evaluator expressions by selecting **View > Show Dependency Conditions**. You can also access the menu from the pop-up menu by right-clicking in the flow canvas.

### Procedure

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

### Procedure

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

#### Procedure

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

#### Procedure

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

### Procedure

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.

## Setting 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 Process Manager system needs to have one or more queues configured to accept them. Check with your 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:

- Flow 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 Process Manager Server

### Methods to set user variables

User variables are set using the following methods:

- Job starter.

This method is still supported for backwards compatibility but is deprecated. Use **ppmsetvar** instead.

- External file.

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

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

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

- The command **ppmsetvar**.

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

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

### 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 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 user 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.

## Use a user variable in a flow definition

### Procedure

Define a job that sets a runtime value for the variable, ensuring that the value will be available to Process Manager before the job that needs the value runs.

### Setting user variables with `ppmsetvar`

#### *The `ppmsetvar` command*

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

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

To set variables for local jobs, use variable files. You cannot use the command **`ppmsetvar`** to set variables for local jobs.

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

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

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

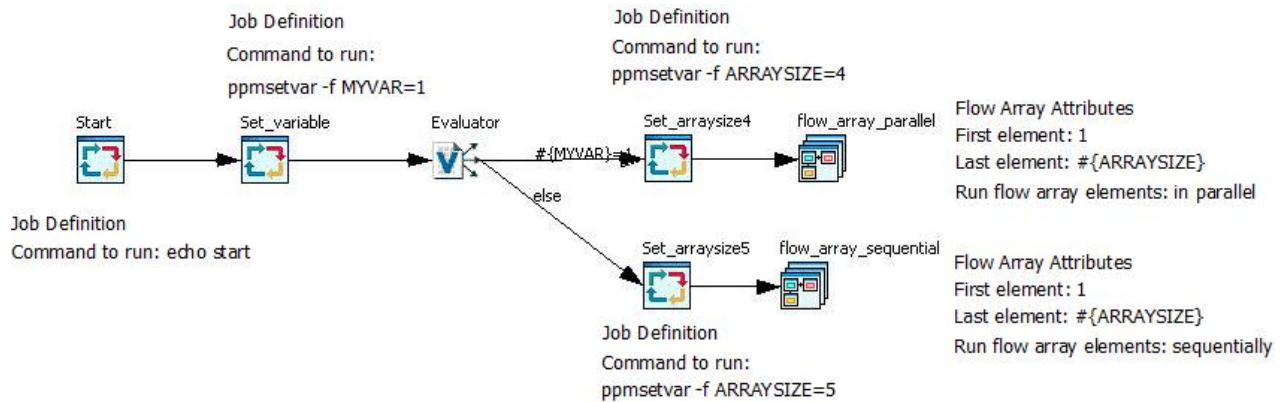
This flow contains two subflows to be run as arrays and a condition evaluator that decides whether to run the arrays in parallel or sequentially. The `Set_variable` job sets the variable `MYVAR=1`, which indicates to run the array in parallel. This flow also sets the arraysize at the time the flow array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

In this example, the job `Set_variable` sets `MYVAR=1` with `ppmsetvar`.

In the variable evaluator, when `MYVAR=1`, the job `Set_arraysize4` runs. The job `Set_arraysize4` sets the variable `ARRAYSIZE=4` with **`ppmsetvar -f`**.

In the variable evaluator, when `MYVAR` is equal to any other number, the job `Set_arraysize5` runs. The job `Set_arraysize5` sets the variable `ARRAYSIZE=5` with `ppmsetvar -f`.

The flow arrays that follow use the variable set by the the jobs `Set_arraysize4` or `Set_arraysize5` to define how many times the subflows are run as flow arrays.



### Passing variables between parent flows and subflows

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

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

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

The last job in the subflow passes the variable `result_{JS_FLOW_SHORT_NAME}=xyz#{MYVAR}` to the parent flow and also writes the variable to a file. The parent flow accesses the user variable set by the subflow by indicating the subflow name such as `echo result_Dynamic_Subflow1` and `echo result_Dynamic_Subflow2`.

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

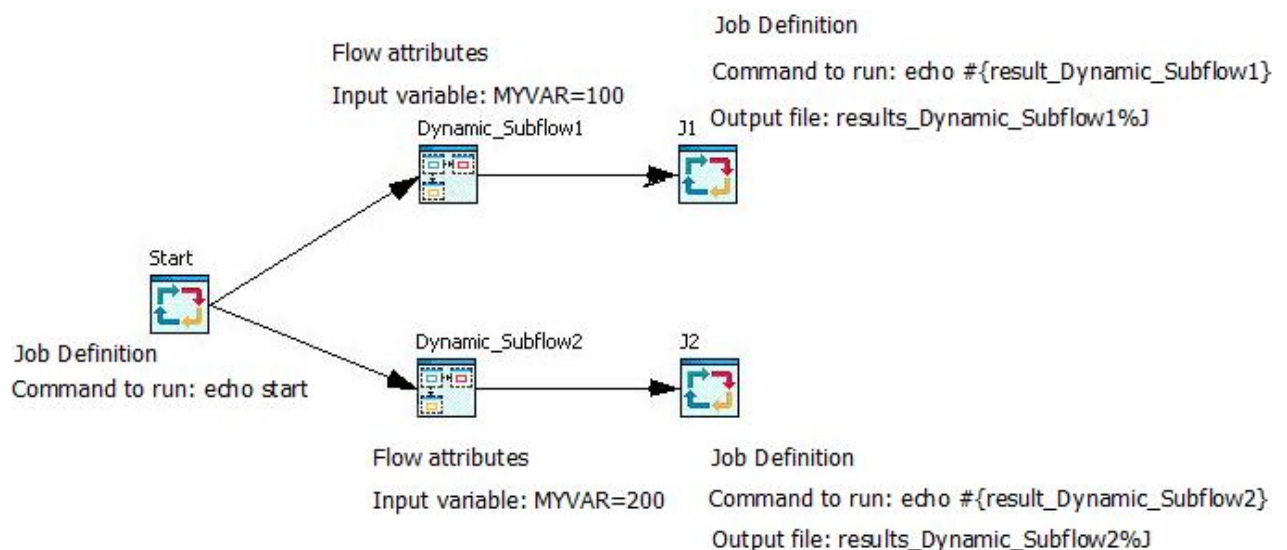


Figure 1. Parent flow

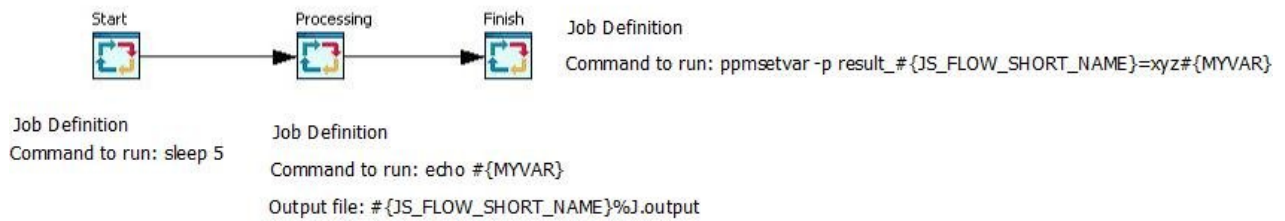
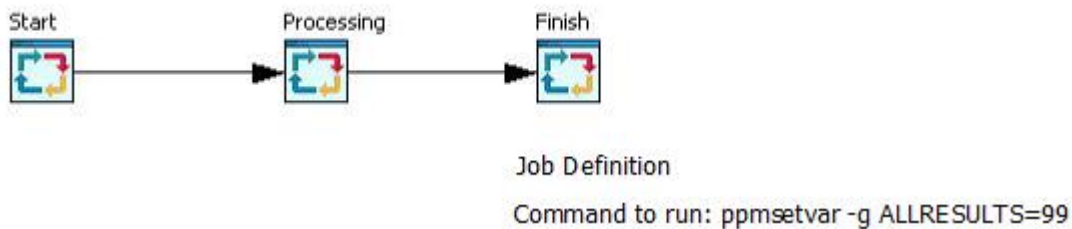


Figure 2. Subflows *Dynamic\_Subflow1* and *Dynamic\_Subflow2*

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

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

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



## Set a user variable in a Windows bat file

### About this task

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

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
...
```

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.

### Procedure

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 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%]
```

## Results

Process Manager sets the file environment variables as follows:

- Process Manager sets %JS\_FLOW\_VARIABLE\_FILE% to JS\_HOME\work\var\_comm\flowvar.job\_name.
- Process Manager sets %JS\_GLOBAL\_VARIABLE\_FILE% to JS\_HOME\work\var\_comm\globalvar.job\_name.
- Process Manager sets %JS\_PARENT\_FLOW\_VARIABLE\_FILE% to JS\_HOME\work\var\_comm\parentflowvar.job\_name.

Within the appropriate scope, 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

### About this task

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

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
...
```

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.

## Procedure

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 "LocalVar1=value1\n";
print OUT "LocalVar2=\"value2 with value\"\n";
close(OUT);
```

- b) To set global variables, whose values are available to all flows within the 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 "GlobalVar1=Gvalue1\n";
print APP "GlobalVar2=\"Gvalue2 with space\"\n";
close(APP);
```

Process Manager sets the \$JS\_GLOBAL\_VARIABLE\_FILE environment variable to JS\_HOME/work/var\_comm/globalvar.*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\[LSB_JOBINDEX\]
```

## Results

Process Manager sets the file environment variables as follows:

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

Within the appropriate scope, 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

### About this task

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, Exit, Waiting, or Suspended state, and the flow item itself (the flow, subflow, or flow array) must be in a Running, Exit, Waiting, or Suspended state.

### Procedure

1. From the flow manager, right-click on a flow, subflow, or a flow array that is either in a Running, Exit, Waiting, or Suspended state, 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:

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**.

## Dependencies

---

### Job dependencies

#### About this task

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.

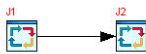
You can display or hide dependency conditions by selecting **View > Show Dependency Conditions**, or accessing the menu from the pop-up menu by right-clicking in the flow canvas.

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

## Procedure

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.

Job Event Definition

Trigger event when ...

Job name: J1

Event type: Completes successfully

Description:

OK 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

Job name: J1

Event type: Starts

Run when predecessor has exit code greater than 2

Job name: J11

Event type: Ends with exit code

Greater than 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: J12

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: J12

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, template jobs, proxy job events, and proxy job array events 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.

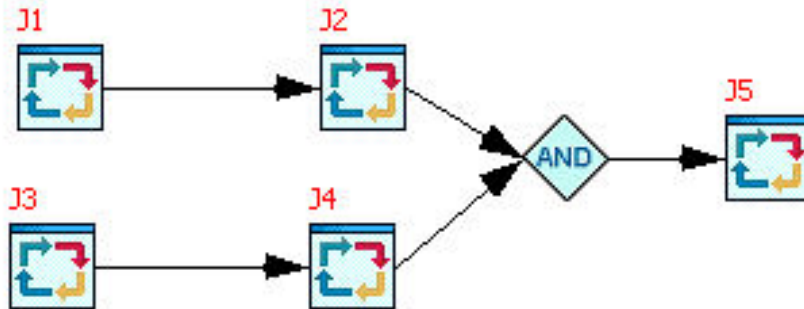


## Examples

### 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.

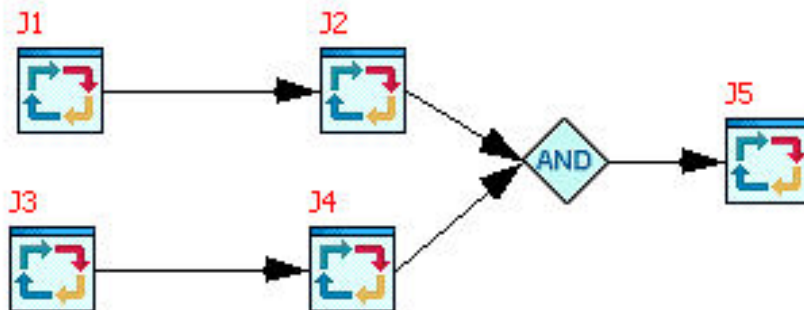
J1 and J3 cannot be considered because they are more than one link away from J5.



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

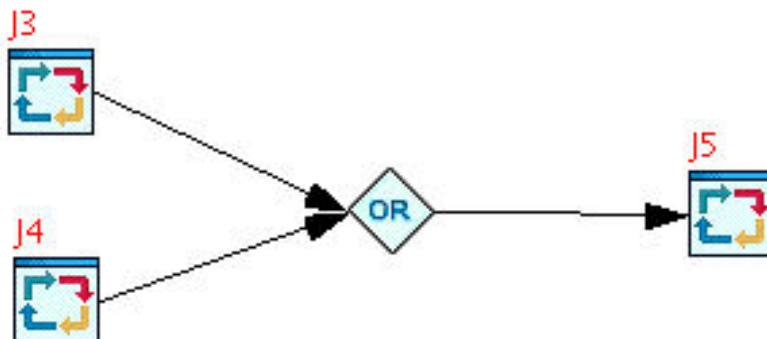
In this flow, you can only specify J2 and J4 to pre-submit J5. As a result, J5 will be submitted right after J2 and J4 are submitted to LSF.

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



### Example: Pre-submission not possible

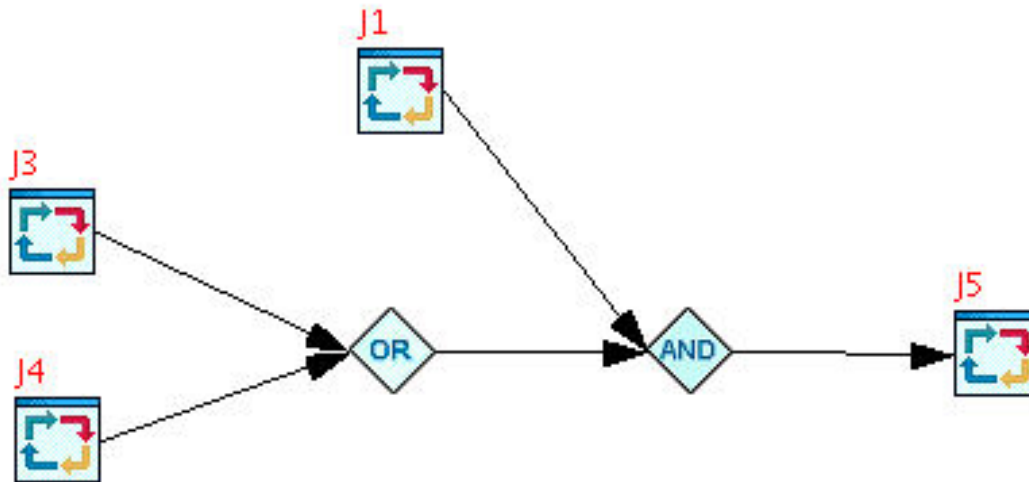
In this flow, you cannot specify any pre-submission. This is because jobs must be logically connected with AND.



### 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

#### Procedure

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

#### Procedure

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.

## Running a job in a flow based on file activity

Sometimes you do not want a work item within a flow to run until something happens to a particular file. 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.

### About this task

You can specify the following criteria for a file event:

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

### Procedure

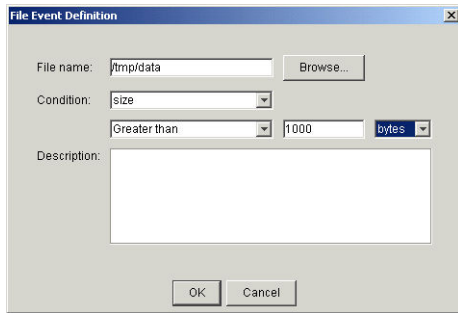
1. Click the **Insert File Event** button.
2. Click in the workspace where you want to insert the file event. The file event icon does not yet appear in the workspace. The Event Definition dialog box is displayed.
3. In the **File name** field, specify the full path name of the file as the Process Manager 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.
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.



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.

## Running a flow based on file activity

Sometimes you do not want a flow to run until something happens to a particular file. Add a file event in the Flow Attributes to automatically trigger a flow based on specific conditions. Use the file event to repeatedly trigger flows if the specified conditions are met.

### About this task

You can specify the following criteria:

- The file arrives
- The file exists (includes when the file is created)
- The file does not exist (includes when the file 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 flow, the flow will run when the file meets the specified condition. Even if the flow is still active the next time that the file meets the specified condition, the file event runs the flow only once.

### Procedure

1. In Flow Editor, select **Edit > Flow Attributes** and select the **Triggering Events** tab.
2. Click **Add**.

The **Trigger Flow with Events** dialog is displayed.

3. In **Select type of event**, select **File Event**.
4. In the **File name** field, specify the full path name of the file as the 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.

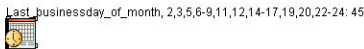
You can also specify a variable for the file name, provided your system is configured to support them.

5. In the **Condition** field, choose the appropriate condition from the list, and specify the number of bytes if applicable.
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**. The file event is displayed in the **Flow Attributes**.

## Change the label displayed for an event

### About this task

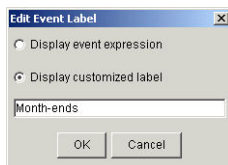
Sometimes when you define an event, the label that appears in the workspace for the event is long and cumbersome, and interferes with the readability of the flow. This is because the label used is derived from the expression that defines the event. In a time event, for example, the label displays the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

### Procedure

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

### About this task

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.

## Procedure

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.

**Note:** You can find a list of valid time zone IDs in `JS_HOME/JS_VERSION/resources/timezones.properties`.

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.

Time Event Definition

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

Calendar Name:

Time zone:

Select a 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

### About this task

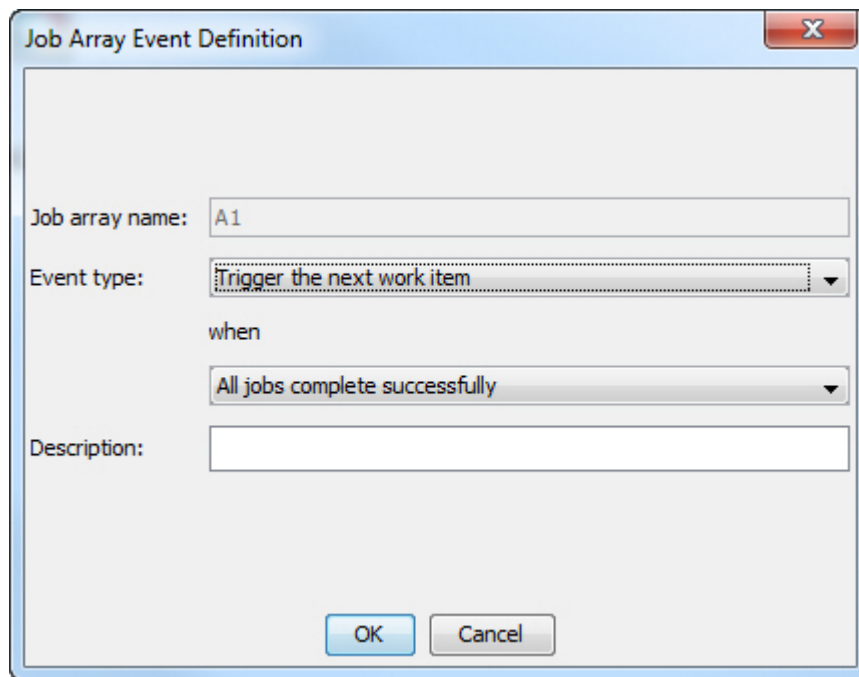
When you draw a dependency line from a job array to another work item in the flow definition, the dependency you create is the default: the work item cannot run until all of the jobs in the job array complete successfully.

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
- Any job fails
- The job array is submitted
- 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

### Procedure

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.



The dialog box is titled "Job Array Event Definition". It contains the following fields:

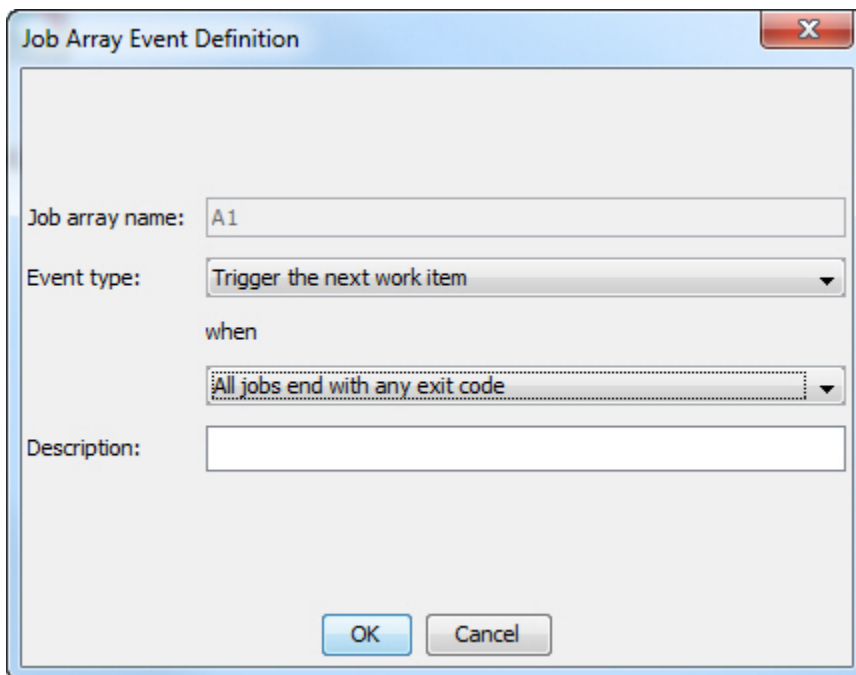
- Job array name:** A text box containing "A1".
- Event type:** A dropdown menu with "Trigger the next work item" selected.
- when:** A dropdown menu with "All jobs complete successfully" selected.
- Description:** An empty text box.

At the bottom are "OK" and "Cancel" buttons.

5. In the **Event type** field, select **Trigger the next work item**, then 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.
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

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



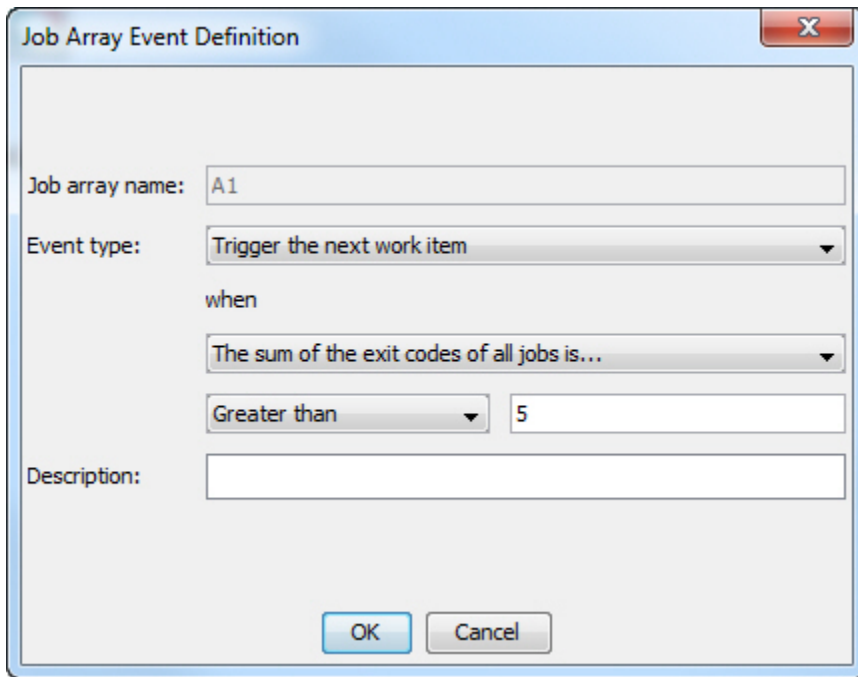
The dialog box is titled "Job Array Event Definition". It contains the following fields:

- Job array name:** A text box containing "A1".
- Event type:** A dropdown menu with "Trigger the next work item" selected.
- when:** A dropdown menu with "All jobs end with any exit code" selected.
- Description:** An empty text box.

At the bottom are "OK" and "Cancel" buttons.

Sum of the exit codes is...

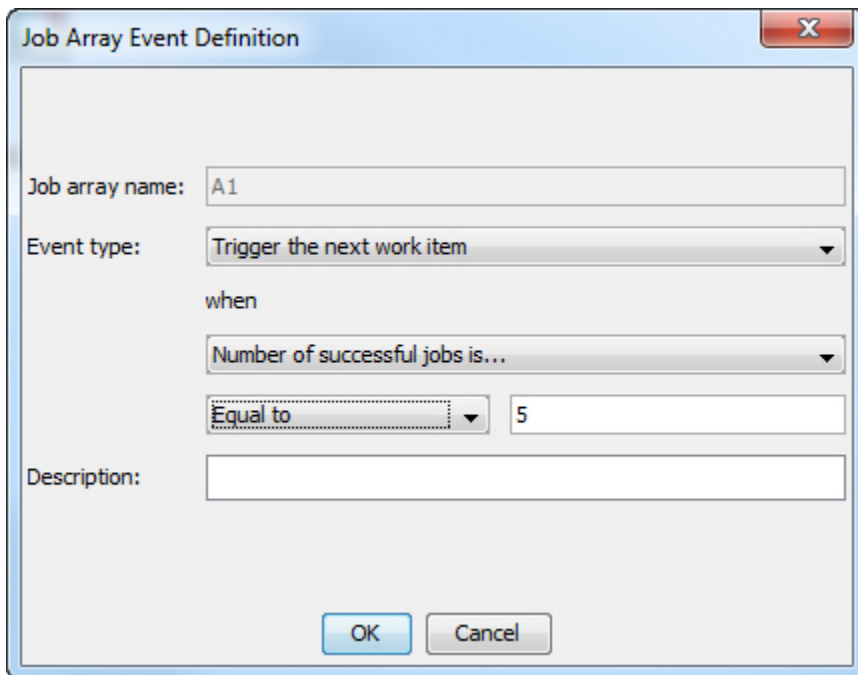




The dialog box is titled "Job Array Event Definition" and has a close button (X) in the top right corner. It contains the following fields:

- Job array name:** A text box containing "A1".
- Event type:** A dropdown menu with "Trigger the next work item" selected.
- when:** A section containing:
  - A dropdown menu with "The sum of the exit codes of all jobs is..." selected.
  - A second dropdown menu with "Greater than" selected.
  - A text box containing the number "5".
- Description:** An empty text box.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

Number of successful jobs is...



The dialog box is titled "Job Array Event Definition" and has a close button (X) in the top right corner. It contains the following fields:

- Job array name:** A text box containing "A1".
- Event type:** A dropdown menu with "Trigger the next work item" selected.
- when:** A section containing:
  - A dropdown menu with "Number of successful jobs is..." selected.
  - A second dropdown menu with "Equal to" selected.
  - A text box containing the number "5".
- Description:** An empty text box.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

Number of failed jobs is...

**Job Array Event Definition**

Job array name:

Event type:

when

Description:

Number of jobs started is...

**Job Array Event Definition**

Job array name:

Event type:

when

Description:

## Define dependencies between elements in job arrays

### About this task

In some cases, you may have a job array that depends on another job array but you want the next job array to start as soon as the first element of the first job array has finished. For example: job array A2 depends on job array A1. Each element in job array A2 can start as soon as the corresponding element of job array A1 is finished.

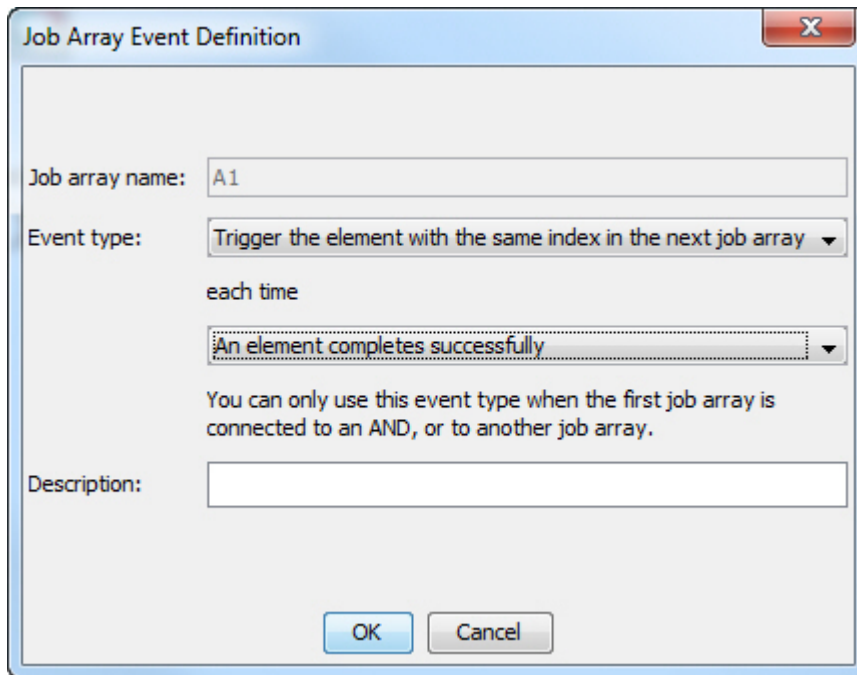
You can define job array element-to-element dependency with the **Trigger the element with the same index in the next job array** dependency definition.

Requirements to define dependencies between elements in job arrays:

- The job arrays must have the same number of elements
- The first job array must be connected to the next job array with a dependency arrow directly, or with a dependency arrow with an AND

### Procedure

1. In Flow Editor, draw a dependency arrow between one job array and the next job array.
2. Select the dependency arrow, right-click, and choose Open Definition .  
The Job Array Event Definition is displayed.
3. In the Event type field, select **Trigger the element with the same index in the next job array**, and choose the criteria.



The image shows a dialog box titled "Job Array Event Definition". It has a close button (X) in the top right corner. The dialog contains the following fields and options:

- Job array name:** A text input field containing "A1".
- Event type:** A dropdown menu with the selected option "Trigger the element with the same index in the next job array".
- each time:** A sub-label for the criteria dropdown.
- Criteria:** A dropdown menu with the selected option "An element completes successfully".
- Warning:** A text message below the criteria dropdown stating: "You can only use this event type when the first job array is connected to an AND, or to another job array."
- Description:** A text input field.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

**Note:** If you select **An element completes successfully** as the criteria, if any element in the first job array fails, the element with the same index in the next job array will remain in the PEND state and the status of the second job array will always remain Running. To prevent this, specify in the Flow Completion Attributes under **When any work item fails, Stop the flow and kill any running work items**.

4. Click **OK**.

## Specify dependencies on a subflow

### About this task

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.

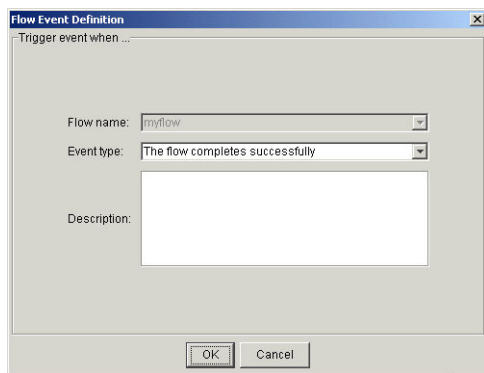
You can specify the type of dependency to be one of the following:

- 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
- The subflow fails
- The subflow fails with the specified exit code
- The subflow completes successfully with the specified exit code

- 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

## Procedure

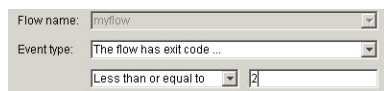
1. Draw both the predecessor subflow and the work item that succeeds it.
2. Change to job dependency mode by clicking the **Insert Dependency** button.
3. Draw a dependency line from the subflow to the work item that depends on the subflow.
4. To change the type of dependency, right-click on the dependency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



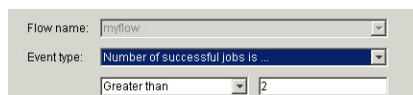
5. In the **Event type** field, select the type of dependency you want to use to trigger the successor job, and the appropriate operator and values. See the examples that follow for subflow dependencies you can use.
6. In the **Description** field, add any descriptive text that may be helpful for understanding 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



Specified number of jobs complete successfully



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.

Specified number of jobs have started

## Specify dependencies on an unconnected work item

### Specify a dependency on a proxy job

#### About this task

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.

#### Procedure

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 within which the proxy event should remain valid after it becomes true. If the number is 0(default duration), the proxy event will always be valid and will never expire after it becomes true.
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.

**Proxy Event Definition**

Create proxy for...

☒ Job ☐ Job Array ☐ Flow ☐ Subflow

Trigger event when...

Job name: myflow.J5 Owner: jheys

Duration: 1 minutes

Event type: Completes successfully

Description:

OK Cancel

- 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

### Procedure

- Change to proxy event mode by clicking the **Insert Proxy Event** button.
- 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.
- In the **Create proxy for...** box, select **Job Array**.
- 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.
- 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.
- In the **Duration** field, specify the number of minutes within which the proxy event should remain valid after it becomes true. If the number is 0(default duration), the proxy event will always be valid and will never expire after it becomes true.
- 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.
- In the **Description** field, add any descriptive text that may be used for understanding this event.

**Proxy Event Definition**

Create proxy for...

☐ Job ☒ Job Array ☐ Flow ☐ Subflow

Trigger event when...

Job array name: myflow.A1 Owner: jheys

Duration: 1 minutes

Event type: Number of jobs finished with any exit code is...

Greater than 5

Description:

OK Cancel

- 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

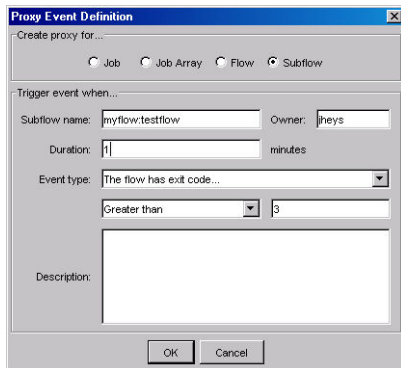
### About this task

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

- The flow completes successfully—the default
- The flow fails
- The flow ends with any exit code
- The flow fails with the specified exit code
- The subflow completes successfully with the specified exit code
- The flow has the specified exit code
- A specified number of successful jobs
- A specified number of jobs finished with any exit code
- A specified number of unsuccessful jobs
- A specified number of started jobs

## Procedure

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 within which the proxy event should remain valid after it becomes true. If the number is 0 (default duration), the proxy event will always be valid and will never expire after it becomes true.
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 flow

### Procedure

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 within which the proxy event should remain valid after it becomes true. If the number is 0(default duration), the proxy event will always be valid and will never expire after it becomes true.
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

#### Procedure

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

#### Procedure

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 a job icon, the Edit Job dialog is displayed. You use this dialog to specify any information that is required to define the job itself, such as the command it runs, and to specify any requirements that 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. Below the title bar is a tabbed interface with tabs for 'General', 'Submit', 'Processing', 'Resources', 'Limits', 'File Transfer', 'Advanced', 'Exception Handling', and 'Description'. The 'General' tab is active and contains the following sections:

- Define the Job:** This section contains several input fields and buttons:
  - Name:** A text field containing 'Processing'.
  - Command to run:** A text field containing 'sleep 5' with a 'Modify...' button to its right.
  - Login shell to use:** A dropdown menu with a downward arrow, followed by a 'Part of project:' text field and a 'Modify...' button.
  - Working directory:** A text field.
  - Environment Variables:** A text field with a 'Modify...' button to its right.
  - Non-zero success exit codes:** A text field.
- Input file:** A section with a 'File name:' text field.
- Output Files:** A section with a checkbox labeled 'Override flow settings for output file creation'.
- Error Files:** A section with a checkbox labeled 'Override flow settings for error file creation'.
- Email Notification:** A section with a checkbox labeled 'Notify when job:' (which is checked), a dropdown menu set to 'starts', and an 'Email address:' text field containing 'clotito'.
- Run As:** A section with a 'User name:' text field.

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.

**Note:** On Windows, if you specify a dependency on a job with the -w option to the **bsub** command, ensure you use single quotes (') if the job name contains special characters. For example, if your command is sleep 1 and you want to specify a dependency on job with the name a:b, then specify: -w "done('a:b')"  
sleep 1

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 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. Use an absolute path rather than a relative path. The working directory is also the job's submission directory (the directory from which the job is submitted). 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 non-zero success exit codes

Specify which numbers in addition to 0 represent success for the job. Specify a space-separated list of exit codes from 1 to 255.

## 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/.

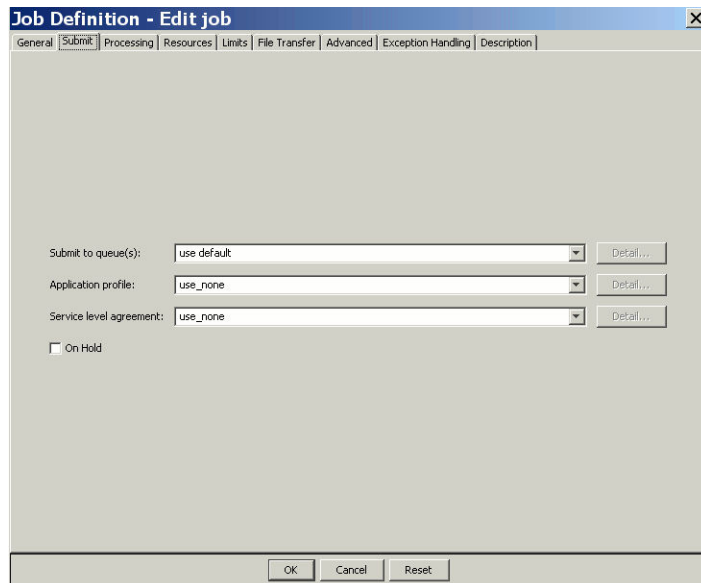
## Notify a user when the job ...

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. 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.

You put a job on hold when you want to stop a flow at a specific point so that you can fix problems.

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.

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**.

For more details, see [“Stop a flow at a specific point by putting a job on hold” on page 142](#).

## The Processing tab

The screenshot shows the 'Job Definition - Edit job' dialog box with the 'Processing' tab selected. The dialog has several sections: 'Host Requirements' with radio buttons for 'Run on host(s)', 'Must have exclusive use of host', and 'Rerun if host becomes unavailable', and a 'Same host as:' dropdown; 'Number of Processors for Parallel Jobs' with 'Minimum' and 'Maximum' spinners; 'User Priority' with a 'Priority' spinner; 'User Group' with a text field and a checkbox 'Associate job with user group'; and 'Before Execution' with a 'Run command' text field. At the bottom are 'OK', 'Cancel', and 'Reset' buttons.

### 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 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 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 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.

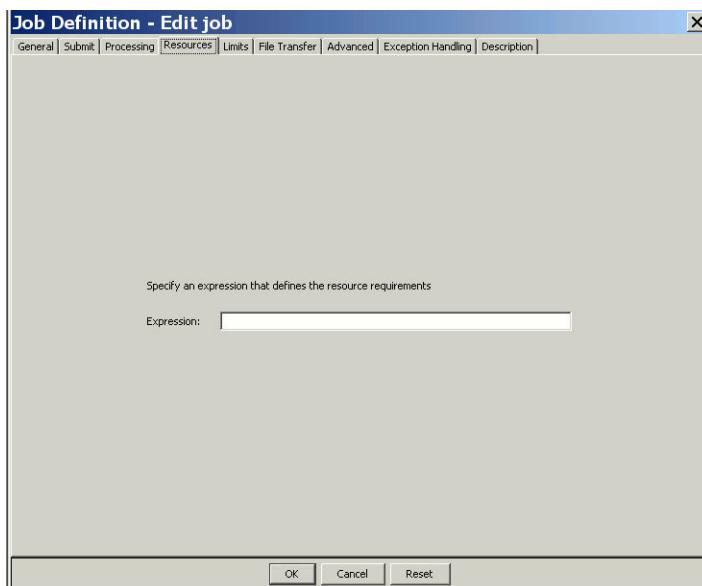
In the **Run command** field, specify the command to be run on the execution host before running the actual job. If the command runs successfully, the job can run. Otherwise the command and job are rescheduled. Be sure the command is capable of being run multiple times on the target host.

### 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



The screenshot shows a window titled "Job Definition - Edit job" 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 (selected), Limits, File Transfer, Advanced, Exception Handling, and Description. The main area of the window is a light gray rectangle. Inside this area, the text "Specify an expression that defines the resource requirements" is centered. Below this text is a label "Expression:" followed by a text input field. At the bottom of the window, there are three buttons: "OK", "Cancel", and "Reset".

### 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 IBM Spectrum 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]
```

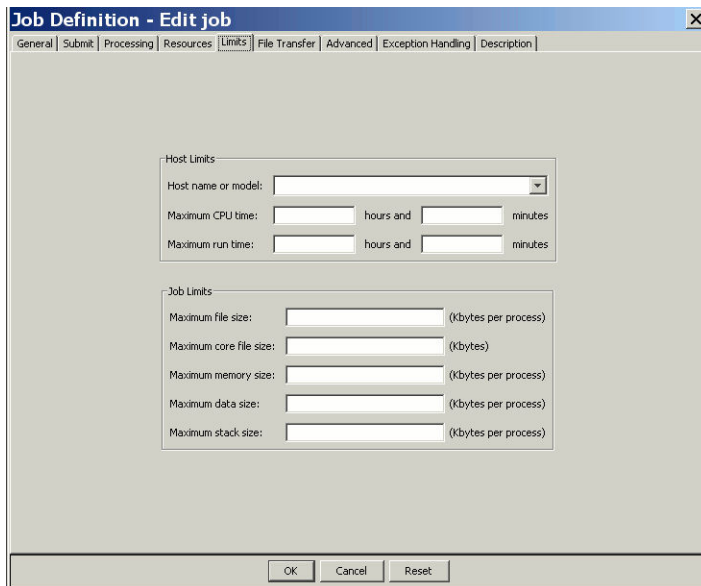
In the above example, `verilog` must first be defined as a shared resource in LSF.

## 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



### 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 the 'Job Definition - Edit job' dialog box with the 'File Transfer' tab selected. The dialog has several tabs: General, Submit, Processing, Resources, Limits, File Transfer (active), Advanced, Exception Handling, and Description. The 'File Location' section contains two text fields: 'Local path including name:' and 'File on execution host:'. The 'Operation' section has three radio buttons: 'Copy file to remote host before running job' (selected), 'Copy file to local location after running job', and 'Append file to local location after running job'. The 'Expression(s)' section features a large text area and three buttons: 'Add', 'Remove', and 'Replace'. At the bottom are 'OK', 'Cancel', and 'Reset' buttons.

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.

#### Procedure

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.

#### Procedure

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.



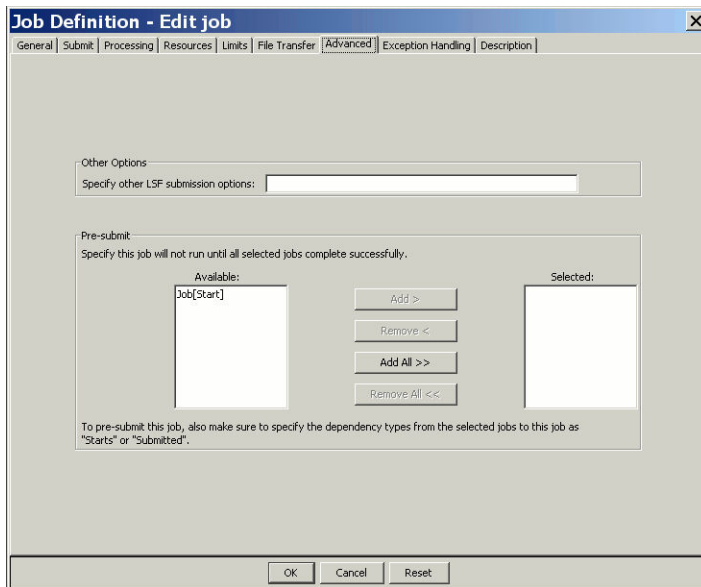
## Append output to a local file after the job runs

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:

### Procedure

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: **-I**, **-Ip**, **-Is** for interactive jobs, **-K** for submitting a job and waiting for it to complete, and **-e** and **-o** as these options are available in the General tab for error and output files.

- 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:

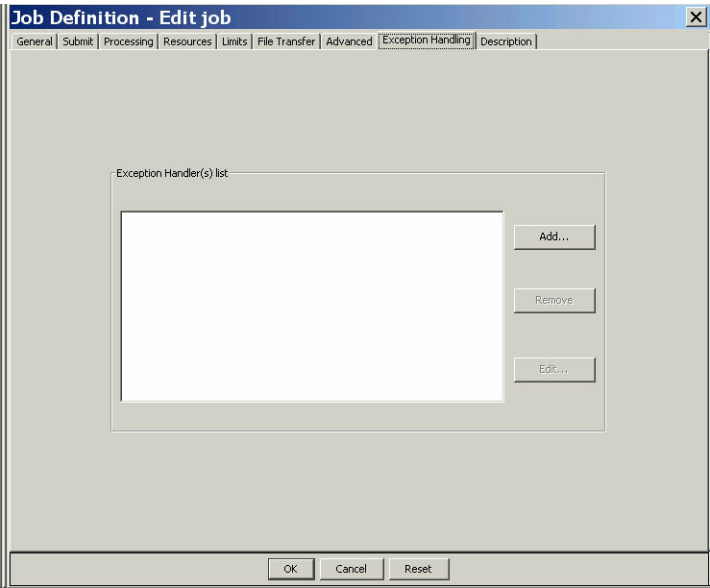
```
-w "exit('{JS_FLOW_FULL_NAME}({JS_FLOW_INDEX}):J1', > 2) "
```

## 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



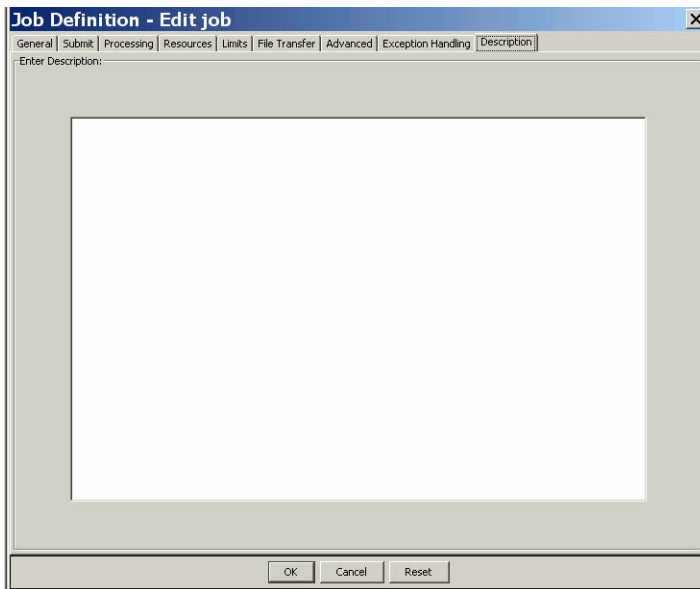
You use this tab to specify what action to take if a specific exception occurs while running this job or job array.

## 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.



## Determine the flow state with Flow Completion Attributes

---

### About completion attributes

Because flows can be as individual as their creators, and may contain recovery jobs that run when another job fails, 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 Process Manager, the latter by defining flow completion attributes.

You define flow completion attributes to a flow to describe the criteria the 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 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, 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.

#### 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.

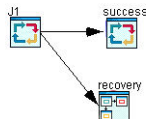
### Alternative completion behavior

You can direct Process Manager to continue processing work in a flow even after it is considered complete and has been assigned a state. In this case, 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

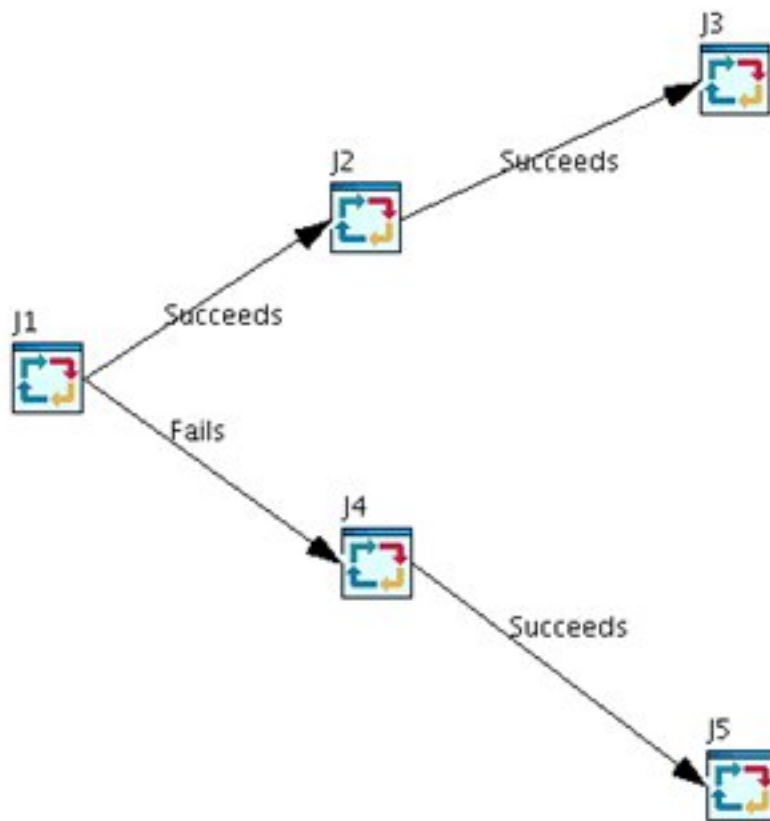
You may define a flow that contains multiple branches. In this flow, if one branch fails, you may not want the flow to stop processing. Perhaps you want to let the flow to run as much as it can, and then you will perform some manual recovery and rerun the failed branch.

## Defining when a flow is successful or failed when using branches in a flow

### About this task

In cases in which you have a flow that has multiple branches and only some branches may run you want to be able to specify to Process Manager to identify a flow as successful when all jobs that are able to run have completed, and identify the flow as failed if any job in the flow fails. You can configure this in the flow's Flow Completion Attributes.

For example, you have a flow with this type of scenario, in which you run one branch when a job succeeds and another branch when the job fails. In this type of case, all jobs in the flow will never run. You want the flow to be considered successful when jobs J1, J2, or J3 complete successfully. When jobs J1, J2, J4 or J5 fail, you want the flow to be considered failed, but you do want jobs J4 and J5 to run to completion.



### Procedure

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

The Flow Completion Attributes dialog is displayed.

2. In **Determine the state of the flow when....**, select **All work items complete successfully or any work item fails**, and select **Ignore work items in the Waiting state that will never run, or that depend on a file, time, or proxy event** to place a checkmark in it.

You select the checkbox to indicate to determine the state of the flow when all work items that have been submitted to run have completed successfully, or any work item that has been submitted to run has failed. If work items are not submitted to run (such as a branch of a flow), they should not be considered in determining the state of the flow.

3. Under **When any work item fails**, select **Continue running the flow and when the flow is complete, change the flow state**.

In our example, this allows the branch in the flow that runs when J1 fails to continue running until completion.

4. Click **OK**.

### Specify flow completion attributes

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

#### Procedure

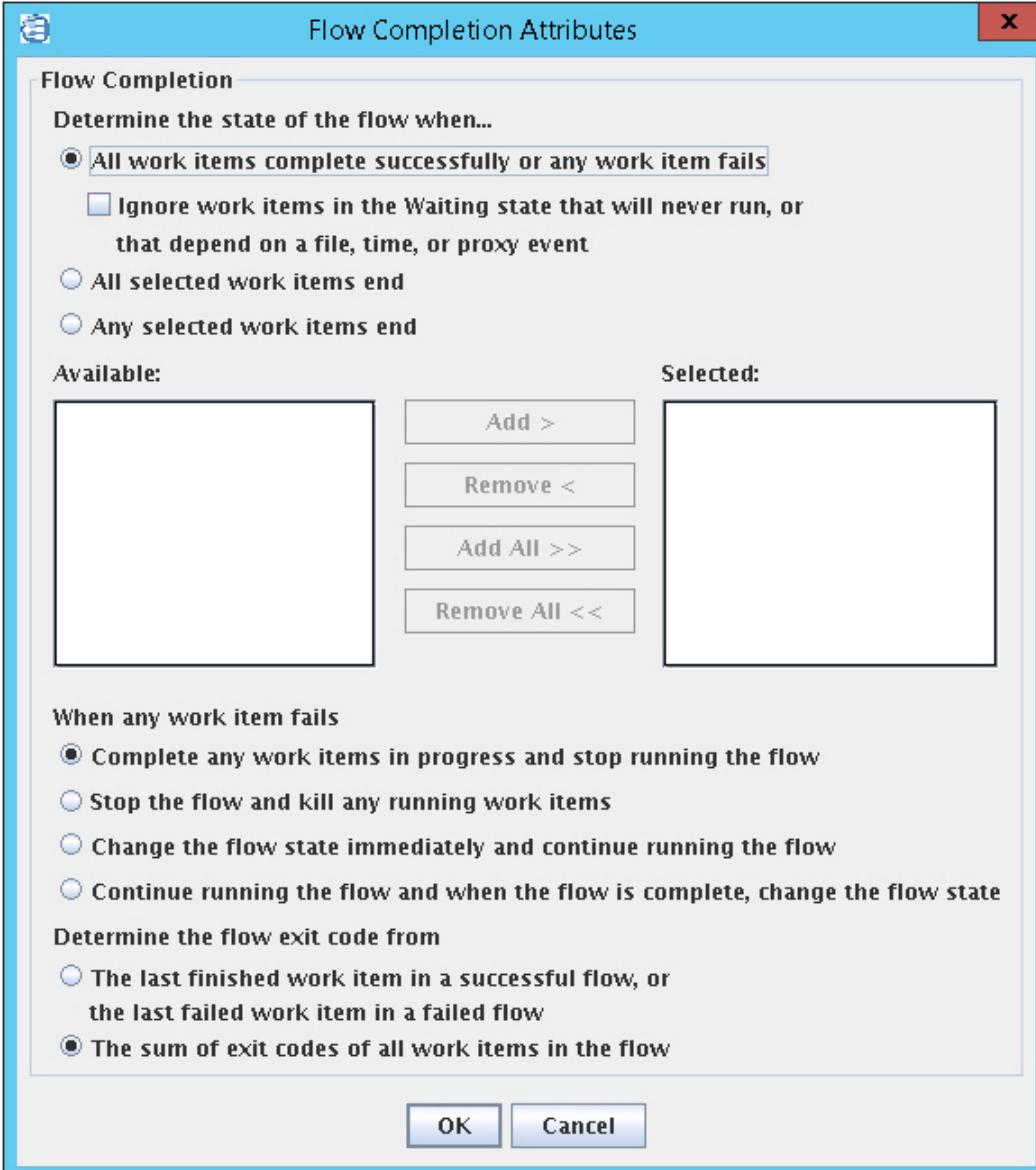
- All work completes successfully or any work item fails. This is the default.
- All selected work items end.

- Any selected work items end. You can also specify what 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.
- Determine the flow exit code either from the sum of exit codes of all work items in the flow(this is the default), or from the last finished work item in a successful flow, or the last failed item in a failed flow.

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

#### Procedure

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.



The dialog box is titled "Flow Completion Attributes" and contains the following sections:

- Flow Completion**
  - Determine the state of the flow when...**
    - ☒ All work items complete successfully or any work item fails
    - ☐ Ignore work items in the Waiting state that will never run, or that depend on a file, time, or proxy event
    - ☐ All selected work items end
    - ☐ Any selected work items end
- Available:** (Empty list box)
- Selected:** (Empty list box)
- Buttons between lists: Add >, Remove <, Add All >>, Remove All <<
- When any work item fails**
  - ☒ Complete any work items in progress and stop running the flow
  - ☐ Stop the flow and kill any running work items
  - ☐ Change the flow state immediately and continue running the flow
  - ☐ Continue running the flow and when the flow is complete, change the flow state
- 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
  - ☒ The sum of exit codes of all work items in the flow

At the bottom are **OK** and **Cancel** buttons.

2. Leave the first option set to the default **All work completes successfully or any work item fails**.

3. When a work item fails, you can either stop the flow or continue running the flow.

To stop the flow if any work item fails:

- If you want the flow to stop running if any work item exits but complete work that is currently running, leave the option as the default **Complete any work in progress and stop running the flow**.
- If you want the flow to stop running if any work item exits and stop any running work items, choose **Stop the flow and kill any running work items**.

To continue the flow if any work item fails:

- If you want the flow 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, select **Continue running the flow and when the flow is complete, change the flow state**.

4. Select how the flow exit code is calculated in **Determine the flow exit code from**.

5. 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**

#### **Procedure**

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. When all selected work items end, you can either stop the flow or continue running the flow.

To stop the flow when all selected work items end:

- If you want the flow to stop running but complete work that is currently running, leave the option as the default **Complete any work in progress and stop running the flow**.
- If you want the flow to stop running and stop any running work items, choose **Stop the flow and kill any running work items**.

To continue the flow when all selected work items end:

- If you want the flow 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, choose **Continue running the flow and when the flow is complete, change the flow state**.

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**

#### **Procedure**

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. When any selected work items end, you can either stop the flow or continue running the flow.

To stop the flow when any selected work items end:

- If you want the flow to stop running but complete work that is currently running, leave the option as the default **Complete any work in progress and stop running the flow**.
- If you want the flow to stop running and stop any running work items, choose **Stop the flow and kill any running work items**.

To continue the flow when any selected work items end:

- If you want the flow 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, choose **Continue running the flow and when the flow is complete, change the flow state**.

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

#### Procedure

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 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

#### Procedure

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 and when the flow is complete, change the flow state**.
3. Click **OK**. The flow will continue to run and only change state once 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

### Procedure

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**

3. Click **OK.**

## Configure dependencies for subflows

### About this task

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.

### Procedure

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

---

### About this task

You can use 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

### Procedure

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. The working directory is also the work item's submission directory (the directory from which the work item is submitted).

You can also specify the flow working directory by triggering a flow and setting the variable `JS_FLOW_WORKING_DIR`.

If you do not specify the flow working directory in the flow definition and you specify `JS_FLOW_WORKING_DIR` when you trigger the flow, the value you specify with the variable becomes the flow's default working directory. If the working directory is specified in the flow definition, the `JS_FLOW_WORKING_DIR` variable has no effect.

The override order is:

1. The working directory that is defined at the job level in the flow definition.
2. The working directory that is defined at the flow level in flow definition.
3. The working directory that is passed in with the **`JS_FLOW_WORKING_DIR`** variable when the flow is triggered.
4. The working directory that is defined with the **`JS_DEFAULT_FLOW_WORKING_DIR`** parameter in the `js.conf` file.

You can use user variables when specifying the working directory.

## Output and error files

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

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

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

## Email notification for a flow

By default, 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 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

---

### Procedure

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_#{Var1}`, you must have defined the `Var1` variable first. `Var1` can either be defined in the current flow level (if it was defined before `MyVar_#{Var1}`) 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

### Procedure

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 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 Process Manager.

## Loop a flow or subflow

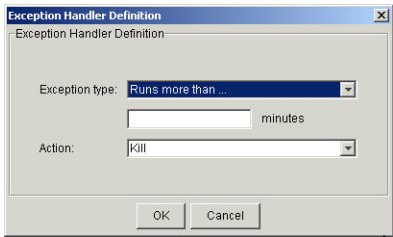
### About this task

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:



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:

### Procedure

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

### Procedure

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 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.



---

## Chapter 4. About Process Manager exceptions

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

### Process Manager exceptions

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. 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 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 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 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 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 $n$
Job array	An exit code of $n$
	$n$ unsuccessful jobs

## About exception handling

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

### 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 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

Flow		
When a Flow Experiences this Exception ...	and the Handler Used is ...	This Happens ...
Overrun	Kill	The flow is killed. All incomplete jobs in the flow are killed. The flow status is 'Killed'
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'

## Subflow

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

## Job or job array

<b>When a Job or Job Array Experiences this Exception ...</b>	<b>and the Handler Used is ...</b>	<b>This Happens ...</b>
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.

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

### About this task

When you define a job, job array, flow or subflow, you can specify what exceptions you want 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:

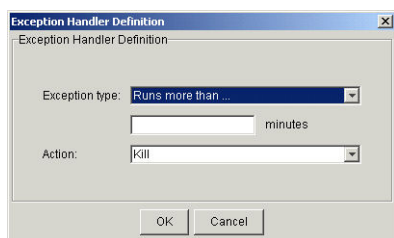
### Procedure

- [“Handle exceptions of a job or job array using built-in handlers” on page 112](#)
- [“Handle exceptions of a subflow using built-in handlers” on page 113](#)
- [“Handle exceptions with a recovery job” on page 114](#)
- [“Handle exceptions with a recovery flow” on page 114](#)

## Handle exceptions of a job or job array using built-in handlers

### Procedure

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.



- 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

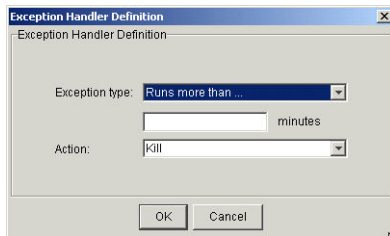
If you specify a rerun exception, you can specify a number of minutes to delay before rerunning the subflow and the maximum number of times you want the exception handler to rerun the subflow.

- Click **OK**. The exception handling specification is added to the list.
- 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

### Procedure

- Within the flow definition in the Flow Editor, right-click on the subflow.
- Select **Attributes**. The Subflow Attributes dialog appears.
- Click on the **Exception Handling** tab.
- Click **Add**. The Exception Handler Definition dialog appears.

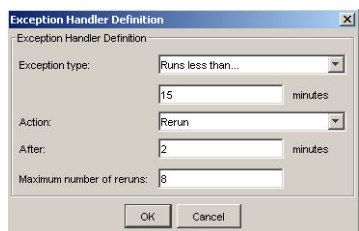


- In the **Exception type** field, select the exception you want to handle.
- Select the corresponding criteria for the **Exception type** that you specified.
  - If you chose **Runs more than...**, specify the maximum time, in minutes, the subflow can run before it should be killed or should trigger 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

If you monitor for this exception...	This handler is used...
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:



8. Click **OK**. The exception handling specification is added to the list.
9. Repeat steps 3 through 6 until you have finished specifying exceptions to handle, then click **OK**.

## Handle exceptions with a recovery job

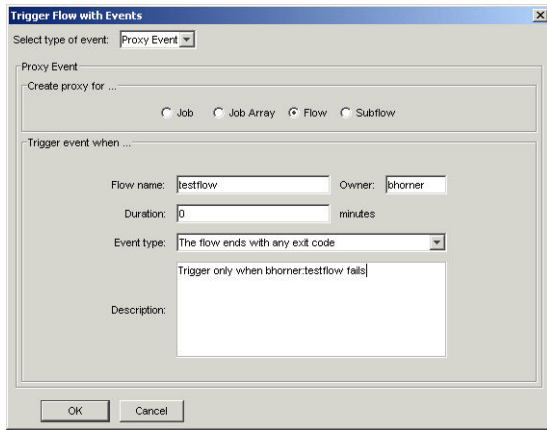
### Procedure

1. Within the flow definition in the Flow Editor, draw both the predecessor and recovery jobs (or job arrays or subflows).
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 recovery job.
4. Right-click on the dependency line and select Open Definition. The Event Definition dialog box appears.
5. In the Event Type field, select the appropriate exception.
6. Click OK.

## Handle exceptions with a recovery flow

### Procedure

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.



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 Process Manager system, where it will await the appropriate conditions to be run.

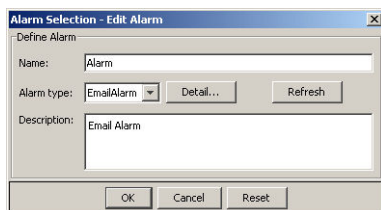
## Alarms

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 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

#### Procedure

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.

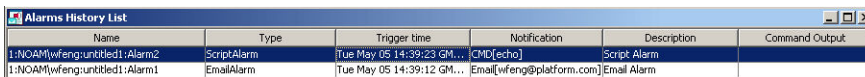


4. In the **Name** field, specify a unique name for the alarm. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (\_) in the job array name. A unique name is automatically assigned to the alarm, but you can change it to make it more meaningful.
5. In the **Alarm type** field, select the type of alarm you want to use from the list of configured types. Alarms are configured by your 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

### Procedure

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 GMT...	CMD[echo]	Script Alarm	
1:NOAM[wfeng:untitled1:Alarm1	EmailAlarm	Tue May 05 14:39:12 GMT...	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

### About this task

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 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:

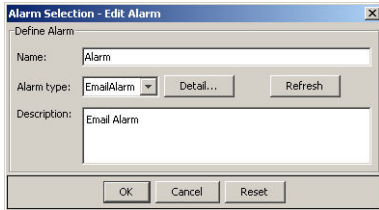
- By inserting an alarm as a successor to a work item in the flow, and specifying a dependency on the work item that opens the alarm when the dependency is met. This method is recommended when it is important to have a visual cue in the flow definition that an alarm is defined in a particular place. 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:

### Procedure

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.



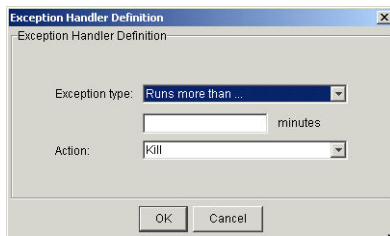
4. In the **Name** field, specify a unique name for the alarm. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (\_) in the alarm name. A unique name is automatically assigned to the alarm, but you can change it to make it more meaningful.
5. In the Alarm type field, select the type of alarm you want to use from the list of configured types. Alarms are configured by your 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.

### Procedure

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 Process Manager administrator. To see an updated list of alarms, click **Refresh**.
8. Click **OK**. The exception handling specification is added to the list.

---

## Chapter 5. Run your flow

### Note:

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

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 Process Manager.

The first decision you need to make is how the flow will be triggered. (*Triggering a flow* is the act of telling 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 [Creating a flow definition to be triggered manually](#) for instructions.

If you can specify a recurring schedule for the flow, see [Run a flow at a specific time](#) for instructions.

If you want to run a flow whenever something happens to a particular file, see [Run a flow based on file activity](#) 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 [Running your flow once](#) 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 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.

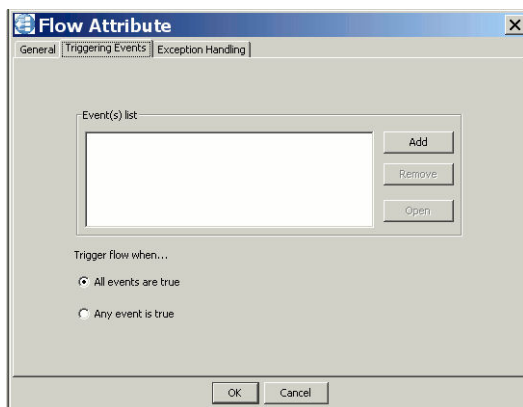
## Create a flow definition to be triggered manually

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

### Procedure

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

### Procedure

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.

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

#### Procedure

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 Process Manager, where it will be scheduled at the specified time, on each day that the specified calendar is true.

### From the command line

#### Procedure

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

### Procedure

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](#).
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 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:

### Procedure

- 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. For example, to run a flow every hour, in the Hours field, specify an asterisk (\*). Do not use an asterisk(\*) in the Minutes field as it will be too frequent. For minutes, use 5 minutes as the minimum time.

## Run a flow based on file activity

### In the flow definition

#### Procedure

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 Process Manager Server sees it, that is to be monitored for the activity, or click **Browse** to locate the file in the file system.

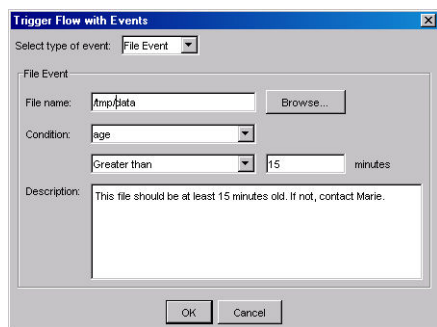
When specifying the file name, you can also specify wildcard characters: \* to represent a string or ? to represent a single character. For example, a\*.dat\* matches abc.dat, another.dat and abc.dat23. S??day\* matches Satdays.tar and Sundays.dat. \*e matches smile.

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](#) 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 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:

File name:	<input type="text" value="/tmp/core"/>	<input type="button" value="Browse..."/>
Condition:	<input type="text" value="exists"/>	

When triggering a flow when a file exists, keep the following in mind:

- 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 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 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:

File name:	<input type="text" value="/tmp/update"/>	<input type="button" value="Browse..."/>
Condition:	<input type="text" value="does not exist"/>	

After the flow is triggered, it will trigger again each time 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:	<input type="text" value="/tmp/data"/>	<input type="button" value="Browse..."/>
Condition:	<input type="text" value="age"/> <input type="text" value="Greater than"/> <input type="text" value="15"/> <input type="text" value="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:	<input type="text" value="/tmp/* tar"/>	<input type="button" value="Browse..."/>
Condition:	<input type="text" value="arrival"/>	

## From the command line

### Procedure

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
```

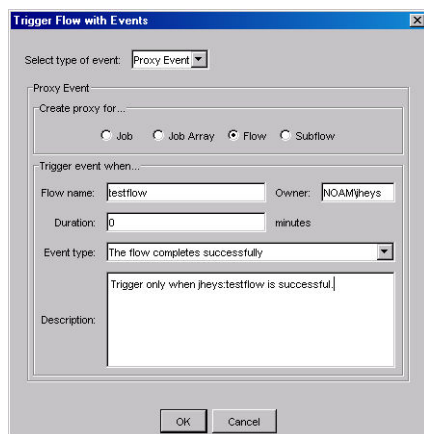
## Run a flow when another flow...

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

#### Procedure

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.

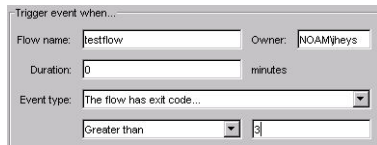


12. Click **OK**. The Triggering Event(s) tab reappears, and the proxy event you defined appears in the list.
13. Click **OK**.
14. From the **Action** menu, select **Submit** to submit the flow. The flow definition is submitted to Process Manager, where it is triggered when the specified file event is true.

## Examples

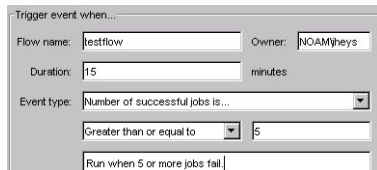
- Triggering when a flow has exit code greater than 3:

The following proxy event triggers the flow when the flow testflow exits with an exit code greater than 3:



- 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 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

### Procedure

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*

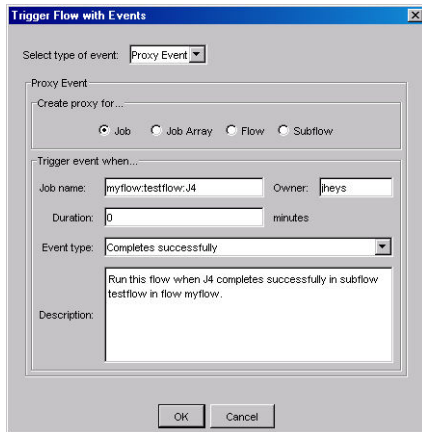
If the job is not defined within a subflow, simply specify the flow name and the job name, separated by a colon.

#### 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 Process Manager, where it is triggered when the specified file event is true.

### From the command line—trigger when job fails

#### Procedure

1. On the command line, to trigger when the job fails, type the following:  
`jsub -p "job(exit(bhorner: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

#### Procedure

1. Ensure that the 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

#### Procedure

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

---

### About this task

Until you submit a flow definition, Process Manager is not aware of it. Submitting a flow definition places it under the control of Process Manager. Once a flow definition is submitted, 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.

### Procedure

- [Submit your flow without version comments.](#)
- [Submit your flow with version comments.](#)

### Results

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.

### What to do next

After submitting the flow to 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

### Procedure

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

### Procedure

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.

- 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.



---

## Chapter 6. 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
- Remove a flow
- Kill, run or rerun a job
- Force a job to complete

The 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 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*)

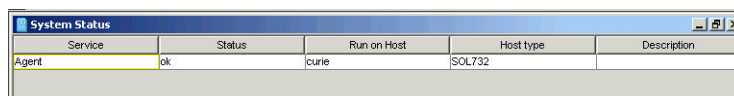
When a view is selected, the right-hand pane shows the graphical illustration of the currently selected flow definition or flow.

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

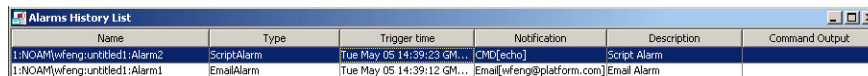
You can view the current status of the Process Manager system from the **View** menu, by selecting **System Status**. The System Status view displays the status of the Process Manager agents—the hosts that run the jobs.



Service	Status	Run on Host	Host type	Description
Agent	ok	curie	SOL732	

## List of alarms

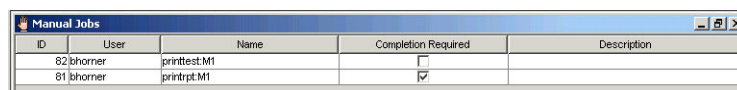
When you choose to view the alarms (from the **View** menu, select **Alarms**), a window shows all of the open alarms in the system. Open alarms remain in the list until the history log file containing the alarm is archived or deleted.



Name	Type	Trigger time	Notification	Description	Command Output
1:NOAM[wfeng:untitled1:Alarm2	ScriptAlarm	Tue May 05 14:39:23 GMT...	CMD[echo]	Script Alarm	
1:NOAM[wfeng:untitled1:Alarm1	EmailAlarm	Tue May 05 14:39:12 GMT...	Email[wfeng@platform.com]	Email Alarm	

## List of manual jobs

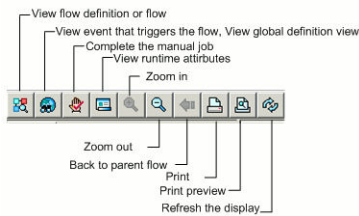
When you choose to view manual jobs (from the **View** menu, select **Manual Jobs**), a window displays all of the manual jobs in the Process Manager system. Those that are waiting for acknowledgement now have a check mark in the Completion Required field.



ID	User	Name	Completion Required	Description
82	bhomer	printtestM1	<input type="checkbox"/>	
81	bhomer	printprt.M1	<input checked="" type="checkbox"/>	

## About the toolbar

The Flow Manager toolbar looks like this:



## 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 automatic refresh entirely.

### Refresh the data displayed manually

#### Procedure

Click the refresh button or from the **View** menu, select **Refresh**.

### Change the automatic refresh interval

#### Procedure

1. From the **File** menu, select **Properties**.  
The **Properties** dialog is displayed.
2. Specify the number of seconds between refreshes of the data.
3. Click **OK**.

## Print data

---

#### About this task

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.

#### Procedure

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

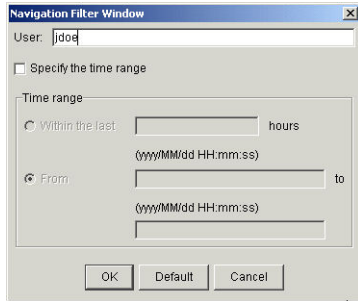
---

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 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

### Procedure

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.

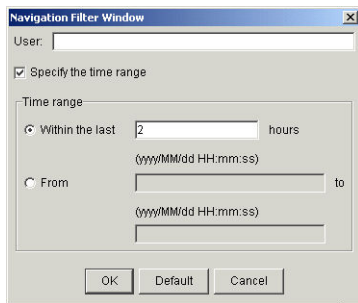


4. Click **OK**. The view of the flows is refreshed with the new filter applied.

## Limit the flows displayed to last x hours

### Procedure

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.

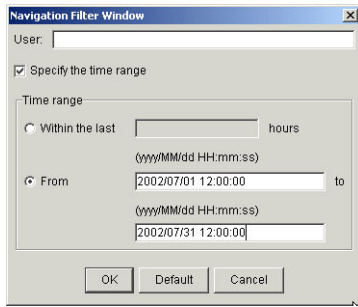


5. Click **OK**. The view of the flows is refreshed with the new filter applied.

## Limit the flows displayed to a time period

### Procedure

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.



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 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 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

##### Procedure

1. In the Flow Manager, select **By Definition**.
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

##### Procedure

1. On the command line, type the following:  

```
jtrigger flow_definition_name
```

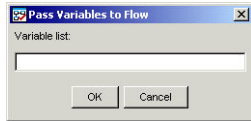
where *flow\_definition\_name* is the name of the flow definition you want to trigger.
2. Press **Enter**.

### Trigger a flow, passing it values for variables

#### From the Flow Manager

##### Procedure

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

---

### About this task

When working within the Flow Manager, you are not limited to working with flows—you can also view the definition of a flow.

### Procedure

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.
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

### About this task

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, 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.

### Procedure

1. In the Flow Manager, select **By Definition**.
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

### About this task

You can view relevant flow instance information summary:

### Procedure

1. In the Flow Manager, select **By Definition**.
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

---

### About this task

You can view all of your flow definitions in 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.

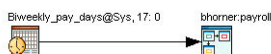
You use the global view to see how your flow definitions relate to each other, and to see what triggers each flow. From the global view, you can:

- 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 Process Manager, or are on hold, waiting for a manual trigger

View the events that trigger a flow:

### Procedure

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:

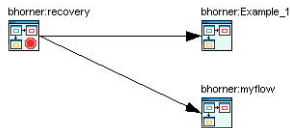


## View proxy dependencies that trigger flows

### Procedure

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

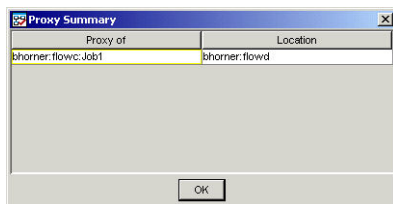
### Procedure

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`.

3. 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

### Procedure

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.
3. Right-click on the dependency line representing the dependency you want to complete.
4. 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

### Procedure

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.

When a flow has a dependency on another flow whose definition does not yet exist in 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

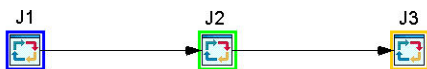
### Colored border around the icon

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

When the Color is ...	The State is ...	Which means the Work Item ...
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.  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]    : bhorner
[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

---

### About this task

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.

### Procedure

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. Note that you cannot kill a manual job.

### From the Flow Manager

#### Procedure

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

#### Procedure

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**.

## Run or rerun a single job

---

### About this task

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.

### Procedure

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

### Procedure

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**.

## Stop a flow at a specific point by putting a job on hold

---

### About this task

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.

#### Procedure

1. Select the **By Definition** 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

#### Procedure

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.

## Mark a job complete

---

#### About this task

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.

#### Procedure

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

#### Procedure

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**.

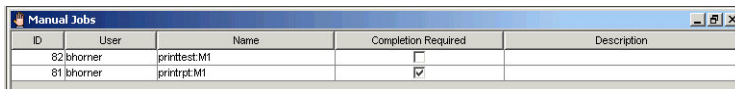
## Work with manual jobs

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

#### Procedure

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	printpst.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

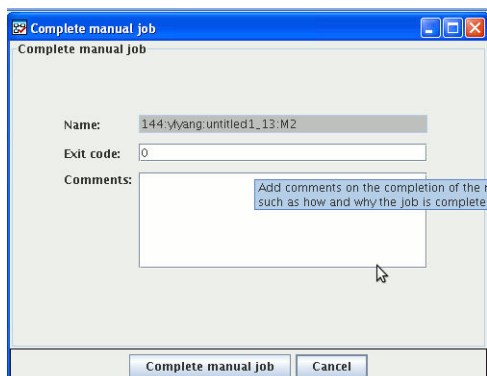
#### Procedure

1. On the command line, type the following:  
`jmanuals`
2. Press **Enter**.

### Complete a manual job

#### Procedure

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 the **Complete Manual Job** button. The **Complete manual job** dialog appears.
5. If applicable, in the **Comments** field, specify any comments required to describe what happened. For example:



Complete manual job

Name: 144.yyang.untitled1\_13.M2

Exit code: 0

Comments:   
Add comments on the completion of the r  
such as how and why the job is complete

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

#### Procedure

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 Exit.

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

### Complete a manual job with an exit code

#### Procedure

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 **jcomplete** 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

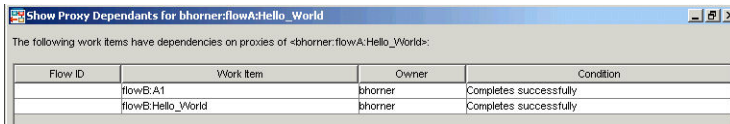
---

#### About this task

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.



The following work items have dependencies on proxies of <bhorner:flowAHello\_World>:

Flow ID	Work Item	Owner	Condition
flowB:A1	bhorner	bhorner	Completes successfully
flowB:Hello_World	bhorner	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:

#### Procedure

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

#### Procedure

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

#### Procedure

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

### Procedure

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

---

### About this task

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.

### Procedure

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

### Procedure

1. On the command line, type the following:

```
jkill flow_id
```

where *flow\_id* is the unique ID of the flow you want to kill.

2. Press **Enter**.

## Suspend a running flow

---

### About this task

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.

### Procedure

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

### Procedure

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

---

### About this task

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.

### Procedure

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

### Procedure

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

---

Using Flow Manager, a flow that is in an Exited state can be rerun.

### About this task

You can rerun a flow that is in the Exited, Done, or Killed state, as well as the Running or Waiting state.

When you rerun a flow, the flow runs again, starting at the jobs that exited and any jobs set as rerun starting points, and all successor jobs are also rerun, even if a job's status is Done.

### 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.

### Procedure

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.

## From the command line

### Procedure

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

Using Flow Manager, a flow can be rerun from a selected starting point.

### About this task

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.

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, Killed, Exited, Waiting, Running, or Pending 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.

### Procedure

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**.

## Results

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

### About this task

To remove a work item as a rerun starting point, use the flow manager to unset a work item as the rerun starting flow.

### Procedure

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

---

Using Flow Manager, a flow that is still in a Running state can be rerun.

### About this task

You can rerun a flow that is in the Running or Waiting state, as well as the Exited, Done, or Killed 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.

### Procedure

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

---

### About this task

You can rerun an exited job array. You can rerun the entire job array, or only those elements of the array that exited. When you rerun an exited job array, the job array has a new ID.

**Note:**

Rerunning an exited job array that triggers an alarm will not reopen a previously opened alarm.

**Procedure**

1. In the Flow Manager, locate the job array you want to rerun.
2. Right-click on the job array and select **Run**.

## Hold a flow definition

---

**About this task**

You can hold a flow definition that has been submitted to the 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.

**Procedure**

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

**Procedure**

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

---

**About this task**

When a flow definition is placed on hold, it cannot be triggered automatically until it has been explicitly released.

**Procedure**

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

### Procedure

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**.

## View a flow definition and specify versioning options

---

### About this task

When working within the Flow Manager, you are not limited to working with flows—you can also view the definition of a flow.

### Procedure

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.
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

### About this task

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, 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.

### Procedure

1. In the Flow Manager, select **By Definition**.
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

**About this task**

You can view relevant flow instance information summary:

**Procedure**

1. In the Flow Manager, select **By Definition**.
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**.

## Remove a flow definition

---

**About this task**

When you no longer require a flow definition, you can remove it from the list of flows the Process Manager system knows about. If you remove a flow definition, and some flows belonging to the flow definition are still in the Process Manager system, they appear in the Flow Manager in the **adhoc** folder.

**Procedure**

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 Flow**.
4. Confirm that you want to remove this definition. The flow definition is removed from the system.

## From the command line

**Procedure**

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**.



---

## Chapter 7. Mainframe support

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

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

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

### How does it work?

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

### Requirements

- A valid z/OS mainframe user ID

### Limitations

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

---

## Using mainframe

To use the mainframe support, you must:

1. Copy the template file `z/OS_Template.xml` from `JS_TOP/10.2/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_Template.xml` file from `JS_TOP/10.2/examples` to `JS_TOP/work/templates`.
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:

Parameter Name	Parameter Value
z/OS host name*	devs0101.lst.platform.com
Login user ID*	jneys
Password*	*****
Is your JCL file located on z/OS host?	Yes, my JCL file is located on z/OS host
JCL file*	Jusr/jobs/entry.jcl
Output file*	Jusr/jobs/entry.jcl.out
Estimated run time (in minutes)	1
Check interval (in minutes)	1
Time out (in minutes)	0

### 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.
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	<p>(Optional) Specify file transfers between the Process Manager Server and LSF execution hosts.</p> <p>Format:</p> <p><i>"local_file operator [remote_file]"</i></p> <p>where:</p> <ul style="list-style-type: none"><li>• <i>local_file</i> is the path and file name on the Process Manager server</li><li>• <i>operator</i> : The operator(&lt;, &gt;, &lt;&gt;, &gt;&lt;, &lt;&lt;) specifies whether the file is copied to the remote host, or whether the file is copied back from the remote host. The operator must be surrounded by white space. The following are valid operators:<ul style="list-style-type: none"><li>– &lt; Copies the remote file to the local file after the job completes. Overwrites the local file if it exists.</li><li>– &gt; Copies the local file to the remote file before the job starts. Overwrites the remote file if it exists.</li><li>– &lt;&gt; Copies the local file to the remote file before the job starts. Overwrites the remote file if it exists. Then copies the remote file to the local file after the job completes. Overwrites the local file.</li><li>– &gt;&lt; Copies the local file to the remote file before the job starts. Overwrites the remote file if it exists. Then copies the remote file to the local file after the job completes. Overwrites the local file.</li><li>– &lt;&lt; Appends the remote file to the local file after the job completes. The local file must exist.</li></ul></li></ul> <p>Example: copy the local file on the Process Manager server <code>c.s</code> to the LSF execution host and name the file <code>a.s</code>:</p> <pre>/share/usr1/c.s &gt; /home/usr1/a.s</pre> <ul style="list-style-type: none"><li>• <i>remote_file</i> is the path and file name on the LSF execution host</li></ul>
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	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.

Exit Code	Failure Reason
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.





---

## Chapter 8. Commands

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

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

You cannot use commands to create a flow definition.

### Calendar commands

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

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

### Flow definition commands

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

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

### Flow monitor and control commands

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

- **flowmanager**—to start the Flow Manager graphical user interface
- **jalarms**—to list open alarms
- **jcomplete**—to complete a manual job
- **jflows**—to display information about a flow
- **jjob**—to kill or run a job, or to mark a job complete

- **jkill**—to kill a flow
- **jmanuals**—to list all manual jobs waiting for completion
- **jrerun**—to rerun an exited flow
- **jresume**—to resume a suspended flow
- **jsetvars**—to change the value of a local or global variable while a flow is running
- **jstop**—to suspend a flow

#### Other commands

- **jid**—to verify the connection between the Process Manager Client and the Process Manager Server
- **jadmin**—to control the Process Manager daemon on Unix
- **jhist**—to view the historic information about server, flow definitions, flows, and jobs.
- **jreconfigalarm**—to reload the alarm definitions.
- **jreconfigadmin**— to dynamically reconfigure and update the list of administrators.

## caleditor

---

starts the Calendar Editor Process Manager Client.

#### Synopsis

`caleditor`

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

#### Examples

`caleditor`

opens the Calendar Editor.

## floweditor

---

starts the Flow Editor Process Manager Client.

#### Synopsis

`floweditor [file_name [file_name ...]]`

#### Description

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

#### Note:

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

#### Options

##### *file\_name*

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

## Examples

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

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

```
floweditor
```

opens the Flow Editor with no files opened.

## flowmanager

---

starts the Flow Manager Process Manager Client.

### Synopsis

```
flowmanager
```

### Description

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

### Example

```
flowmanager
```

opens the Flow Manager.

## jadmin

---

controls the Process Manager daemon `jfd` on UNIX.

### Synopsis

```
jadmin [-s] start
```

```
jadmin stop
```

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

### Description

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

### Options

#### start

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

#### -s start

Starts the Process Manager daemon on UNIX in single-user mode. You must be the primary Process Manager administrator to use this option.

#### stop

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

**-h**

Prints the command usage to stderr and exits.

**-V**

Prints the Process Manager release version to stderr and exits.

### Examples

```
jadmin start
```

Starts the Process Manager daemon.

```
jadmin -s start
```

Starts the Process Manager daemon in single-user mode.

```
jadmin stop
```

Stops the Process Manager daemon.

### See also

**jfd, js.conf**

## jalarms

---

lists the open alarms in Process Manager.

### Synopsis

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

```
jalarms [-h][[-V]]
```

### Description

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

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

### Options

#### **-u *user\_name***

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

#### **-f *flow\_name***

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

#### **-i *flow\_ID***

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

**-t *start\_time,end\_time***

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

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

The time interval can be specified in many ways.

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

**Time interval format**

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

Specify the time interval as follows:

*start\_time,end\_time|start\_time[,end\_time|start\_time*

Specify *start\_time* or *end\_time* in the following format:

*[year/][month/][day]/[hour:minute|hour:]|.|-relative\_int*

Where:

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

***start\_time,end\_time***

Specifies both the start and end times of the interval.

***start\_time,***

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

***,end\_time***

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

***start\_time***

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

**Absolute time examples**

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

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

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

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

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

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

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

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

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

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

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

### Relative time examples

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

,.-2/ = the time of the first occurrence to Mar 9 17:06 2002

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

Example: Display all opened alarms for the last seven days

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

## jcadd

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

### Synopsis

```
jcadd [-s | -u user_name][-d description] -t "cal_expression" "cal_name"
```

```
jcadd [-h][[-V]]
```

### Description

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

### Options

#### -d *description*

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

#### -u *user\_name*

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

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

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

#### -s

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

### **-t *cal\_expression***

Specifies the dates on which you want some action to take place. You can enter specific dates, a range of dates, or a more complex expression that resolves to a series of dates.

#### **Note:**

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

### ***cal\_name***

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

### **-h**

Prints the command usage to `stderr` and exits.

### **-v**

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

## **Limitations**

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

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

Some examples that follow this format are:

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

where ABC, DEF, HIJ are predefined calendars.

## **Creating calendar expressions**

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

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

### **To create absolute dates:**

Specify the date in the following standard format:

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

For example:

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

Specify multiple dates separated by commas. For example:

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

### To create schedules that recur daily:

Specify the expression in the following format:

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

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

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

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

### To create schedules that recur weekly:

Specify the expression in one of the following formats:

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

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

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

or

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

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

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

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

### To create schedules that recur monthly:

Specify the expression in one of the following formats:

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

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

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

or

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

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

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

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



or

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

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

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

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

### To create schedules that recur yearly:

Specify the expression in the following format:

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

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

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

### To merge calendar expressions:

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

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

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

### Built-in keywords-reserved words

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

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

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

### Examples

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

Creates a calendar called Mon\_Not\_Holiday. This calendar resolves to any Monday that is not a holiday, as defined in the Holidays system calendar.

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

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

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

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

```
jcadd -d "Db report dates" -t "*:JAN,JUN,DEC:day(1)" db rpt
```

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

### See also

**jcdel**, **jcals**

## jcals

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

### Synopsis

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

```
jcals [-h][[-V]]
```

### Description

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

- user name
- calendar name

- the expression

## Options

### **-l**

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

### **-u *user\_name***

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

### ***cal\_name***

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

### **-h**

Prints the command usage to `stderr` and exits.

### **-V**

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

## Examples

```
jcalcs -u all
```

Displays all calendars in Process Manager.

## jcdel

---

deletes an existing calendar.

## Synopsis

```
jcdel [-f][-u user_name] cal_name [cal_name ...]
```

```
jcdel [-h][[-V]]
```

## Description

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

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

## Options

### **-f**

Specifies to force the deletion of the calendar.

### **-u *user\_name***

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

***cal\_name***

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

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

**Examples**

```
jcdel -u "barneyt" Runday2001
```

Deletes the calendar Runday2001 owned by the user barneyt.

**See also**

**jcadd, jcal**

## jcmod

---

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

**Synopsis**

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

```
jcmod [-h][[-V]]
```

**Description**

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

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

**Options****-d *description***

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

**-u *user\_name***

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

**-t *cal\_expression***

Specifies the dates on which you want some action to take place. You can enter specific dates, a range of dates, or a more complex expression that resolves to a series of dates.

***cal\_name***

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

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

### Creating calendar expressions

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

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

#### To create absolute dates:

Specify the date in the following standard format:

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

For example:

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

Specify multiple dates separated by commas. For example:

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

#### To create schedules that recur daily:

Specify the expression in the following format:

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

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

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

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

#### To create schedules that recur weekly:

Specify the expression in one of the following formats:

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

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

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

or

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

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

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

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

### To create schedules that recur monthly:

Specify the expression in one of the following formats:

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

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

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

or

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

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

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

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

or

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

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

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

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

### To create schedules that recur yearly:

Specify the expression in the following format:

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

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

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

### To merge calendar expressions:

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

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

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

### Built-in keywords—reserved words

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

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

### EXAMPLES

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

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

## jcommit

---

submits a flow definition to Process Manager.

### Synopsis

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

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

```
jcommit -h|-V
```

### Description

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

*Note:* The flow definition may contain pre-defined events that trigger the flow. Use the -k option to preserve triggering events defined in the flow definition. If you do not use the use the -k option, when you submit this flow using the **jcommit** 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 and do not use the -k option, 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 [-v *version*]

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

If you do not specify -r and the flow definition already exists, submission fails.

#### -d

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

#### -m "*ver\_comment*"

Submit the flow with version comments. A flow version number is returned after each successful submission.

#### -T *time\_event*

Overwrites time events specified in the flow definition. 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]
```

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



### ***cal\_name***

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

### ***hour:minute***

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

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

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

### **%duration**

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

### **-F "file\_event"**

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

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

### **Note:**

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

Specify the file event in one of the following formats:

*arrival(file\_location)*

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

```
absolute_directory/filename
```

*exist(file\_location)*

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

```
absolute_directory/filename
```

*! exist(file\_location)*

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

```
absolute_directory/filename
```

*size(file\_location) operator size*

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

```
absolute_directory/filename
```

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

**Note:**

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

*age(file\_location) operator age*

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

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

```
absolute_directory/filename
```

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

**Note:**

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

**-p "proxy\_event"**

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

### **-C combination\_type**

Overwrites combination events specified in the flow definition. When multiple events are specified, the combination type specifies whether one event is sufficient to 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.

### **-k**

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

### **flow\_file\_name**

Specifies the name of the file containing the flow definition.

### **-h**

Prints the command usage to `stderr` and exits.

## **-V**

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

## **Examples**

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

Submit 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
```

Submit 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.

## **jcomplete**

---

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

### **Synopsis**

```
jcomplete [-d description] [-u user_name] [-e exit_code]-i flow_id  
flow_name[:subflow_name]:manual_job_name
```

```
jcomplete [-h][[-V]]
```

### **Description**

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

### **Options**

#### **-d *description***

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

#### **-e *exit\_code***

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

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

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

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

**-i *flow\_id***

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

***flow\_name:subflow\_name>manual\_job\_name***

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

```
myflow:prtcheck:prtpage
```

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

**-u *user\_name***

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

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

## Examples

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

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

## See also

**jmanuals jjob**

## jdefs

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

### Synopsis

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

```
jdefs [-h][[-V]]
```

### Description

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

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

- the state of each flow
- flow version history and details

## Options

### **-l**

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

- any events defined to trigger the flow
- any exit conditions specified in the flow definition
- the default version and the latest version of the flow

### **-v**

Displays the version history of the flow.

### **-u *user\_name***

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

### **-s *status***

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

#### **ONHOLD**

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

#### **RELEASE**

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

### ***definition\_name***

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

### **-h**

Prints the command usage to `stderr` and exits.

### **-V**

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

## Examples

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

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

## jexport

---

exports flow definitions to a file

### Synopsis

```
jexport [-o] [-u user_name | -u all] [-s status] [flow_name ...]  
jexport [-o] [-u user_name] -v version flow_name | -v all flow_name  
jexport -h | -V
```

### Description

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

### Options

#### -o

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

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

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

#### -s *status*

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

- ONHOLD: Flow definitions that are not eligible to be automatically triggered.
- RELEASE: Flow definitions that are not On Hold.

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

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

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

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

#### -h

Prints the command usage to `stderr` and exits.

#### -V

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

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

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

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

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

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

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

## jflows

---

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

### Synopsis

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

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

```
jflows [-h][[-V]]
```

### Description

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

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

### Options

#### -l

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

#### -u *user\_name*

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

#### -f *flow\_name*

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

#### -s *state*

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

##### Done

Displays information about flows that completed successfully.

##### Exit

Displays information about flows that failed.



**Killed**

Displays information about flows that were killed.

**Running**

Displays information about flows that are running.

**Suspended**

Displays information about flows that were suspended.

**Waiting**

Displays information about flows that are waiting.

***flow\_id***

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

**0**

Specifies to display all flows.

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

**Examples**

```
jflows -f myflow
```

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

## jhold

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

**Synopsis**

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

```
jhold [-h][[-V]]
```

**Description**

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

**Options****-u *user\_name***

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

***flow\_name***

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

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

**Examples**

```
jhold myflow
```

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

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

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

**See also****jrelease**

## jid

---

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

**Synopsis**

```
jid [-h|-V]
```

**Description**

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

**Options****-h**

Prints command usage to `stderr` and exits.

**-V**

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

## jjob

---

controls a job in a running flow.

**Synopsis**

```
jjob [-u user_name] -i flow_id -c|-k|-r|-p|-g|-l [-a] "flow_name[:subflow_name]:job_name"
```

```
jjob [-h][[-V]]
```

## Description

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

## Options

### **-u *user\_name***

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

### **-i *flow\_id***

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

### **-c**

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

### **-k**

Specifies to kill the job.

### **-r**

Specifies to run or rerun the job.

### **-p**

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

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

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

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

### **-g**

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

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

### **-l**

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

### **-a**

Specifies that the job to control is a job array.

### ***flow\_name:subflow\_name:job\_name***

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

```
myflow:print:prtreport
```

## Note:

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

For example:

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

#### **-h**

Prints the command usage to `stderr` and exits.

#### **-V**

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

### **Examples**

#### **Kill a specific flow**

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

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

#### **Hold and release a job**

- Hold a job

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

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

- Release the job

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

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

#### **Hold and release a job array**

- Hold a job array

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

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

- Release the job array

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

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

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

- Hold a job in a flow array

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

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

- Release the job in the flow array

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

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

- Hold all jobs in all elements in the flow array

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

- Release all jobs in all elements in the flow array

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

## See Also

**jmanuals**

## jkill

---

kills a flow.

### Synopsis

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

```
jkill flow_id [flow_id ...] | 0
```

```
jkill [-h][[-V]]
```

### Description

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

### Options

#### **-u *user\_name***

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

#### **-f *flow\_name***

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

#### ***flow\_id***

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

#### **0**

Specifies to kill all flows.

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

### Examples

```
jkill -f myflow
```

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

## jlicenseupdate

---

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

### Synopsis

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

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

### Description

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

### Options

**-p**

Updates only the Process Manager license file.

### *license\_file*

Specifies the path to the license file.

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

**-V**

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

**-h**

Prints command usage to `stderr` and exits.

**-V**

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

## jmanuals

---

displays all manual jobs that have not yet been completed.

### Synopsis

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

```
jmanuals [-h] | [-V]
```

## Description

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

## Options

### **-i flow\_ID**

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

### **-u user\_name**

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

### **-f flow\_definition**

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

### **-r yes**

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

### **-r no**

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

### **-h**

Prints the command usage to `stderr` and exits.

### **-V**

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

## See also

**jcomplete**

## jpublish

---

publishes a flow to Process Manager and to other users.

## Synopsis

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

```
jpublish [-h][[-V]]
```

## Description

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

Dynamic subflows and flow arrays can only refer to published target flows.

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

## Options

### **-u user\_name**

Specifies the name of the user who owns the flow.

**-f *flow\_name***

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

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

**Examples**

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

Publishes the flow1 flow belonging to user A.

**See also**

**junpublish**

## jreconfigalarm

---

reloads the alarm definitions.

**Synopsis**

```
jreconfigalarm [-h|-V]
```

**Description**

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

**Options****-h**

Prints the command usage to `stderr` and exits.

**-V**

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

## jrelease

---

releases a previously held flow definition.

**Synopsis**

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

```
jrelease [-h][[-V]]
```

**Description**

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



## Options

### **-u *user\_name***

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

### ***flow\_name***

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

### **-h**

Prints the command usage to `stderr` and exits.

### **-V**

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

## Examples

```
jrelease myflow
```

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

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

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

## See also

### **jhold**

## jremove

---

removes a previously submitted flow definition from Process Manager.

## Synopsis

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

```
jremove [-h][[-V]]
```

## Description

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

## Options

### **-u *user\_name***

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

### **-f**

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

***flow\_name***

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

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

**Examples**

```
jremove myflow
```

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

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

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

**See also**

**jsub, jhold**

## jrerun

---

reruns an exited, done, or running flow.

**Synopsis**

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

```
jrerun [-h][[-V]
```

**Description**

You use the **jrerun** command to rerun a flow. The flow must have a state of Exit, Done, or Running.

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

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

If the flow contains multiple branches, the flow is rerun from the first exited jobs or starting points in each branch and continues to process as designed.

You must be the owner of a flow or a Process Manager administrator to use this command.

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

**Options****-v var=value**

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

***flow\_id***

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

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

## Examples

```
jrerun 1234
```

reruns the flow with the flow ID 1234.

```
jrerun -v "USER=jdoe" 277
```

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

## jresume

---

resumes a suspended flow.

### Synopsis

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

```
jresume flow_id [flow_id ...] | 0
```

```
jresume [-h][[-V]]
```

### Description

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

### Options

#### **-u *user\_name***

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

#### **-f *flow\_name***

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

#### ***flow\_id***

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

#### **0**

Specifies to resume all suspended flows.

**-h**

Prints the command usage to `stderr` and exits.

## **-V**

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

## **Examples**

```
jresume 14 17 22
```

Resumes the flows with IDs 14, 17 and 22.

```
jresume 0
```

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

```
jresume -u all
```

Resumes all suspended flows owned by all users.

## **See also**

## **jstop**

# **jrun**

---

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

## **Synopsis**

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

```
jrun [-h][[-V]]
```

## **Description**

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

## **Options**

### **-v *var=value***

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

### ***flow\_file\_name***

Specifies the name of the file containing the flow definition.

### **-h**

Prints the command usage to `stderr` and exits.

### **-V**

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

## Examples

```
jrun /flows/backup.xml
```

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

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

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

## jsetvars

---

sets values for variables during the runtime of a flow.

### Synopsis

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

### Description

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

### Options

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

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

#### [scope]

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

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

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

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

#### [scope:]

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

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

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

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

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

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

### [scope:]

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

### **var** [[scope:]var ...]

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

```
jsetvars -l
```

List all variables at the global scope.

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

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

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

Removes the specified variables at the global scope. Separate variable names with a space.

### **-h**

Prints the command usage to `stderr` and exits.

### **-V**

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

## Examples

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

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

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

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

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

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

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

- Set variables at different scopes:

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

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

- Set variables for flow arrays:

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

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

- List all variables for a flow:

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

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

- Remove variables at different scopes:

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

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

## jsetversion

---

sets the default version of a flow.

### Synopsis

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

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

### Description

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

### Options

#### **-v *default\_version***

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

#### **-u *user\_name***

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

#### ***flow\_name***

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

#### **-h**

Prints the command usage to `stderr` and exits.

#### **-V**

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

### Examples

```
jsetversion -v 1.3 flow1
```

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

## jsinstall

---

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

### Synopsis

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

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

## Description

**jinstall** runs the Process Manager installation scripts and configuration utilities to install a new Process Manager component. You should install as root.

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

During installation, **jinstall** logs installation progress in the **install.log** file, uncompresses, extracts and copies Process Manager files, and configures Process Manager Server.

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

## Options

### -s

Performs a silent installation of Process Manager.

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

### -y

Accepts the license agreement for the silent installation.

### -f **install.config**

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

Installation parameters are described in the file.

### -h

Prints the command usage to `stderr` and exits.

### -V

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

## jstop

---

suspends a running flow.

## Synopsis

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

```
jstop flow_id [flow_id ...] | 0
```

```
jstop [-h][-V]
```

## Description

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

## Options

### -u **user\_name**

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



name defaults to the user who invoked this command. If you specify `-u all`, and you have administrator authority, you can suspend flows belonging to all users.

**-f *flow\_name***

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

***flow\_id***

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

**0**

Specifies to suspend all flows.

**-h**

Prints the command usage to `stderr` and exits.

**-V**

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

**Examples**

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

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

```
jstop 14
```

Suspends flow ID 14.

```
jstop 0
```

Suspends all flows.

**See also**

**jresume**

## jsub

---

submits a flow definition to Process Manager. This command has been replaced with `jcommit` in version 10.1 but can still be used for compatibility. **jsub** and **jcommit** are aliases.

## jsubmit

---

manually triggers a previously submitted flow definition. Alias for `jtrigger`

## jtrigger

---

manually triggers a previously submitted flow definition.

## Synopsis

`jtrigger` [-u *user\_name*] [-e *version*] [-v "*var=value*[:*var1=value1*[:...]]"] [-f *variable\_file*] *flow\_name*  
*flow\_name...*

`jtrigger -h|-V`

## Description

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

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

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

## Options

### -u *user\_name*

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

### -e *version*

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

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

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

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

### -f *variable\_file*

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

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

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

Example file:

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

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

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

Variable file format:

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

### ***flow\_name***

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

### **-h**

Prints the command usage to `stderr` and exits.

### **-V**

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

## **Examples**

```
jtrigger myflow
```

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

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

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

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

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

## **See also**

**jrun**

## **junpublish**

---

unpublishes a target flow from Process Manager.

### **Synopsis**

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

```
junpublish [-h][[-V]]
```

### **Description**

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

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

### Options

#### **-u *user\_name***

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

#### **-f *flow\_name***

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

#### **-h**

Prints the command usage to `stderr` and exits.

#### **-V**

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

### Examples

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

Unpublishes the flow2 flow belonging to userA.

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

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

### See also

**jpublish**

## licenseinfo

---

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

### Synopsis

```
licenseinfo [-h|-V]
```

### Description

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

### Options

#### **-h**

Prints command usage to `stderr` and exits.

#### **-V**

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

## ppmsetvar

---

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

### Synopsis

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

### Description

This command is installed with IBM Spectrum LSF.

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

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

This is a blocking command.

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

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

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

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

### Options

#### **-f variable\_name=value [variable\_name=value ...]**

Sets user variables for a flow. Use this option to set user variables that can only be accessed by work items within a flow. These user variables cannot be accessed from other flows. Separate multiple variables with a space.

#### **-p variable\_name=value [variable\_name=value ...]**

Sets user variables from a subflow to be used in the parent flow. Use this option when you want to pass a user variable from a subflow to its parent flow. Separate multiple variables with a space.

**-g *variable\_name=value* [*variable\_name=value ...*]**

Sets global user variables for flows. Use this option to set a global user variable that is available to all flows in the system. Separate multiple variables with a space.

**-f -r *variable\_name* [*variable\_name ...*]**

Clears the specified user variable for work items within a flow. Separate multiple variables with a space.

**-p -r *variable\_name* [*variable\_name ...*]**

Clears the specified user variable for the parent subflow. Separate multiple variables with a space.

**-g -r *variable\_name* [*variable\_name ...*]**

Clears global user for flows. Separate multiple variables with a space.

## Return Values

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

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

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

## Clear flow user variables ABC and XYZ

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

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

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

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

## Clear a user variable from a subflow

In this example, the subflow clears the user variable with the subflow short name in its parent subflow. If the subflow name is `Subflow1`, this command clear the user variable `other_result_Subflow1`.

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

## Set a global user variable from any work item

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

## Clear a global user variable from any work item

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

## Notices

---

This information was developed for products and services offered in the U.S.A.

IBM® may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing  
IBM Corporation  
North Castle Drive  
Armonk, NY 10504-1785  
U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
Intellectual Property Law  
Mail Station P300  
2455 South Road,

Poughkeepsie, NY 12601-5400  
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. \_enter the year or years\_.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Trademarks

---

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)® are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>.



Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.



LSF, Platform, and Platform Computing are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Other company, product, or service names may be trademarks or service marks of others.





