Using Platform Process Manager™

Version 3.0
June 8, 2005
Comments to: doc@platform.com
## Using Platform Process Manager

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Process Manager Exceptions and Alarms</td>
<td>95</td>
</tr>
<tr>
<td>5 Running Your Flow</td>
<td>111</td>
</tr>
<tr>
<td>6 Controlling a Flow</td>
<td>135</td>
</tr>
</tbody>
</table>

### 4 Process Manager Exceptions and Alarms

- About Process Manager Exceptions                                     | 96   |
- About Exception Handling                                              | 100  |
- Handling Exceptions                                                   | 104  |
- About Alarms                                                          | 109  |

### 5 Running Your Flow

- About Running Your Flow                                               | 112  |
- Creating a Flow Definition to be Triggered Manually                    | 114  |
- Scheduling Your Flow                                                  | 115  |
- Running a Flow when Another Flow...                                   | 121  |
- Specifying Flow Attributes                                             | 125  |
- About Flow Completion Attributes                                       | 127  |
- Specifying Flow Completion Attributes                                  | 129  |
- Saving the Flow Definition                                            | 131  |
- Running Your Flow Once                                                | 132  |
- Submitting Your Flow Definition                                       | 133  |

### 6 Controlling a Flow

- About the Flow Manager                                                | 136  |
- Filtering the Data Displayed in the Tree View                          | 141  |
- Triggering a Flow                                                     | 144  |
- Viewing a Flow                                                         | 145  |
- Determining the Status of Jobs in a Flow                              | 149  |
- Manually Completing a Dependency                                      | 151  |
- Killing a Running Job                                                 | 152  |
- Running a Single Job                                                  | 153  |
- Marking a Job Complete                                                | 154  |
- Working with Manual Jobs                                              | 155  |
7 Process Manager Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Killing a Running Flow</td>
<td>158</td>
</tr>
<tr>
<td>Suspending a Running Flow</td>
<td>159</td>
</tr>
<tr>
<td>Resuming a Suspended Flow</td>
<td>160</td>
</tr>
<tr>
<td>Rerunning an Exited Flow</td>
<td>161</td>
</tr>
<tr>
<td>Viewing History</td>
<td>162</td>
</tr>
<tr>
<td>Holding a Flow Definition</td>
<td>164</td>
</tr>
<tr>
<td>Releasing a Flow Definition from Hold</td>
<td>165</td>
</tr>
<tr>
<td>Viewing a Flow Definition</td>
<td>166</td>
</tr>
<tr>
<td>Removing a Flow Definition</td>
<td>167</td>
</tr>
</tbody>
</table>

About Process Manager Commands

- caleditor: 172
- floweditor: 173
- flowmanager: 174
- jadmin: 176
- jalarms: 177
- jcadd: 180
- jcals: 185
- jcdel: 187
- jcmd: 188
- jcomplete: 192
- jdefs: 194
- jflows: 197
- jhist: 200
- jhold: 205
- jid: 206
- jjob: 207
- jkill: 209
- jmanuals: 211
- jreconfigalarm: 213
- jrelease: 214
- jremove: 215
- jrerun: 216
- jresume: 217
- jrun: 219
- jsetvars: 220
- jsinstall: 221

Removing a Flow Definition: 158
Viewing a Flow Definition: 159
Resuming a Suspended Flow: 160
Rerunning an Exited Flow: 161
Viewing History: 162
Holding a Flow Definition: 164
Releasing a Flow Definition from Hold: 165
Viewing a Flow Definition: 166
Removing a Flow Definition: 167
<table>
<thead>
<tr>
<th>Command</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>jstop</td>
<td>222</td>
</tr>
<tr>
<td>jsb</td>
<td>224</td>
</tr>
<tr>
<td>jtrigger</td>
<td>231</td>
</tr>
</tbody>
</table>

Index .......................... 233
Welcome

Contents
- “About this Guide” on page 8
- “Learning about Platform Process Manager” on page 9
About this Guide

This is a guide to using Platform Process Manager®. It provides an overview of Process Manager concepts, how to use the Process Manager graphical interfaces to define, schedule, submit, monitor and control your work flows.

Who should use this guide

This guide is written for new Process Manager users who want to familiarize themselves with the fundamentals of creating and managing flows.

What you should already know

This guide assumes that you are familiar with the requirements of the work flows you are defining. You need to know the names of the executables, where they are located, what resources are required by them, and any dependencies they may have.

Typographical Conventions

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Courier</td>
<td>The names of on-screen computer output, commands, files, and directories</td>
<td>The lsid command</td>
</tr>
<tr>
<td><strong>Bold Courier</strong></td>
<td>What you type, exactly as shown</td>
<td><strong>ld</strong></td>
</tr>
<tr>
<td><em>Italics</em></td>
<td>Book titles, new words or terms, or words to be emphasized</td>
<td><em>The queue specified</em></td>
</tr>
<tr>
<td><em>Command-line place holders—replace with a real name or value</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bold Sans Serif</strong></td>
<td>Names of GUI elements that you manipulate</td>
<td><strong>Click OK</strong></td>
</tr>
</tbody>
</table>

Command Notation

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

World Wide Web and FTP


The Platform FTP site (ftp.platform.com) also provides current README files for all supported releases of Process Manager.

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

Platform training

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

Customized on-site course delivery is also available.

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

Process Manager manuals

All of the Process Manager manuals are available in html and PDF format on the Platform Web site at www.platform.com.

Technical support

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

Email support@platform.com

Toll-free phone 1 877 444 4573

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

We’d like to hear from you

If you find an error in any Process Manager documentation, or you have a suggestion for improving it, please let us know. Contact doc@platform.com.
Learning about Platform Process Manager
CHAPTER 1

Introduction to Platform Process Manager

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

Contents
◆ “About Process Manager” on page 12
◆ “About Process Manager Terms” on page 14
◆ “About Flow Definitions and Flows” on page 18
About Process Manager

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

- 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, LSF.

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 Platform Process Manager, and then 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 **Platform Process Manager**, and then **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 **Platform Process Manager**, and then **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 LSF 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 dependencies

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

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

Job arrays

A job array is a group of homogeneous jobs—jobs that share the same executable and resource requirements, but have different input files, for example 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.
Manual jobs

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

Flow definitions

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

Subflows

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

Events

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

**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:
Introduction to Platform Process Manager

- Kill
- Rerun
- Alarms

**Alarms**
An alarm is a type of built-in exception handler, used to send a notification to key personnel that an error has occurred that requires intervention.

**Variables**
You can use Process Manager to pass variables to and from scripts. Process Manager supports two kinds of variables:
- Local variables, which allow you to set a value of a variable and have the value available within a flow;
- Global variables, which allow you to set a value of a variable and have the value available anywhere within the Process Manager Server.

**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.
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:

**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.

**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.

**Submit** The term *submit* is used to describe the act of storing a flow definition in the Process Manager system and placing it under the control of Process Manager. Until a flow definition is submitted, it is not recognized by Process Manager.
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.

**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

**What can I do with a job?**
- Kill a running job
- Hold a waiting job in a running flow
- Rerun a job in a completed flow
- Force a job complete in a completed flow

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

**What makes a flow Done?**
Unless you specify otherwise by defining an exit condition for the flow, 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.
Using Platform Process Manager

CHAPTER 2

Working with 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.

Contents

◆ “About Process Manager Calendars” on page 22
◆ “About the Calendar Editor” on page 24
◆ “Creating a Calendar with Specific Dates” on page 26
◆ “Creating a Calendar using an Expression” on page 27
◆ “Creating a Calendar with a Complex Expression” on page 30
◆ “Calendar Examples” on page 32
◆ “Editing an Existing Calendar” on page 34
◆ “Deleting a Calendar” on page 35
About Process Manager Calendars

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

<table>
<thead>
<tr>
<th>Types of Calendars</th>
<th>Calendar Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weekly calendars</td>
<td>Mondays, Tuesdays, Wednesdays, Thursdays, Fridays, Saturdays, Sundays, Daily, Weekdays, Weekends, Businessdays</td>
</tr>
<tr>
<td>Yearly calendars</td>
<td>Holidays&lt;sup&gt;a&lt;/sup&gt;, First_day_of_year, Last_day_of_year, First_businessday_of_year, Last_businessday_of_year, First_weekday_of_year, Last_weekday_of_year</td>
</tr>
</tbody>
</table>

<sup>a</sup>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 Process Manager 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:

- Open a calendar
- Delete a calendar
- Create a new calendar
- Save the calendar
- Refresh the list of calendars
- View dates the calendar resolves to

About calendar names

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

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

Use this method to create a calendar when:

◆ The calendar needs to be valid for only one or two dates
◆ The calendar needs to be valid for specific, random dates that do not repeat with any pattern

To create a calendar with specific dates:

1. Ensure that Process Manager is running, and open the Calendar Editor.

2. From the Calendar menu, select New Calendar or click ✖️, and select Clicking on Date(s). The date selection dialog box appears. The dialog box is shown here with the list of calendars expanded on the left.

3. In the Description field, specify a description for the calendar that makes it obvious when this calendar is valid. This description is very useful: it is displayed in the fly-over text when you point to a calendar name in the list, and helps you quickly determine which calendar you want to use.

4. Select the dates on which you want this calendar to be valid by left-clicking on each date. You cannot select dates in the past.

5. When you have finished selecting dates, save the calendar: from the Calendar menu, select Save Calendar. When prompted, specify a meaningful name for the calendar. Click Save. The calendar is added to the list of centrally stored calendars, under your user name. For information on naming calendars, see “About calendar names” on page 25.
Creating a Calendar using an Expression

Use this method to create a calendar when:

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

To create a calendar using an expression:

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 \textbf{This Happens …} field, select \textbf{Weekly}. Then specify how frequently this occurs, in the \textbf{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 \textbf{MON} button, then click the \textbf{WED} button, then click the \textbf{FRI} button.

❖ If the expression should be true on specific days every month or every \( n \) months, in the \textbf{This Happens …} field, select \textbf{Monthly}. Then specify how frequently this occurs, in the \textbf{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 \textbf{This Happens …} field, select \textbf{Yearly}. Then specify each date by selecting the month and date and clicking \textbf{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.

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. For information on naming calendars, see “About calendar names” on page 25.
Creating a Calendar with a Complex Expression

Use this method to create a calendar when:

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

To create a calendar with complex expressions:

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” on page 31 for a description of the operators.

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. See “Calendar Examples” on page 32 for complex calendar examples.

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” on page 25.

**Operators and their meanings**

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In common with</td>
<td>Use this operator to resolve to the intersection of two sets of dates—only those dates in common between two calendars</td>
</tr>
<tr>
<td>Plus</td>
<td>Use this operator to combine two sets of dates—any of the dates specified in the two calendars</td>
</tr>
<tr>
<td>Not</td>
<td>Use this operator to exclude the dates in a calendar—use only those dates that are not included in the calendar.</td>
</tr>
<tr>
<td>(</td>
<td>Use this operator to begin an expression nested within the expression</td>
</tr>
<tr>
<td>)</td>
<td>Use this operator to end an expression nested within the expression.</td>
</tr>
<tr>
<td>Not in common with</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

Using Platform Process Manager 31
Calendar Examples

Each of 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.

   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 On the following days is selected, and click Friday, as follows:

   ![Weekly Pattern Selection]

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:

\texttt{last\_friday\_of\_month@user\ In\ common\ with\ businessdays@Sys}

Do this by double-clicking on \texttt{last\_friday\_of\_month}, then double-click on \texttt{In common with}, and then \texttt{businessdays@Sys}.

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’.
Editing an Existing Calendar

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

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

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

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

To edit a calendar to change the dates:

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

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

To edit a calendar to change the pattern:

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

1. In the tree view on the left, double-click on the calendar you want to edit.
2. Ensure you deselect 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 deselect them.
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.

To edit a calendar to change calendar combinations:

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

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

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

To delete a calendar:

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.

If you receive a message that the calendar is in use, do not delete the calendar—remove the flow definition and then delete the calendar. If a calendar is deleted when it is in use, after the Process Manager Server is restarted, the flow definition or flow is moved to the error directory on the Process Manager Server. Contact your administrator for information about it.
Deleting a Calendar
Creating a Flow Definition

This section describes how to create a new flow definition using the Flow Editor.

Contents

◆ “About the Flow Editor” on page 38
◆ “About Creating a Flow Definition—Methods” on page 40
◆ “Using the Example Flows” on page 41
◆ “Creating a Flow Diagram” on page 42
◆ “Including a Job Array in the Flow Diagram” on page 43
◆ “Including a Subflow in the Flow Diagram” on page 46
◆ “Including a Manual Job in the Flow Diagram” on page 47
◆ “About Variables in Process Manager”
◆ “Using User Variables within a Flow Definition” on page 55
◆ “Specifying Job Dependencies” on page 58
◆ “Specifying Dependencies on a File” on page 60
◆ “Changing the Label Displayed for an Event” on page 63
◆ “Specifying Dependencies on a Date and Time” on page 64
◆ “Specifying Dependencies on a Job Array” on page 66
◆ “Specifying Dependencies on a Subflow” on page 68
◆ “Specifying Dependencies on an Unconnected Work Item” on page 70
◆ “Specifying Multiple Dependencies” on page 75
◆ “Specifying the Details of a Job” on page 76
◆ “About Flow Completion Attributes” on page 87
◆ “Specifying Flow Completion Attributes” on page 89
◆ “Specifying a Description for the Flow” on page 92
◆ “Saving the Flow Definition” on page 93
About the Flow Editor

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

About the Flow Editor user interface

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

The Flow Editor toolbar looks like this:

About the design palette

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

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

1: One job at a time

1. Draw the first job on the workspace
2. Define the details of the job
3. Draw the next job
4. Define the details of that job
5. Draw the dependency line between the two jobs
6. Continue with the next job in the flow, and repeat

2: Draw the flow, then fill in the details

1. Draw all of the jobs, flows and events on the workspace
2. Draw the dependency lines between the work items, establishing the relationships between them
3. Define the details for each job and job array, one at a time

3: Combine methods 1 and 2

1. Draw a group of jobs, job arrays and flows on the workspace
2. Draw the dependency lines between those work items
3. Define the details for each job and job array, one at a time.
4. Create another section of the flow

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

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

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

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

The 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:

<table>
<thead>
<tr>
<th>The flow definition named...</th>
<th>Illustrates...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example_1</td>
<td>A simple job dependency, where one job runs when its predecessor completes successfully</td>
</tr>
<tr>
<td>Example_2</td>
<td>The use of a recovery job</td>
</tr>
<tr>
<td>Example_3</td>
<td>The use of a manual task and a job array</td>
</tr>
<tr>
<td>Example_4</td>
<td>The use of a file event to trigger some processing</td>
</tr>
<tr>
<td>Example_5</td>
<td>The use of a time event to run a job array, which raises and alarm if it misses its schedule(^a)</td>
</tr>
</tbody>
</table>

\(^a\)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.

To view the sample flow definitions:

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.

To use a sample flow definition:

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

To create a simple flow diagram:

1. Click the Job button \( \text{Job} \) 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 \( \text{Dependency} \) 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:

   ![Flow Diagram Example](image)

   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.

   ![Edit Job Dialog](image)

   In the remaining input fields and tabs, specify any other details required to define the job. For more information on specifying job details, see “Specifying the Details of a Job” on page 76.
7. Save the flow definition. You can save it in your local file system or in a shared file location.
Including a Job Array in the Flow Diagram

You can include a job array in the flow diagram. Using a job array is a convenient way to specify a group of jobs that share the same executable and resource requirements, but use different input data, with a single definition.

All jobs in a job array have the same name and same job ID. Each job runs the same executable. Any parameters you specify apply to all jobs in the array. All jobs use an input file from the same location, and write to the same output file location. However, each element of a job array is distinguished by its array index.

Before you can use a job array, you need to prepare your input files. See “Job Arrays” chapter in Administering Platform LSF for more information.

To insert a job array:

1. Click the 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.

To define job array details:

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

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
      \[1-5\]
      In this example, the job array will consist of five jobs.
      The index expression can also be in the format
      \[start-end[ : step]\]
Including a Job Array in the Flow Diagram

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

\textbf{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—\texttt{start}, \texttt{end} or \texttt{step}. For example:

\textbf{1-#{COUNT}}

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

Using the Job Array Index Expression Builder dialog.

Click the \textbf{...} button next to the \textbf{Index expression} field. The Job Array Index Expression Builder dialog appears.
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}`

iii In the **End** field, specify the end of the range of indices. For example: 10. Or specify a variable. For example: `{endix}`

iv Optional. In the **Step** field, specify the value to increment the indices in the range. The default is a step of 1. For example: 2. Or specify a variable. For example: `{stepix}`

v Click **Add** to add the expression to the list.

vi Repeat steps ii through v until you have completed specifying all the index ranges.

vii 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 Specify any additional options on this tab and the other definition tabs. For detailed information about each of the options, see “Specifying the Details of a Job” on page 76.

9 Click **OK**.
Including a Subflow in the Flow Diagram

At any time, you can include an existing flow definition as a 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.

To insert a subflow:

1. Click the Subflow button to put the Flow Editor in 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 subflow. The subflow icon is added to the flow diagram.
4. Draw the lines describing any dependencies the subflow has on other work items in the flow definition.
5. If you want to include a description or instructions regarding the subflow, right-click on the subflow, and select Flow Attributes. The Subflow Attributes dialog appears.

Specify the description in the field provided. Click OK.

6. If you want to monitor this subflow for a particular exception, and handle it automatically, open the Subflow Attributes dialog by right-clicking on the subflow, and selecting Flow Attributes. Click Exception Handling and specify the exceptions and handlers. Then click OK. For more detailed information about exceptions and handlers, see “Handling Exceptions” on page 104.

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

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

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

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

To insert a manual job:

1. Click the 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.

To define manual job details:


2. In the Job name field, specify a unique, meaningful name for the manual job. You will use this name when completing the job as the flow runs. You can use alphabetic characters, numerals 0 to 9, period (.), dash (-) and underscore (_) in the manual job name. The Flow Editor assigns a unique name to each manual job when you draw
it on the workspace, so you are not required to change the name. However, if you want to change the name, you can. The name itself is required.

3 In the **Email address** field, specify the email address to notify when the job is ready for completion. If you do not change the email address, it defaults to your user ID. If you delete the email address, no notification is sent when the job requires completion.

4 In the **Message** field, specify the message text that should appear in the email notification. For example, if the manual job will be used to verify a report, you might include the following as the message text:

   **Verify the output of report paylist.**

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

   **Check output from printer #{PRINT}**.

5 In the **Description** field, add any descriptive text that may be used for managing this job within the flow. For example, if this job requires special instructions for operations staff, place those instructions here.
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, or in the message sent when a manual job requires completion.

Types of variables

Process Manager supports two types of variables:

- User variables
- Built-in variables

**User variables** User variables are those defined by the user, where the value is set within a job, and made available to Process Manager.

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

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

- %I
- %J
- FILEEVENT[n]_NAME
- JS_FLOW_ID
- JS_FLOW_NAME
- JS_ITERATION_COUNTER

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

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

Specify the variable as follows:

`file_name.%I`

Do not specify brace brackets and # sign.

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

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

Specify the variable as follows:

`file_name.%J`

Do not specify # sign and brace brackets.

FILEEVENT[n]_NAME
You use the built-in variable FILEEVENT[n]_NAME when you need to use the name of the file that triggered this particular flow. If a flow is triggered by multiple files, multiple variables are created, each with a different value for n. The value of n is determined by the position of the triggering event in the list of possible events. Separate a list of variables with commas, semicolons or spaces.

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

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

`#{FILEEVENT[n]_NAME}`

where n is the position of the triggering event in the list of possible events.
Example A: one event at a time

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

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

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

FILEEVENT[2]_NAME=/tmp/payupdt

If the flow is triggered by the time event, no value is set for FILEEVENT[2]_NAME.

Note that the value of $n$ in the name of the variable corresponds to the position of the file event in the list of events.
**Example B:** multiple events

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

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

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:

FILEEVENT[1].NAME=/tmp/payupdt
FILEEVENT[3].NAME=/tmp/dbupdt

Note that the value of \( n \) in the name of the variable corresponds to the position of the file event in the list of events.
Example C: any tar file arrives

In this example, myflow is triggered when any tar file arrives:

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

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

\#{JS_FLOW_ID}

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

`flow_ID:flow_name:user_name`

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

\#{JS_FLOW_NAME}

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 rerun. The first time the flow or subflow runs, the JS_ITERATION_COUNTER is 0. If the flow or subflow is rerun, the counter increases. For example, if the JS_ITERATION_COUNTER is 3 it means the flow or subflow has rerun 3 times; it is running for the 4th time overall.
**Usage**  Specify the variable as follows when you want to use its current value:

`#{JS_ITERATION_COUNTER}`
Creating a Flow Definition

Using User Variables within a Flow Definition

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.

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

Types of user variables you can set

There are two types of variables you can set:
- Local variables—those whose values are available only to jobs, job arrays, subflows or events within the current flow
- Global variables—those whose values are available to all the flows within the Process Manager Server

Setting the value of a user variable

You can set the value of one or more variables as follows:
- On UNIX, by setting the value within a script
- On Windows, by setting the value within a bat file

Multiple variables in a list

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

To set a user variable in a UNIX script:

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 a local variable, whose value is not available outside the scope of this flow (or subflow), specify the following:

      \texttt{JS\_FLOW\_VARIABLE\_LIST=\textasciitilde var=value;var1=value1;}}

      For example:

      \texttt{
      
      #!/bin/sh
todays_date\textasciitilde \textasciitilde var=`date +%Y%m%d'
user_name=`logname'
JS\_FLOW\_VARIABLE\_LIST="TDATE=$todays_date;USER=$user_name"
export JS\_FLOW\_VARIABLE\_LIST
echo $JS\_FLOW\_VARIABLE\_LIST
      }

   b. To set a global variable, whose value is available to all flows within the Process Manager Server, specify the following:

      \texttt{JS\_GLOBAL\_VARIABLE\_LIST=\textasciitilde var=value;var1=value1;}}

      For example:
Using User Variables within a Flow Definition

```
#!/bin/sh
JS_GLOBAL_VARIABLE_LIST="execPATH=/home/admin1/executables"
export JS_GLOBAL_VARIABLE_LIST
```

3 Export the variable (make the value available) as follows:

```
export JS_FLOW_VARIABLE_LIST
```

or

```
export JS_GLOBAL_VARIABLE_LIST
```

The script that sets your variables cannot call exit—the variables will not be set.

To set a user variable in a Windows bat file:

1 Within the batch file, set the scope of the variable by specifying which list of variables to create, as follows:

   a To set a local variable, whose value is not available outside the scope of this flow (or subflow), specify the following:

   ```
   set JS_FLOW_VARIABLE_LIST="var=value[;var1=value1...]"
   ```

   For example:

   ```
   set JS_FLOW_VARIABLE_LIST="custno=12345;loc=Toronto"
   ```

   b To set a global variable, whose value is available to all flows within the Process Manager Server, specify the following:

   ```
   set JS_GLOBAL_VARIABLE_LIST="var=value[;var1=value1...]"
   ```

   For example:

   ```
   set JS_GLOBAL_VARIABLE_LIST="NTADMIN=bsmith"
   ```

Using the value of a variable

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

- On the Job/Job Array Definition—Edit Job dialogs, General tab, in the following fields:
  - Name—applies only to jobs, not job arrays
  - Command to run
  - Index expression—applies only to job arrays
  - Input file
  - Output file
  - Error file
- 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
Creating a Flow Definition

- On the Alarm Definition dialog, in the following field:
  - Description

**To use a variable:**

1. Ensure that a value for the variable is set, either within the current flow, or as a global variable within the Process Manager Server, prior to the point in the flow where the value is required.

2. In the appropriate input field, specify the variable in the following format:
   
   ```
   #{variable}
   ```

   For example:

<table>
<thead>
<tr>
<th>Define the Job</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: J1</td>
</tr>
<tr>
<td>Command to run: #execPATH/paydata</td>
</tr>
</tbody>
</table>

**When the value of a variable is evaluated**

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

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

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

You can choose from the following criteria:

- Run a job when the predecessor completes successfully
- Run a job when the predecessor job starts
- Run a job when the predecessor 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

To create a job dependency:

1. Draw both the predecessor job and the job that succeeds it.
2. Change to job dependency mode by clicking the 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.

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

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.

Click **OK**.

**Example: run a job when predecessor starts**

| Job name: | J1 |
| Event type: | Starts |

**Example: run when predecessor has exit code greater than 2**

| Job name: | J1 |
| Event type: | Ends with exit code Greater than 2 |
Specifying Dependencies on a File

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

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

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

To specify a dependency on a file:

1. Change to file event mode by clicking the button.
2. Click in the workspace where you want to insert the file event. The file event icon does not yet appear in the workspace. The Event Definition dialog box appears.
3. In the **File name** field, specify the full path name of the file as the 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 Saturdays.tar and Sundays.dat. *e matches smile.

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

   You can also specify a variable for the file name, provided your system is configured to support them. See “Using User Variables within a Flow Definition” on page 55 for more information about user variables.

4. In the **Condition** field, choose the appropriate condition from the list, and specify the number of bytes if applicable. See the examples that follow this topic.

5. In the **Description** field, add any descriptive text that may be used for understanding this event. For example, if this event requires special instructions for operations staff, place those instructions here.
Creating a Flow Definition

Using Platform Process Manager

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.

You can change the text of the label that appears above the file event in the workspace if the label text is too long. See “Changing the Label Displayed for an Event” on page 63 for instructions.

**Example: run a job when a file arrives**
Specifying Dependencies on a File

Example: run a job if file is updated within a certain time period

![File Event Definition](image)

**File name:** /tmp/cieta

**Condition:**

- **Age:** greater than
- **16 minutes**

**Description:** This file should be at least this old. If not, contact Marie.
Changing the Label Displayed for an Event

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

```
last_businessday_of_month, 2,3,5,6-9,11,12,14-17,19,20,22-24: 45
```

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.

To change an event label:

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.
Specifying Dependencies on a Date and Time

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

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

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

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

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

To control the time when a job runs:

1. Change to time dependency mode by clicking the 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 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. For the syntax of the time expression, see “Scheduling Your Flow” on page 115.
5. 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.
6. 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. 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**. 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.

You can change the text of the label that appears above the file event in the workspace if the label text is too long. See “Changing the Label Displayed for an Event” on page 63 for instructions.
Specifying Dependencies on a Job Array

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

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

To specify a dependency on a job array:

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

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

**Example: all jobs in the array end, regardless of success or failure**

<table>
<thead>
<tr>
<th>Job array name:</th>
<th>A1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event type:</td>
<td>All jobs end with any exit code</td>
</tr>
</tbody>
</table>

**Example: sum of the exit codes is...**

<table>
<thead>
<tr>
<th>Job array name:</th>
<th>A1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event type:</td>
<td>The sum of the exit codes of all jobs is...</td>
</tr>
<tr>
<td></td>
<td>Greater than 5</td>
</tr>
</tbody>
</table>

**Example: number of successful jobs is...**

<table>
<thead>
<tr>
<th>Job array name:</th>
<th>A1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event type:</td>
<td>Number of successful jobs is...</td>
</tr>
<tr>
<td></td>
<td>Equal to 5</td>
</tr>
</tbody>
</table>

**Example: number of unsuccessful jobs is...**

<table>
<thead>
<tr>
<th>Job array name:</th>
<th>A1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event type:</td>
<td>Number of unsuccessful jobs is...</td>
</tr>
<tr>
<td></td>
<td>Equal to 2</td>
</tr>
</tbody>
</table>

**Example: number of jobs started is...**

<table>
<thead>
<tr>
<th>Job array name:</th>
<th>A1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event type:</td>
<td>Number of jobs started is...</td>
</tr>
<tr>
<td></td>
<td>Greater than or equal to 4</td>
</tr>
</tbody>
</table>
Specifying Dependencies on a Subflow

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

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

- All jobs in the subflow complete successfully—the default
- All jobs in the subflow end, regardless of the exit code
- The sum of the exit codes of all the jobs in the subflow has a value
- A specified number of jobs in the subflow complete successfully
- A specified number of jobs in the subflow fail
- A specified number of jobs in the subflow end regardless of success or failure
- A specified number of jobs in the subflow have started
- The subflow runs longer than it should
- The subflow runs for an abnormally short length of time
- The subflow misses its schedule

To specify a subflow dependency:

1. Draw both the predecessor subflow and the work item that succeeds it.
2. Change to job dependency mode by clicking the 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**.

**Example: sum of the exit codes has a specific value**

| Flow name: | myflow |
| Event type: | The flow has exit code... |
| operator | Less than or equal to |
| value | 2 |

**Example: specified number of jobs complete successfully**

| Flow name: | myflow |
| Event type: | Number of successful jobs is... |
| operator | Greater than |
| value | 2 |

**Example: specified number of jobs fails**

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

| Flow name: | myflow |
| Event type: | Number of unsuccessful jobs is... |
| operator | Greater than |
| value | 2 |

**Example: specified number of jobs have started**

| Flow name: | myflow |
| Event type: | Number of jobs started is... |
| operator | Greater than |
| value | 5 |
Specifying Dependencies on an Unconnected Work Item

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 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 a specific number of jobs in a job array or subflow starts
- When a specific number of jobs in a job array or subflow ends
- When a specific number of jobs in a job array or subflow exits

To specify a dependency on a proxy job:

1. Change to proxy event mode by clicking the 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:
   \( \text{flow_name:subflow_name:job_name} \)
   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 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.
7. In the Description field, add any descriptive text that may be used for understanding this event.
Creating a Flow Definition

Using Platform Process Manager

71

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

To specify a dependency on a proxy job array:

1. Change to proxy event mode by clicking the button.

2. Click in the workspace where you want to insert the proxy event. The proxy event icon does not yet appear in the workspace. The Proxy Event Definition dialog box appears.

3. In the Create proxy for... box, select Job Array.

4. In the Job array name field, specify the fully qualified name of the job array, in the following format:

   \textit{flow\_name:subflow\_name:job\_array\_name}

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

5. If the flow containing the job array is not owned by your user ID, in the Owner field, specify the user ID that owns the flow containing the proxy job array.

6. In the 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.

7. In the Description field, add any descriptive text that may be used for understanding this event.
Specifying Dependencies on an Unconnected Work Item

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

To specify a dependency on a proxy subflow:

1. Change to proxy event mode by clicking the 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 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.
7. In the Description field, add any descriptive text that may be used for understanding this event.
Click **OK**. The proxy event appears in the workspace, and you can draw the appropriate dependency lines to any work items.

**To specify a dependency on a proxy flow:**

1. Change to proxy event mode by clicking the 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 **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.
7. In the **Description** field, add any descriptive text that may be used for understanding this event.
Specifying Dependencies on an Unconnected Work Item

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

You can change the text of the label that appears above the proxy event in the workspace if the label text is too long. See “Changing the Label Displayed for an Event” on page 63 for instructions.
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.

To specify that all dependencies must be met:

1. Change to AND link mode by clicking the \( \text{AND} \) button.
2. Click in the workspace where you want to insert the AND link.
3. Change to job dependency mode by clicking the \( \text{Job Dependency} \) 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.

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.

To specify that at least one dependency must be met:

1. Change to OR link mode by clicking the \( \text{OR} \) button.
2. Click in the workspace where you want to insert the OR link.
3. Change to job dependency mode by clicking the \( \text{Job Dependency} \) 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. For more information on exit conditions, see “About Flow Completion Attributes” on page 87.

You can change an AND link into an OR link by double-clicking on the AND icon.
Specifying the Details of a Job

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

The General tab:

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.
Creating a Flow Definition

Specifying the command the job runs
The purpose of a job is to run a command, so a command name is mandatory. In the **Command to run** field, specify the name of the command this job runs, and any arguments required by the command, ensuring the syntax of the command is correct, or specify the script to run. Because the job will run under your user ID, ensure the path to the script is specified in your path environment variable, or specify the full path to the script.

If running this command or script requires access to a file or application, ensure the files are in a shared location that is accessible to the 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.

Specifying 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.

Running 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.

Submitting 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 commas.

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.

Submitting 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.
Specifying the Details of a Job

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, in the **Submit** box, check **On hold**.

**Specifying 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/.

**Notifying 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.

**Running 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 Processing tab:

Running 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.

Running 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.

Rerunning 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 **Run on another host if host becomes unavailable**.
Specifying the Details of a Job

Running 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.

Specifying 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.

Assigning 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.

Running 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.

Assigning 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 Fairshare group field, specify the name of the group.
Creating a Flow Definition

The Resources tab:

Specifying resources required to run the job

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

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

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

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

Floating 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.
Specifying the Details of a Job

**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:

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

**The Limits tab:**

<table>
<thead>
<tr>
<th>Job Definition - Edit job</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Limits</strong></td>
</tr>
<tr>
<td><strong>Host Limits</strong></td>
</tr>
<tr>
<td>Host name or model</td>
</tr>
<tr>
<td>Maximum CPU time: hours and minutes</td>
</tr>
<tr>
<td>Maximum run time: hours and minutes</td>
</tr>
</tbody>
</table>

**Specifying 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.

**Specifying 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:

![File Transfer Tab](image)

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.

**Transferring a local file required by the job**

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

1. In the **Local path including name** field, specify the full path name of the file to be transferred.
2. If the location on the host where the job will run is different from the local path, in the **File on execution host** field, specify the full path where the file should be located when the job runs.
3. Select **Copy file to remote host before running job**.
4. Click **Add** to add this operation to the list of operations to perform.
5. Repeat as required.
Transferring an output file locally after the job runs

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

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

Appending output to a local file after the job runs

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

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

<table>
<thead>
<tr>
<th>In a...</th>
<th>If this exception occurs...</th>
<th>You can use this handler...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job</td>
<td>Overrun</td>
<td>Kill</td>
</tr>
<tr>
<td></td>
<td>Underrun</td>
<td>Rerun</td>
</tr>
<tr>
<td></td>
<td>Specified exit code</td>
<td>Rerun</td>
</tr>
<tr>
<td>Job array</td>
<td>Overrun</td>
<td>Kill</td>
</tr>
<tr>
<td></td>
<td>Underrun</td>
<td>Rerun</td>
</tr>
<tr>
<td></td>
<td>Sum of exit codes</td>
<td>Rerun</td>
</tr>
<tr>
<td></td>
<td>Number of unsuccessful jobs</td>
<td>Kill</td>
</tr>
</tbody>
</table>

For more information about exceptions, exception handlers and alarms, see “Process Manager Exceptions and Alarms” on page 95.
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.
About Flow 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.

Alternative completion criteria

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

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

Default completion behavior of a flow

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

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

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

**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:

![Flow diagram]

**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.
Specifying 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:

- All work completes successfully or any work item fails or is killed. 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.
- Continue running the flow until any remaining work items that can complete, complete.

To assign a state to a flow when all work items are Done:

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

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

3. If you want the flow to stop running if any work item exits or is killed, leave the second option at the default Complete any work in progress and stop running the flow.
Specifying Flow Completion Attributes

running the flow. If you want to continue to process as many jobs in the flow as possible, click **Continue running the flow**.

4 Click **OK**. The flow will be assigned a state when all of the work items complete successfully or any work item fails or is killed.

To assign a state to a flow when all selected work items end:

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

2 Select **All selected work items end**.

3 From the list of available work items, select those that must process before the flow can be assigned a state. Select each item and click **Add** to move it to the list of selected items, or double-click on an item to move it to the other list.

4 If you want the flow to stop running when the specified jobs end, leave the second option at the default **Complete any work in progress and stop running the flow**. If you want to continue to process as many jobs in the flow as possible, click **Continue running the flow**.

5 Click **OK**. The flow will be assigned a state when all of the selected work items end.

To assign a state to a flow when any selected work item ends:

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

2 Select **Any selected work items end**.

3 From the list of available work items, select those that may process before the flow can be assigned a state. Select each item and click **Add** to move it to the list of selected items, or double-click on an item to move it to the other list. When one item in this list ends, the flow will be assigned a state.

4 If you want the flow to stop running when one of the specified jobs end, leave the second option at the default **Complete any work in progress and stop running the flow**. If you want to continue to process as many jobs in the flow as possible, click **Continue running the flow**.

5 Click **OK**. The flow will be considered complete when one of the selected work items ends.

To continue processing when the state of the flow is determined:

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

2 Select **Continue running the flow**.

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.
Specifying Exception Handling for a Flow

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

To specify exception handling for a flow:

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:

<table>
<thead>
<tr>
<th>If you monitor for this exception...</th>
<th>This handler is used...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overrun</td>
<td>Kill</td>
</tr>
<tr>
<td>Underrun</td>
<td>Rerun</td>
</tr>
<tr>
<td>Exit code</td>
<td>Rerun</td>
</tr>
<tr>
<td>Number of unsuccessful jobs</td>
<td>Kill</td>
</tr>
</tbody>
</table>

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.
Specifying a Description for the Flow

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.

To add a description to the flow:

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 tab.

2. On the General tab, enter the description text in the field provided. When you have finished typing the description, click OK.
Creating a Flow Definition

Saving 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.
Saving the Flow Definition
This section describes Process Manager exceptions, built-in exception handlers you can use when they occur, and how to use Process Manager alarms.

Contents
◆ “About Process Manager Exceptions” on page 96
◆ “About Exception Handling” on page 100
◆ “Handling Exceptions” on page 104
◆ “About Alarms” on page 109
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.

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:

<table>
<thead>
<tr>
<th>When a ...</th>
<th>Experiences this exception</th>
<th>This happens ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow definition</td>
<td>Misschedule</td>
<td>The flow is not triggered.</td>
</tr>
<tr>
<td>Flow</td>
<td>Overrun</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Underrun</td>
<td>The time is calculated from when the flow is first triggered until its status changes from Running to Exit or Done.</td>
</tr>
<tr>
<td>Subflow</td>
<td>Misschedule</td>
<td>The subflow is not run.</td>
</tr>
<tr>
<td></td>
<td>Overrun</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Underrun</td>
<td>The time is calculated from when the subflow first starts running until its status changes from Running to Exit or Done.</td>
</tr>
<tr>
<td>Job</td>
<td>Misschedule</td>
<td>The job is not run.</td>
</tr>
<tr>
<td></td>
<td>Cannot Run</td>
<td>The job is not run.</td>
</tr>
<tr>
<td></td>
<td>Start Failed</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Overrun</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Underrun</td>
<td>The time is calculated from the when the job is successfully submitted until it reaches Exit or Done state.</td>
</tr>
<tr>
<td>Job array</td>
<td>Misschedule</td>
<td>The job array is not run.</td>
</tr>
<tr>
<td></td>
<td>Cannot Run</td>
<td>The job array is not run.</td>
</tr>
<tr>
<td></td>
<td>Start Failed</td>
<td>The job array is still waiting. Submission of the job array is retried until it runs.</td>
</tr>
<tr>
<td></td>
<td>Overrun</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Underrun</td>
<td>The time is calculated from when the job array is successfully submitted until all elements in the array reach Exit or Done state.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Work Item</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow</td>
<td>An exit code of ( n ) (sum of all exit codes)( n ) unsuccessful jobs</td>
</tr>
<tr>
<td>Subflow</td>
<td>An exit code of ( n )</td>
</tr>
<tr>
<td></td>
<td>( n ) unsuccessful jobs</td>
</tr>
<tr>
<td>Job</td>
<td>An exit code of ( n )</td>
</tr>
<tr>
<td>Job array</td>
<td>An exit code of ( n )</td>
</tr>
<tr>
<td></td>
<td>( n ) unsuccessful jobs</td>
</tr>
</tbody>
</table>
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 both a visual, graphical cue that an exception has occurred, and a notification to one or more email addresses. You use an alarm to notify key personnel, such as database administrators, of problems that require attention. An alarm has no effect on the flow itself.

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. See “To handle exceptions with a recovery job:” on page 106 for instructions.

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. See “To handle exceptions with a recovery flow:” on page 106 for instructions.

Behavior when exception handlers are used

The following describes Process Manager behavior when an exception handler is used:

<table>
<thead>
<tr>
<th>When a ...</th>
<th>Experiences this Exception ...</th>
<th>and the Handler Used is ...</th>
<th>This Happens ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow</td>
<td>Overrun</td>
<td>Kill</td>
<td>The flow is killed. All incomplete jobs in the flow are killed. The flow status is ‘Killed’</td>
</tr>
<tr>
<td></td>
<td>Underrun</td>
<td>Rerun</td>
<td>Flows that have a dependency on this flow may not be triggered, depending on the type of dependency. The flow is recreated with the same flow ID. The flow is rerun from the first job, as many times as required until the execution time exceeds the underrun time specified.</td>
</tr>
<tr>
<td>An exit code of $n$</td>
<td>Rerun</td>
<td>Flows that have a dependency on this flow may not be triggered, depending on the type of dependency. The flow is recreated with the same flow ID. The flow is rerun from the first job, as many times as required until an exit code other than $n$ is reached.</td>
<td></td>
</tr>
<tr>
<td>$n$ unsuccessful jobs</td>
<td>Kill</td>
<td>The flow is killed. All incomplete jobs in the flow are killed. The flow status is ‘Killed’</td>
<td></td>
</tr>
<tr>
<td>When a ...</td>
<td>Exceptions this Exception ...</td>
<td>and the Handler Used is ...</td>
<td>This Happens ...</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------</td>
<td>-----------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Subflow</td>
<td>Misschedule</td>
<td>Alarm</td>
<td>The alarm is opened. The subflow is not run. The flow continues execution as designed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recovery job or flow</td>
<td>The subflow is not run. The flow continues execution as designed. The recovery job or flow is triggered.</td>
</tr>
<tr>
<td>Overrun</td>
<td>Alarm</td>
<td></td>
<td>The alarm is opened. Both the flow and subflow continue execution as designed.</td>
</tr>
<tr>
<td></td>
<td>Recovery job or flow</td>
<td></td>
<td>Both the flow and subflow continue execution as designed. The recovery job or flow is triggered.</td>
</tr>
<tr>
<td>Kill</td>
<td>Alarm</td>
<td></td>
<td>The subflow is killed. The flow behaves as designed.</td>
</tr>
<tr>
<td>Underrun</td>
<td>Alarm</td>
<td></td>
<td>The alarm is opened. The flow continues execution as designed.</td>
</tr>
<tr>
<td></td>
<td>Recovery job or flow</td>
<td></td>
<td>The subflow continues execution as designed. The recovery job or flow is triggered.</td>
</tr>
<tr>
<td>Rerun</td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>An exit code of $n$</td>
<td>Rerun</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>$n$ unsuccessful jobs</td>
<td>Kill</td>
<td>The subflow is killed. The flow behaves as designed.</td>
</tr>
<tr>
<td>When a ...</td>
<td>Experiences this Exception ...</td>
<td>and the Handler Used is ...</td>
<td>This Happens ...</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------</td>
<td>-----------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Job or job array</td>
<td>Misschedule</td>
<td>Alarm</td>
<td>The alarm is opened. The job or job array is not run. The flow continues execution as designed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recovery job or flow</td>
<td>The job or job array is not run. The flow continues execution as designed. The recovery job or flow is triggered.</td>
</tr>
<tr>
<td></td>
<td>Overrun</td>
<td>Alarm</td>
<td>The alarm is opened. Both the flow and job or job array continue to execute as designed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Recovery job or flow</td>
<td>Both the flow and job or job array continue to execute as designed. The recovery job or flow is triggered.</td>
</tr>
<tr>
<td></td>
<td>Kill</td>
<td></td>
<td>The job or job array is killed. The flow behaves as designed. The job or job array status is determined by its exit value.</td>
</tr>
<tr>
<td>Underrun</td>
<td>Alarm</td>
<td></td>
<td>The alarm is opened. The flow continues execution as designed.</td>
</tr>
<tr>
<td></td>
<td>Recovery job or flow</td>
<td></td>
<td>The flow continues execution as designed. The recovery job or flow is triggered.</td>
</tr>
<tr>
<td></td>
<td>Rerun</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Start Failed</td>
<td>Alarm</td>
<td></td>
<td>The alarm is opened. The flow continues execution as designed.</td>
</tr>
<tr>
<td></td>
<td>Recovery job or flow</td>
<td></td>
<td>The recovery job or flow is triggered.</td>
</tr>
<tr>
<td></td>
<td>Rerun</td>
<td></td>
<td>The job or job array is rerun as many times as required until it starts successfully.</td>
</tr>
<tr>
<td>Cannot Run</td>
<td>Alarm</td>
<td></td>
<td>The alarm is opened. The flow continues execution as designed.</td>
</tr>
<tr>
<td></td>
<td>Recovery job or flow</td>
<td></td>
<td>The recovery job or flow is triggered.</td>
</tr>
<tr>
<td>An exit code of $n$</td>
<td>Rerun</td>
<td></td>
<td>The job or job array is rerun as many times as required until it starts successfully.</td>
</tr>
<tr>
<td>$n$ unsuccessful jobs</td>
<td>Kill</td>
<td></td>
<td>The job array is killed. The flow behaves as designed. The job array status is determined by its exit value.</td>
</tr>
</tbody>
</table>
Handling Exceptions

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:

- Built-in exception handler
- Recovery job
- Recovery flow

To handle exceptions of a job or job array using built-in handlers:

1. Within the flow definition in the Flow Editor, open the job or job array definition.
2. Click on the Exception Handling tab.
3. Click Add. The Exception Handler Definition dialog appears.
4. In the Exception type field, select the exception you want to handle.
5. If you chose Runs more than..., specify the maximum time, in minutes, the job or job array can run before it should be killed.
   If you chose Runs less than..., specify the minimum time, in minutes, the job or job array can run before it should be rerun.
   If you chose Has exit code, choose the operator and value that best define the exit code requirement. For example, greater than 5.
   If you chose Number of unsuccessful jobs is … choose the operator and value that best define the exit code requirement. For example, greater than 3.
6. In the Action field, select the appropriate exception handler. In most cases, however, the appropriate exception handler is selected for you, as follows:

<table>
<thead>
<tr>
<th>If you monitor for this exception...</th>
<th>This handler is used...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overrun</td>
<td>Kill</td>
</tr>
<tr>
<td>Underrun</td>
<td>Rerun</td>
</tr>
</tbody>
</table>
If you specify a rerun exception, you can specify a number of minutes to delay before rerunning the subflow.

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

8 Repeat steps 3 through 7 until you have finished specifying exceptions to handle then click OK.

To handle exceptions of a subflow using built-in handlers:

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

5 In the Exception type field, select the exception you want to handle.
6 If you chose Runs more than..., specify the maximum time, in minutes, the subflow can run before it should be killed.
   If you chose Runs less than..., specify the minimum time, in minutes, the subflow can run before it should be rerun.
   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.

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

<table>
<thead>
<tr>
<th>If you monitor for this exception...</th>
<th>This handler is used...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overrun</td>
<td>Kill</td>
</tr>
<tr>
<td>Underrun</td>
<td>Rerun</td>
</tr>
</tbody>
</table>
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, as shown:

<table>
<thead>
<tr>
<th>If you monitor for this exception...</th>
<th>This handler is used...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit code</td>
<td>Rerun</td>
</tr>
<tr>
<td>Number of unsuccessful jobs</td>
<td>Rerun</td>
</tr>
</tbody>
</table>

To handle exceptions with a recovery job:

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 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.

To handle exceptions with a recovery flow:

1. In the Flow Editor, define the recovery flow such that it performs the required functions.
2. When the flow definition is complete, from the Action menu, select Submit. The Submit Flow dialog box appears.
3. In the This Flow will be Triggered ... box, select By one or more events and click OK. The Flow-Triggering Events dialog appears.
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.
About 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.

To raise an alarm when an exception occurs within a flow:

1. In the Flow Editor, with the flow definition opened, change to alarm mode by clicking the \( \text{\textbullet} \) 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**.

**To view the opened alarms:**

In the Flow Manager, from the **View** menu, select **Alarms**. The View Alarms dialog box appears. It lists all of the open alarms.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Trigger Time</th>
<th>Notification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarms sample Alarm1</td>
<td>IBS</td>
<td>Thu Nov 14 13:36:55 2002</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
Running Your Flow

The attributes of a flow include what, if any, events trigger the flow to run, what constitutes successful completion of the flow, the type of email notification to implement regarding the flow, which flow exceptions to monitor for, and what to do if they occur.

Contents

- “About Running Your Flow” on page 112
- “Creating a Flow Definition to be Triggered Manually” on page 114
- “Scheduling Your Flow” on page 115
- “Running a Flow when Another Flow...” on page 121
- “Specifying Flow Attributes” on page 125
- “About Flow Completion Attributes” on page 127
- “Specifying Flow Completion Attributes” on page 129
- “Saving the Flow Definition” on page 131
- “Running Your Flow Once” on page 132
- “Submitting Your Flow Definition” on page 133
About Running Your Flow

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 LSF.

The first decision you need to make is how the flow will be triggered. (*Triggering a flow* is the act of telling LSF 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” on page 114 for instructions.

If you can specify a recurring schedule for the flow, see “To run a flow at a specific time:” on page 115 for instructions.

If you want to run a flow whenever something happens to a particular file, see “To run a flow based on file activity:” on page 118 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” on page 132 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. For more information on passing values to the flow when triggering it, see “To trigger a flow, passing it values for variables:” on page 144.

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 LSF, 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
Running Your Flow

Using Platform Process Manager

- 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.
Creating 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.

To specify that the flow is to be triggered manually:

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 see “Triggering a Flow” on page 144 for instructions.

From the command line
1. On the command line, type the following:
   
   jsub flow_file_name
   
   where flow_file_name is the full path name of the file containing the flow definition.
2. Press Enter.
Scheduling 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.

To run a flow at a specific time:

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 appears.
3. Click the Triggering Events tab.
4. Click Add to define an event to trigger the flow. The Trigger Flow with Events dialog box appears.
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 Hours and Minutes fields, specify the time when you want the flow to start running.
8. 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.
9. Click OK. The Triggering Event(s) tab reappears, and the time 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 LSF, where it will be scheduled at the specified time, on each day that the specified calendar is true.
From the command line

1. On the command line, type the following:
   
   \[ \text{jsub -T time\_event flow\_file\_name} \]
   
   where \textit{time\_event} is the definition of the time event that triggers this flow and \textit{flow\_file\_name} is the full path name of the file containing the flow definition.

2. Press \textit{Enter}.

To run a flow at multiple times on a single date:

1. In the Flow Editor, open the flow definition.
2. Right-click in an empty space in the flow definition and select \textbf{Flow Attribute}. The Flow Attributes dialog box appears.
3. Click the \textbf{Triggering Events} tab.
4. Click \textbf{Add} to define an event to trigger the flow. The Trigger Flow with Events dialog box appears.

   ![Trigger Flow with Events dialog box]

5. In the \textbf{Select type of event} field, select \textit{Time Event}.
6. In the \textbf{Calendar name} field, select the calendar that resolves to the dates on which you want this flow to run.
7. In the \textbf{Hours} and \textbf{Minutes} fields, specify an expression that resolves to the times when you want the flow to start running. Be sure to specify the times as they appear on a 24-hour clock, where valid values for hours are from 0 to 23. For the syntax of the time expression, see “Specifying time expressions” on page 117.
8. In the \textbf{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.
9 Click **OK**. The Triggering Event(s) tab reappears, and the time 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 LSF, where it will be scheduled at the specified times, each day the calendar is true.

### Specifying time expressions

You can specify several times for the event to trigger. You can:

- Specify a list of times separated by commas. For example, to run the flow at 2:00 p.m., 3:00 p.m. and 5:00 p.m., specify the following in the **Hours** field:
  
  14, 15, 17

- Specify a range of hours. For example, to run the flow every hour from 1:00 a.m. to 5:00 a.m., specify the following in the **Hours** field:
  
  1–5

- Specify a combination of the above. For example, to run the flow at 2:00 p.m., 3:00 p.m., and every hour from 7:00 p.m. to 10:00 p.m., specify the following in the **Hours** field:
  
  14, 15, 19–22

- Use the Minutes field to modify the value in the Hours field. For example, specify the following in the **Hours** field:
  
  7, 9, 11–13

  and the following in the **Minutes** field:

  15, 30


- Use an asterisk (*) in the Hours field to specify every hour, or in the Minutes field to specify every minute. For example, to run a flow every hour, in the Hours field, specify an asterisk (*).
To run a flow based on file activity:

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 appears.

3. Click the **Triggering Events** tab.

4. Click **Add** to define an event to trigger the flow. The Trigger Flow with Events dialog box appears.

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 “Using User Variables within a Flow Definition” on page 55 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 LSF, where it is triggered when the specified file event is true.

**Example: triggering when a file exists**

The following file event triggers the flow when the file `/tmp/core` exists:

When triggering a flow when a file exists, keep the following in mind:
- LSF 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 LSF 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 LSF polls and finds the file exists, unless you combine this event with another such as a time event.
Example: triggering when a file is deleted

The following file event triggers the flow when the file tmp/update is deleted:

| File name: | tmp/update | Browse... |
| Condition: | does not exist | |

After the flow is triggered, it will trigger again each time LSF polls and finds the file does not exist, unless you combine this event with another such as a time event.

Example: 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: | /tmp/data | Browse... |
| Condition: | age | Greater than 15 minutes |

Example: triggering whenever a file arrives

The following file event triggers the flow every time a tar file arrives in the tmp directory:

| File name: | tmp/*tar | Browse... |
| Condition: | arrival |

From the command line

1. On the command line, type the following:
   
   `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
   ```

2. Press Enter.
Running Your Flow

Using Platform Process Manager

Running 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.

To run a flow when another flow completes:

1. In the Flow Editor, open the flow definition.
2. Right-click in an empty space in the flow definition and select **Flow Attribute**. The Flow Attributes dialog box appears.
3. Click the **Triggering Events** tab.
4. Click **Add** to define an event to trigger the 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. 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 LSF, where it is triggered when the specified file event is true.

**Example: 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:

**Example: 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 LSF 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.

To run a flow when a proxy job completes:

1. In the Flow Editor, open the flow definition.
2. Right-click in an empty space in the flow definition and select **Flow Attribute**. The Flow Attributes dialog box appears.
3. Click the **Triggering Events** tab.
4. Click **Add** to define an event to trigger the 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...** 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 LSF, where it is triggered when the specified file event is true.

---

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

1 On the command line, to trigger when the job fails, type the following:

```
jsub -p "job(exit(bhorner:testflow:J2))"
```

2 Press **Enter**.
Specifying 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.

Email notification for a flow

By default, LSF 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 LSF 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.

To specify flow attributes:

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

2 On the **General** tab, enter the description text in the field provided. When you have finished typing the description, click **OK**.

3 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.

4 In the **Email address** field, specify the email address to be notified. The default email address is your user name.

5 To prevent concurrent versions of the same flow, in the **Options** box, check **Allow only one flow to run at a time**.

6 Click **OK**.

**To turn off email notification for a flow:**

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

2 Click **OK**.
About Flow Completion Attributes

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

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

Default completion criteria of a flow

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

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

Alternative completion criteria

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

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

Default completion behavior of a flow

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

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

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

You can direct LSF to continue processing work in a flow even after it is considered complete and has been assigned a state. In this case, LSF 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 run as much as it can, and then you will perform some manual recovery and rerun the failed branch.
Specifying Flow Completion Attributes

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

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

You can also specify what LSF should do when the state of the flow is determined:

- Complete any work in progress and stop running the flow. This is the default.
- Continue running the flow until any remaining work items that can complete, complete.

To assign a state to a flow when all work items are Done:

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

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

3. If you want the flow to stop running if any work item exits or is killed, leave the second option at the default Complete any work in progress and stop running the flow. If you want to continue to process as many jobs in the flow as possible, click Continue running the flow.
Specifying Flow Completion Attributes

4. Click **OK**. The flow will be assigned a state when all of the work items complete successfully or any work item fails or is killed.

### To assign a state to a flow when all selected work items end:

1. From the **Action** menu, select **Specify Flow Completion Attributes**, or right-click in a blank section of the flow definition, and select **Completion Attributes**. The Flow Completion Attributes dialog box appears.
2. Select **All selected work items end**.
3. From the list of available work items, select those that must process before the flow can be assigned a state. Select each item and click Add> to move it to the list of selected items, or double-click on an item to move it to the other list.
4. If you want the flow to stop running when the specified jobs end, leave the second option at the default **Complete any work in progress and stop running the flow**. If you want to continue to process as many jobs in the flow as possible, click **Continue running the flow**.
5. Click **OK**. The flow will be assigned a state when all of the selected work items end.

### To assign a state to a flow when any selected work item ends:

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

### To continue processing when the state of the flow is determined:

1. From the **Action** menu, select **Specify Flow Completion Attributes**, or right-click in a blank section of the flow definition, and select **Completion Attributes**. The Flow Completion Attributes dialog box appears.
2. Select **Continue running the flow**.
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 LSF 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.
Saving 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 name using alphanumeric characters, periods (.), underscores (_), or dashes (-) or pound signs (#). You cannot use a colon (:), or semicolon (;) in a flow definition name.

The file name you assign is concatenated with your user ID to become the flow definition 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 LSF. If you make a change to the flow definition, you need to submit the flow definition again before the changes take effect in LSF.
Running 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. See “Submitting Your Flow Definition” on page 133.

To run a flow immediately, only once:

**From the Flow Editor**

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**

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.
Submitting Your Flow Definition

Until you submit a flow definition, LSF is not aware of it. Submitting a flow definition places it under the control of LSF. Once a flow definition is submitted, LSF determines when the flow is to run, and triggers it as appropriate.

To submit a flow:

1. When you have completed defining your flow, it is recommended that you save it before you submit it. See “Saving the Flow Definition” on page 131.

2. From the **Action** menu, select **Submit**. When the flow is submitted successfully, you receive a confirmation message.
CHAPTER 6

Controlling 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.

Contents

◆ “About the Flow Manager” on page 136
◆ “Filtering the Data Displayed in the Tree View” on page 141
◆ “Triggering a Flow” on page 144
◆ “Viewing a Flow” on page 145
◆ “Determining the Status of Jobs in a Flow” on page 149
◆ “Manually Completing a Dependency” on page 151
◆ “Killing a Running Job” on page 152
◆ “Running a Single Job” on page 153
◆ “Marking a Job Complete” on page 154
◆ “Working with Manual Jobs” on page 155
◆ “Killing a Running Flow” on page 158
◆ “Suspending a Running Flow” on page 159
◆ “Resuming a Suspended Flow” on page 160
◆ “Rerunning an Exited Flow” on page 161
◆ “Viewing History” on page 162
◆ “Holding a Flow Definition” on page 164
◆ “Releasing a Flow Definition from Hold” on page 165
◆ “Viewing a Flow Definition” on page 166
◆ “Removing a Flow Definition” on page 167
About the Flow Manager

You use the Flow Manager to view the status of flows, jobs, job arrays and subflows that are currently in the system, or have run recently. You also use the Flow Manager to:

- Trigger a flow
- Place a flow definition on hold, or release it from hold
- Kill, suspend, resume or rerun a flow
- Kill, run or rerun a job
- Force a job complete

The Process Manager Server must be running before you can open the Flow Manager.

About the Flow Manager user interface

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 User—flow definitions and flows are sorted by user ID
  - By State—flows are sorted by their current state
  - By Event—flows are sorted by their triggering events
  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

About the Action menu

The Action menu allows you to act upon a flow in the tree view. Menu options from the Action menu do not affect data in the right-hand pane.

By User view

When the By 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.
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.
About the Flow Manager

By Event view

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.

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.

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.
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.

<table>
<thead>
<tr>
<th>ID</th>
<th>User</th>
<th>Name</th>
<th>Completion Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>John</td>
<td>Job1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>Jane</td>
<td>Job2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 the automatic refresh entirely.

To refresh the data displayed manually:

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

To change the automatic refresh option:

1. From the File menu, select Properties. The Set Refresh Rate dialog appears.
2 If you want the data to refresh automatically, leave **Refresh automatically** checked.

3 Specify the number of seconds between refreshes of the data.

4 Click **OK**.
Filtering 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
- When you want to limit the flows displayed to those that ran within a particular time window

To limit the flows displayed to those owned by a user:

1. In the Flow Manager, select the view of the data you want—by state or by event by clicking the appropriate tab at the top of the left pane.
2. From the View menu, select Set Filter...
3. In the User field, specify the name of the user whose flows you want to see.
4. Click OK. The view of the flows is refreshed with the new filter applied.

To limit flows displayed to last x hours:

1. In the Flow Manager, select the view of the data you want—by state or by event by clicking the appropriate tab at the top of the left pane.
2. From the View menu, select Set Filter...
3. Click Specify the time range.
4. In the Within the last field, specify the number of hours of flow data to include.
To limit flows displayed to a time period:

1. In the Flow Manager, select the view of the data you want—by state or by event by clicking the appropriate tab at the top of the left pane.
2. From the View menu, select Set Filter....
3. Click Specify the time range.
4. Select From.
5. In the first input field, specify the starting date and time of the time period for which you want to display flows.
6. In the second input field, specify the ending date and time of the time period for which you want to display flows.

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

To trigger a flow:

From the Flow Manager
1. In the Flow Manager, select **By User**.
2. In the tree view of the Flow Manager, expand the tree until you see the flow definition you want to trigger.
3. Right-click on the flow definition, and select **Trigger**. A flow is created and run.

From the command line
1. On the command line, type the following:
   
   ```
   jtrigger flow_definition_name
   ```
   
   where `flow_definition_name` is the name of the flow definition you want to trigger.
2. Press **Enter**.

To trigger a flow, passing it values for variables:

From the Flow Manager
1. In the tree view of the Flow Manager, expand the tree until you see the flow definition you want to trigger.
2. Right-click on the flow definition, and select **Trigger**, then select **With Variables**.
   
   The Pass Variables to Flow dialog box appears.
3. Specify the parameters to pass in the following format:
   ```
   variable=value; variable=value...
   ```
4. Click **OK**. A flow is created and run.
Viewing a Flow

You can view a flow as it is running, and see both the state of the flow and the state of individual jobs within the flow. You can also see the following:

- Definitions of each of the work items in the flow
- Runtime attributes of each of the work items in the flow
- The events that trigger the flow

To view a flow:

1. In the Flow Manager, select the most appropriate view for finding the flow.
2. Expand the tree view under the appropriate user ID until you see the flow you want to view.
3. Double-click on the flow. The flow is displayed in the right-hand pane.

To view a job definition:

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 view.
3. Double-click on the flow. The flow is displayed in the right-hand pane.
4. Right-click on the job whose definition you want to display, and select Open Definition. The job definition dialog is displayed. You cannot change the definition here.

To view runtime attributes of a work item:

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 view.
3. Double-click on the flow. The flow is displayed in the right-hand pane.
4. Double-click on the work item whose runtime attributes you want to display. The Runtime Attributes window is displayed.
You can also expand a subflow and view the runtime attributes of a job within the subflow.

<table>
<thead>
<tr>
<th>Runtime Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Job name] : J1</td>
</tr>
<tr>
<td>[Job ID] : 1456</td>
</tr>
<tr>
<td>[Submitter] : bhornez</td>
</tr>
<tr>
<td>[Command] : sleep 5</td>
</tr>
<tr>
<td>[State] : Done</td>
</tr>
<tr>
<td>[Exit code] : 0</td>
</tr>
<tr>
<td>[CPU usage] : 0.23</td>
</tr>
<tr>
<td>[Finish time]: Thu Aug 15 17:19:39 GMT 2005</td>
</tr>
</tbody>
</table>

**To view the jobs in a subflow:**

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 view.
3. Double-click on the flow. The flow is displayed in the right-hand pane.
4. Right-click on the subflow you want to expand, and select **Expand Subflow**. The subflow is displayed in place of the parent flow.
5. To return to the original flow, click or right-click on an empty place in the subflow, and select **Return to Parent Flow**.

**To view the completion attributes of a subflow:**

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 view.
3. Double-click on the flow. The flow is displayed in the right-hand pane.
4. Right-click on the subflow you want to expand, and select **Expand Subflow**. The subflow is displayed in place of the parent flow.
5. Right-click in an empty space in the subflow, and select **Completion Attributes**. The Flow Completion Attributes dialog appears.
To return to the original flow, right-click in an empty space in the subflow, and select **Return to Parent Flow**.

**To view the events that trigger a flow:**

You can view the events from either the flow definition or the flow itself.

1. In the Flow Manager, select **By User**.
2. In the tree view, locate the flow you want to view.
3. Right-click on the flow definition name and select **View Event**. The Flow-Triggering Event(s) dialog appears.
Viewing a Flow

Using Platform Process Manager
Determining 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.

<table>
<thead>
<tr>
<th>When the Color is ...</th>
<th>The State is ...</th>
<th>Which means the Work Item ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>Done</td>
<td>Completed successfully</td>
</tr>
<tr>
<td>Red</td>
<td>Exit</td>
<td>Failed</td>
</tr>
<tr>
<td></td>
<td>Killed</td>
<td>Was killed when the flow was killed</td>
</tr>
<tr>
<td>Green</td>
<td>Running</td>
<td>Is currently running</td>
</tr>
<tr>
<td></td>
<td>Waiting for completion</td>
<td>Manual Job completion required</td>
</tr>
<tr>
<td>Yellow/orange</td>
<td>Waiting</td>
<td>Is waiting to be dispatched, or was suspended while it was waiting</td>
</tr>
<tr>
<td></td>
<td>Initializing</td>
<td>Is still initializing</td>
</tr>
<tr>
<td>Black</td>
<td>Suspended</td>
<td>Was suspended after the flow started to run</td>
</tr>
<tr>
<td></td>
<td>Initializing</td>
<td>Was suspended while the flow was initializing</td>
</tr>
<tr>
<td></td>
<td>Suspended</td>
<td>Was suspended while the job was waiting to be dispatched</td>
</tr>
<tr>
<td>Gray</td>
<td>On hold</td>
<td>The job is held in the flow definition—it cannot be run</td>
</tr>
</tbody>
</table>

In the following example, the flow testflow was suspended. However, J1 and J2 completed successfully before the flow was suspended. They have blue borders. J3 has a black border—it is suspended.

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:
Determining the Status of Jobs in a Flow

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:

<table>
<thead>
<tr>
<th>Runtime Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Job name] : J1</td>
</tr>
<tr>
<td>[Job ID] : 1456</td>
</tr>
<tr>
<td>[Submitter] : bhorner</td>
</tr>
<tr>
<td>[Command] : sleep $</td>
</tr>
<tr>
<td>[State] : Done</td>
</tr>
<tr>
<td>[Exit code] : 0</td>
</tr>
<tr>
<td>[CPU usage] : 0.23</td>
</tr>
</tbody>
</table>
Manually Completing a Dependency

You can manually complete a dependency, so that a work item no longer needs to wait for that dependency to be met. You can select any dependency within a flow and complete it. This is useful if the duration on a file event has expired, and the file is now available, or if you determine that the dependency can never be met, and there is a case for running the work item anyway.

You can also manually complete a dependency that triggers a flow.

To manually complete a dependency:

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 or Flow 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.
Killing a Running Job

You can kill an individual job that is running, without killing the entire flow.

To kill a running job:

**From the Flow Manager**

1. In the Flow Manager, locate the flow containing the job you want to kill, and open the flow.
2. Locate the job you want to kill, and right click on it.
3. From the menu, select **Kill**.

**From the command line**

1. On the command line, specify the following:
   
   ```
   jjob -i flow_id -k flow_name:subflow_name:job_name
   ```
   
   where `flow_ID` is the unique flow ID containing the job you want to kill, `flow_name` is the name of your flow, `subflow_name` is the name of your subflow (if you have one), and `job_name` is the name of your job.
2. Press **Enter**.
Running a Single Job

You can run or rerun a single job directly from the Flow Manager. You may want to use this option to debug a flow, or to run a single job to fix a flow. You can use this option to rerun a job, regardless of whether the job failed or completed successfully.

You can run or rerun a job in a suspended flow, but the state of the flow does not change. If you specify a rerun exception handler to rerun a job, and the flow is suspended, the job does not run until the flow has been resumed.

When you rerun a job in a flow that is running, successor work items run as designed. When you rerun a job in a flow that is Exited or Killed, only the job runs—its successors do not. If you want to rerun more than one job, you must wait until one job completes before rerunning the next—you rerun them one at a time. If you want to rerun multiple jobs, or if you want the successor jobs to run, you need to rerun the flow. See “Rerunning an Exited Flow” on page 161.

To run or rerun a single job:

1. **From the Flow Manager**
   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.

2. **From the command line**
   1. On the command line, specify the following:
      ```
      jjob -i flow_id -r flow_name[:subflow_name]:job_name
      ```
      where `flow_ID` is the unique flow ID containing the job you want to run, `flow_name` is the name of your flow, `subflow_name` is the name of your subflow (if you have one), and `job_name` is the name of your job.
   2. Press Enter.
Marking a Job Complete

You can mark a job complete without actually running the job. Use this option when you want a flow to continue running, even though the job failed or did not run. Marking a job complete does not actually run the job—it just changes its state. You mark a job complete so that its successor jobs can run when you rerun the flow.

You can only complete a job in a flow that has exited.

To mark a job complete:

**From the Flow Manager**

1. Locate the flow containing the job you want to mark complete, and open the flow.
2. Locate the job you want to mark complete, and right click on it.
3. From the menu, select **Complete Job**.

**From the command line**

1. On the command line, specify the following:

   ```
   jjob -i flow_ID -c flow_name[:subflow_name]:job_name
   ```

   where `flow_ID` is the unique flow ID containing the job you want to mark complete, `flow_name` is the name of your flow, `subflow_name` is the name of your subflow (if you have one) and `job_name` is the name of your job.

1. Press **Enter**.
Working 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 workflow, 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.

To view the manual jobs awaiting completion:

From the Flow Manager
1. In the Flow Manager, from the View menu, select Manual Jobs. The Manual Jobs window appears, listing the manual jobs that are not yet completed.

   ![Manual Jobs Window](image)

   Note that not all manual jobs in this list are ready to be completed. Those manual jobs that are awaiting completion have a check mark in the Completion Required column.

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

To complete a manual job:

From the Flow Manager:
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 using one of the following methods:
   a. Left-click on the job to select it.
   b. Click the Complete Manual Job button at the bottom of the window.
   or
   a. Right-click on the job.
4. If applicable, in the Description field, specify any comments required to describe what happened. For example:
The description you enter here appears in the Runtime Attributes of the manual job.

5. Click **Complete Manual Job**.

or

1. In the Flow Manager, open the flow containing the manual job requiring completion.

2. Left-click on the manual job requiring completion within the flow.

3. Click the blue button. The Complete Manual Job dialog appears.

4. If applicable, in the **Description** field, specify any comments required to describe what happened. For example:
Controlling a Flow

Using Platform Process Manager

The description you enter here appears in the Runtime Attributes of the manual job.

5 Click Complete Manual Job.

From the command line:

1 On the command line, type the following:

```
jcomplete -i flow_id flow_name[:subflow_name]:job_name
```

where `flow_id` is the unique ID of the flow containing the manual job, and `flow_name[:subflow_name]:job_name` is the fully qualified name of the manual job to complete.

2 Press Enter.
Killing a Running Flow

You can kill a flow any time after it has started running. Killing a flow kills any work items within the flow that have not yet completed.

To kill a flow:

1. **From the Flow Manager**
   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.

1. **From the command line**
   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**.
Suspending a Running Flow

You can suspend a flow after it has started running. Suspending a flow suspends all jobs, job arrays and subflows within the flow that have not yet completed. Any jobs that were already completed before the flow was suspended are not affected by either suspending or resuming the flow.

To suspend a running flow:

**From the Flow Manager**
1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow you want to suspend.
3. Right-click on the flow and select **Suspend**. All incomplete and waiting jobs in the flow are suspended until they are explicitly resumed.

**From the command line**
1. On the command line, specify the following:

   ```
   jstop flow_id
   ```

   where `flow_id` is the unique ID of the flow you want to suspend.
2. Press **Enter**.
Resuming a Suspended Flow

You can resume a flow after it has been suspended. Resuming a flow resumes all suspended jobs, job arrays and subflows within the flow. Any jobs that were already completed before the flow was suspended are not affected by either suspending or resuming the flow.

To resume a flow:

From the Flow Manager

1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow you want to resume.
3. Right-click on the flow and select Resume. All suspended jobs in the flow are now resumed.

From the command line

1. On the command line, specify the following:
   
   ```
   jresume flow_id
   ```
   
   where `flow_id` is the unique ID of the flow you want to resume.
2. Press Enter.
Rerunning an Exited Flow

You can rerun a flow that has exited, provided that the flow was not killed.

When you rerun a flow, jobs are rerun as follows:

- If the flow uses the default completion criteria (the flow exits when a job exits), the flow runs again, beginning with the job that exited. Only exited jobs are rerun.
- If the flow uses completion criteria (the flow is complete when one or more specified jobs in the flow complete), the flow runs again, beginning with the jobs that exited, but all successor jobs are also rerun, even if they are Done.

If you need to rerun a flow that was killed, retrigger the flow.

Rerunning a flow that contains an alarm does not reopen a previously opened alarm. Similarly, rerunning a flow that contains a manual job that was already marked complete does not reset the state of the manual job. If the flow contains a manual job that was already marked complete, you may need to run its successor jobs manually.

To rerun a flow:

**From the Flow Manager**
1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow you want to rerun.
3. Ensure that no jobs are still running within the flow—sometimes elements of a job array or jobs in a subflow may continue to run after a flow exits.
4. Right-click on the flow and select Rerun. The flow is rerun, beginning at any jobs that exited or were killed.

**From the command line**
1. On the command line, specify the following:
   ```
   jrerun flow_id
   ```
   where `flow_id` is the unique ID of the flow you want to rerun.
2. Press Enter.
Viewing History

You can see the history of a flow definition, flow, or work item within a flow.

The history information is available for a set period of time after a flow has completed. By default, that time is 24 hours. Check with your Process Manager administrator to determine the setting at your site.

To view the history of a flow definition:

From the Flow Manager

1. In the Flow Manager, select **By User**.
2. In the tree view, locate the flow definition for which you want history.
3. Right-click on the name of the flow definition and select **View History**. The History dialog appears.
4. Click **OK** when you have finished reviewing the history.

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

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

From the Flow Manager
1. In the Flow Manager, select the most appropriate view for finding the flow.
2. In the tree view, locate the flow for which you want history.
3. Right-click on the flow and select View History, or from the Flow menu, select View History. The History dialog appears.
4. Click OK when you have finished reviewing the history.

For a flow, you can see the following information:
- When it started
- If and when it was killed
- If and when it was suspended
- If and when it was resumed
- If and when any exceptions occurred
- When it completed
- Time zone information for Process Manager Server

To view the history of a job or job array:

From the Flow Manager
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 view.
3. Double-click on the flow. The flow is displayed in the right-hand pane.
4. Right-click on the job for which you want to see history.
5. From the Job menu, select History. The History window appears.
6. Click OK when you have finished reviewing the history.

For a job or job array, you can see the following information:
- The user name
- The ID of the flow in which it ran
- The job name
- The job ID
- The state of the job
- The status of the job
- When the job started
- When the job completed
- The CPU usage of the job
- Time zone information for Process Manager Server
Holding a Flow Definition

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.

To hold a flow definition:

1. **From the Flow Manager**
   1. In the Flow Manager, select **By User**.
   2. Expand the tree view under the appropriate user ID 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.

2. **From the command line**
   1. On the command line, type the following:
      
      ```
jhold flow_definition_name
      ```
      
      where `flow_definition_name` is the name of the flow definition you want to place on hold.
   2. Press **Enter**.
Releasing a Flow Definition from Hold

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

To release a flow definition from hold:

**From the Flow Manager**

1. In the Flow Manager, select **By User**.
2. Expand the tree view under the appropriate user ID until you see the flow you want to release.
3. Right-click on the flow definition and select **Release**. The status of the flow definition changes to Released.

**From the command line**

1. On the command line, type the following:
   
   ```
   jrelease flow_name
   ```
   
   where `flow_name` is the name of the flow definition you want to release.
2. Press **Enter**.
Viewing a Flow Definition

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

To view a flow definition:

1. In the Flow Manager, select **By User**.
2. Under the appropriate user ID in the tree view, locate the flow definition you want to view. It is listed by name, above every occurrence of the flow that is in the system:

   ![Flow Manager Tree View](image)

   In the above example, `pay1` is the flow definition. Below it is `1`, the ID of the flow that just completed.

3. Right-click on the definition name, and select **View Flow**. The flow definition is displayed in the right-hand pane. You cannot edit the definition here—you can only change the definition in the Flow Editor.
Removing a Flow Definition

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.

To remove a flow definition:

**From the Flow Manager**

1. In the Flow Manager, select **By User**.
2. Under the appropriate user ID in the tree view, locate the flow definition you want to remove.
3. Right-click on the flow definition and select **Remove**.
4. Confirm that you want to remove this definition. The flow definition is removed from the system.

**From the command line**

1. On the command line, type the following:
   
   ```
   jremove flow_name
   ```
   
   where `flow_name` is the name of the flow definition you want to remove.
2. Press **Enter**.
Removing a Flow Definition
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.

Contents
◆ “About Process Manager Commands” on page 171
◆ “caleditor” on page 172
◆ “floweditor” on page 173
◆ “flowmanager” on page 174
◆ “jadmin” on page 176
◆ “jalarms” on page 177
◆ “jcadd” on page 180
◆ “jcals” on page 185
◆ “jcdel” on page 187
◆ “jcmd” on page 188
◆ “jcomplete” on page 192
◆ “jdefs” on page 194
◆ “jflows” on page 197
◆ “jhist” on page 200
◆ “jhold” on page 205
◆ “jid” on page 206
◆ “jjob” on page 207
◆ “jkill” on page 209
◆ “jmanuals” on page 211
◆ “jreconfigalarm” on page 213
◆ “jrelease” on page 214
◆ “jremove” on page 215
◆ “jrerun” on page 216
◆ “jresume” on page 217
◆ “jrun” on page 219
◆ “jsetvars” on page 220
◆ “jsinstall” on page 221
◆ “jstop” on page 222
◆ “jsub” on page 224
◆ “jtrigger” on page 231
About Process Manager Commands

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
- `jcde1`—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`—to submit a flow definition to Process Manager
- `jtrigger`—to trigger the creation of a flow
- `jhold`—to place a flow definition on hold, preventing automatic triggering of the flow
- `jrelease`—to release a flow definition from hold, enabling automatic triggering of the flow
- `jdefs`—to display information about flow definitions
- `jremove`—to remove a flow definition from Process Manager

Flow monitor and control commands

You can use the following commands to monitor and control flows that are in the process of running or have recently completed:
- `flowmanager`—to start the Flow Manager graphical user interface
- `jalarms`—to list open alarms
- `jcomplete`—to complete a manual job
- `jflows`—to display information about a flow
- `jkill`—to kill a flow
- `jmanuals`—to list all manual jobs waiting for completion
- `jrerun`—to rerun an exited flow
- `jstop`—to suspend a flow
- `jresume`—to resume a suspended flow
caleditor

starts the Calendar Editor.

SYNOPSIS

caleditor

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

EXAMPLES

% caleditor

opens the Calendar Editor.
**floweditor**

starts the Flow Editor.

**SYNOPSIS**

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

**DESCRIPTION**

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

**OPTIONS**

`file_name`

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

**EXAMPLES**

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

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

```
% floweditor
```

opens the Flow Editor with no files opened.
flowmanager

starts the Flow Manager.

SYNOPSIS

flowmanager

DESCRIPTION

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

EXAMPLE

% flowmanager

opens the Flow Manager.
**jadmin**

controls the Process Manager daemon jfd on UNIX.

**SYNOPSIS**

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

**DESCRIPTION**

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

**OPTIONS**

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

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

- **-h**
  
  Prints the command usage to stderr and exits.

- **-V**
  
  Prints the Process Manager release version to stderr and exits.

**EXAMPLES**

```
# jadmin start
```

Starts the Process Manager daemon.

```
# jadmin stop
```

Stops the Process Manager daemon.

**SEE ALSO**

`jfd`, `js.conf`
jalarms

lists the open alarms in Process Manager.

SYNOPSIS

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

DESCRIPTION

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

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

OPTIONS

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

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

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

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

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

The time interval can be specified in many ways. For more specific syntax and examples of time formats, see “TIME INTERVAL FORMAT” on page 178.

- `-h`
  Prints the command usage to `stderr` and exits.

- `-V`
  Prints the Process Manager release version to `stderr` and exits.
TIME INTERVAL FORMAT

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

Specify the time interval is follows:

\[ start\_time, end\_time | start\_time, | end\_time | start\_time \]

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

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

Where:

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

\[ start\_time, end\_time \]

Specifies both the start and end times of the interval.

\[ start\_time, \]

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

\[ ,end\_time \]

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

\[ start\_time \]

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

ABSOLUTE TIME EXAMPLES

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

\[ 1,8 \] = May 1 00:00 2002 to May 8 23:59 2002
\[ ,4 \] = the time of the first occurrence to May 4 23:59 2002
\[ 6 \] = May 6 00:00 2002 to May 6 23:59 2002
\[ 3/ \] = Mar 1 00:00 2002 to Mar 31 23:59 2002
\[ /12: \] = May 9 12:00 2002 to May 9 12:59 2002
\[ 2/1 \] = Feb 1 00:00 2002 to Feb 1 23:59 2002
\[ 2/1, \] = Feb 1 00:00 to the current time
\[ ,. \] = the time of the first occurrence to the current time
RELATIVE TIME EXAMPLES

\[\text{-9,} \quad \text{= April 30 17:06 2002 to the current time}\]
\[\text{,.-2} \quad \text{= the time of the first occurrence to Mar 7 17:06 2002}\]
\[\text{-9,.-2} \quad \text{= nine days ago to two days ago (April 30, 2002 17:06 to May 7, 2002 17:06)}\]

EXAMPLES

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

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

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

SYNOPSIS

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

jcadd [-h] [-V]

DESCRIPTION

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

OPTIONS

-d description

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

-t cal_expression

Specifies the dates on which you want some action to take place. You can enter specific dates, a range of dates, or a more complex expression that resolves to a series of dates. See “Creating Calendar Expressions” on page 181 for more information.

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

-cal_name

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

-h

Prints the command usage to stderr and exits.

-V

Prints the Process Manager release version to stderr and exits.

Limitations

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

RANGE(startdate [, enddate]) : PERIOD(1, *, step) : occurrence
Some examples that follow this format are:

- `RANGE(2001/1/1,2002/1/1):day(1,*),3`  
- `RANGE(2001/1/1,2002/1/1):week(1,*),3):MON,TUE`  
- `RANGE(2001/1/1,2002/1/1):week(1,*),3):ABC(1)`  
- `RANGE(2001/1/1,2002/1/1):month(1,*,3):1,3,5`  
- `RANGE(2001/1/1,2002/1/1):month(1,*,3):MON(1),TUE(1)`  
- `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 `jcemod` 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
```
Using Platform Process Manager

or

\[ \text{RANGE}(\text{startdate}, \text{enddate}) \text{:week}(1, *, \text{step}):\text{abc}(ii) \]

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

\[ \text{RANGE}(2002/01/01):\text{week}(1, *, 3):\text{MON}(-1) \]

In the above example, \text{MON}(-1) refers to last Monday.

To create schedules that recur monthly:

Specify the expression in one of the following formats:

\[ \text{RANGE}(\text{startdate}, \text{enddate}) \text{:month}(1, *, \text{step}):\text{day_of_month} \]

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

\[ \text{RANGE}(2002/12/31):\text{month}(1, *, 2):1,15,30 \]

or

\[ \text{RANGE}(\text{startdate}, \text{enddate}) \text{:month}(1, *, \text{step}):\text{abc}(ii) \]

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

\[ \text{RANGE}(2002/01/01):\text{month}(1, *, 3):\text{MON}(-1) \]

In the above example, \text{MON}(-1) refers to last Monday.

or

\[ \text{RANGE}(\text{startdate}, \text{enddate}) \text{:month}(1, *, \text{step}):\text{day_of_week}(ii) \]

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

\[ \text{RANGE}(2002/01/01):\text{month}(1, *, 3):\text{MON}(-1) \]

In the above example, \text{MON}(-1) refers to last Monday. For a list of built-in keywords, see “Built-in keywords—reserved words” on page 183.

To create schedules that recur yearly:

Specify the expression in the following format:

\[ \text{RANGE}(\text{startdate}, \text{enddate}) \text{:month:day} \]

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

\[ \text{RANGE}(2002/1/1, 2004/12/31):\text{JAN}:1 \]

To merge calendar expressions:

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

\[ \text{Mondays@Sys} | \text{Fridays@Sys} \&\& \text{!Holidays@Sys} \]

where \text{Mondays@Sys}, \text{Fridays@Sys} and \text{Holidays@Sys} are all predefined system calendars.
Built-in keywords—reserved words

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

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

EXAMPLES

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

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

% jcadd -d "Mondays, Wednesdays and Fridays" -t "Mondays@Sys || Wednesdays@Sys || Fridays@Sys" Everyotherday
Creates a calendar called `Everyotherday` that resolves to Mondays, Wednesdays and Fridays.

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

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

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

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

SEE ALSO

`jcde1`, `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

-1

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.
The output without `-l`:

```
bhorner@curie-64: jcalls
```

<table>
<thead>
<tr>
<th>CALENDAR NAME</th>
<th>OWNER</th>
<th>EXPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>mgmtreport</td>
<td>bhorner</td>
<td>(RANGE(2002/6/7):week(1,3,2):THU,FRI)</td>
</tr>
</tbody>
</table>

The output with `-l`:

```
bhorner@curie-65: jcalls -l
```

```
CALENDAR: payday
-- Pay days
OWNER    TODAY    LAST CAL DATE    NEXT CAL DATE
bhorner  false    Wed May 15 2002

CALENDAR: mgmtreport
-- Management report days
OWNER    TODAY    LAST CAL DATE    NEXT CAL DATE
bhorner  false    Fri Jun 7 2002
EXPRESSION: (RANGE(2002/6/7):week(1,3,2):THU,FRI)
```

**EXAMPLES**

```
% jcalls -u all

Displays all calendars in Process Manager.
```
**jcdel**

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

**SYNOPSIS**

```
jcdel [-u user_name] cal_name[ cal_name...]
jcdel [-h]|[-V]
```

**DESCRIPTION**

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

**OPTIONS**

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

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

- **-h**
  
  Prints the command usage to stderr and exits.

- **-V**
  
  Prints the Process Manager release version to stderr and exits.

**EXAMPLES**

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

Deletes the calendar Rundays2001 owned by the user barneyt.

**SEE ALSO**

jcadd, jcals
jcmod

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

SYNOPSIS

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

DESCRIPTION

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

OPTIONS

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

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

- `-t cal_expression`
  Specifies the dates on which you want some action to take place. You can enter specific dates, a range of dates, or a more complex expression that resolves to a series of dates. See “Creating Calendar Expressions” on page 188 for more information.

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

- `-h`
  Prints the command usage to stderr and exits.

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

Creating Calendar Expressions

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

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

To create absolute dates:
Specify the date in the following standard format:
\((y y y y / m m / d d)\)
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:
\(\text{RANGE}(\text{startdate}[,\text{enddate}]):\text{day}(1,*,\text{step})\)
The ending date is optional. If it is not specified, the calendar is valid indefinitely. For example:
\(\text{RANGE}(2003/2/1, 2003/12/31):\text{day}(1,*,2)\)
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:
\(\text{RANGE}(\text{startdate}[,\text{enddate}]):\text{week}(1,*,\text{step}):\text{day_of_week}\)
where \(\text{step}\) is the interval between weeks and \(\text{day_of_week}\) is one or more days of the week, separated by commas. For example:
\(\text{RANGE}(2002/12/31):\text{week}(1,*,2):\text{MON, FRI, SAT}\)
or
\(\text{RANGE}(\text{startdate}[,\text{enddate}]):\text{week}(1,*,\text{step}):\text{abc}(i i)\)
where \(\text{step}\) is the interval between weeks, \(\text{abc}\) is a previously defined calendar name and \(i i\) is an integer indicating a specific occurrence of a day within that calendar. For example:
\(\text{RANGE}(2002/01/01):\text{week}(1,*,3):\text{MON(-1)}\)
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:
\(\text{RANGE}(\text{startdate}[,\text{enddate}]):\text{month}(1,*,\text{step}):\text{day_of_month}\)
where \(\text{step}\) is the interval between months and \(\text{day_of_month}\) is one or more days of the month by number, separated by commas. For example:
\(\text{RANGE}(2002/12/31):\text{month}(1,*,2):1,15,30\)
or
\(\text{RANGE}(\text{startdate}[,\text{enddate}]):\text{month}(1,*,\text{step}):\text{abc}(i i)\)
where \(\text{step}\) is the interval between months, \(\text{abc}\) is a previously defined calendar name or built-in keyword and \(i i\) is an integer indicating a specific occurrence of a day within that calendar. For example:
**RANGE(2002/01/01):month(1,*,3):MON(-1)**

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

or

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

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

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

In the above example, MON(-1) refers to last Monday. For a list of built-in keywords, see “Built-in keywords—reserved words” on page 190.

**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
EXAMPLES

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

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

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

SYNOPSIS

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

jcomplete [-h]|[-V]

DESCRIPTION

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

OPTIONS

-d description

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

-i flow_id

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

flow_name:subflow_name:manual_job_name

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

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

-u user_name

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

-h

Prints the command usage to stderr and exits.

-v

Prints the Process Manager release version to stderr and exits.

EXAMPLES

% jcomplete -d "printed check numbers 4002 to 4532" -i 42
payprt:checkprinter
completes the manual job checkprinter in the flow payprt with flow ID 42, and adds the comment “printed check numbers 4002 to 4532”.

SEE ALSO

jmanuels jjob
jdefs

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

SYNOPSIS

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

DESCRIPTION

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

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

OPTIONS

- `-l`

  Specifies to display the information in long format. In addition to the information listed above, this option displays the following information:
  - any events defined to trigger the flow
  - any exit conditions specified in the flow definition

- `-u user_name`

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

- `-s status`

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

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

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

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

**-h**

Prints the command usage to stderr and exits.

**-v**

Prints the Process Manager release version to stderr and exits.

**OUTPUT**

The output without -l:

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

The output with -l:

```
bhorner@curie-63: jdefs -l
NAME: myflow
-- (No description)
USER STATUS FLOW IDS
bhorner Onhold 6(Running) 7(Running)

Triggering Events:
<None>
Exit Condition:
    All jobs complete successfully
```

```
NAME: pay1
-- (No description)
USER STATUS FLOW IDS
bhorner Onhold 2(Done) 8(Running)

Triggering Events:
<None>
Exit Condition:
    All jobs complete successfully
```

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

EXAMPLES

% jdefs -u barneyt -s RELEASE
Displays all flow definitions owned by barneyt that are not on hold.
**jflows**

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

**SYNOPSIS**

```
jflows [-l] [-u user_name] [-u all] [-f flow_name] [-s state]
jflows [-l] [flow_id[ flow_id ...]|0]
jflows [-h]|[-V]
```

**DESCRIPTION**

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

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

**OPTIONS**

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

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

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

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

  - **Done**
    Displays information about flows that completed successfully.

  - **Exit**
    Displays information about flows that failed.

  - **Killed**
    Displays information about flows that were killed.
Running
Displays information about flows that are running.

Suspended
Displays information about flows that were suspended.

Waiting
Displays information about flows that are waiting.

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

0
Specifies to display all flows.

-h
Prints the command usage to stderr and exits.

-v
Prints the Process Manager release version to stderr and exits.

OUTPUT
The output without -l:
bhorner@curie-66: jflows

<table>
<thead>
<tr>
<th>ID</th>
<th>USER</th>
<th>NAME</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bhorner</td>
<td>untitled1</td>
<td>Done</td>
</tr>
<tr>
<td>2</td>
<td>bhorner</td>
<td>pay1</td>
<td>Done</td>
</tr>
<tr>
<td>3</td>
<td>bhorner</td>
<td>untitled2</td>
<td>Running</td>
</tr>
<tr>
<td>4</td>
<td>bhorner</td>
<td>untitled2</td>
<td>Running</td>
</tr>
<tr>
<td>5</td>
<td>bhorner</td>
<td>pay2</td>
<td>Done</td>
</tr>
<tr>
<td>6</td>
<td>bhorner</td>
<td>myflow</td>
<td>Done</td>
</tr>
<tr>
<td>7</td>
<td>bhorner</td>
<td>myflow</td>
<td>Done</td>
</tr>
<tr>
<td>8</td>
<td>bhorner</td>
<td>pay1</td>
<td>Done</td>
</tr>
</tbody>
</table>

The output with -l:
bhorner@curie-67: jflows -l

FLOW ID: 2

<table>
<thead>
<tr>
<th>USER</th>
<th>NAME</th>
<th>STATE</th>
<th>START TIME</th>
<th>END TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>bhorner</td>
<td>pay1</td>
<td>Done</td>
<td>May 3 04:49:03 2005</td>
<td>May 3 04:52:20 2005</td>
</tr>
</tbody>
</table>

DETAILS:

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>STATE</th>
<th>JOBIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:bhorner:pay1:J1</td>
<td>Job</td>
<td>Done</td>
<td>462</td>
</tr>
<tr>
<td>2:bhorner:pay1:J2</td>
<td>Job</td>
<td>Done</td>
<td>463</td>
</tr>
<tr>
<td>2:bhorner:pay1:J3</td>
<td>Job</td>
<td>Done</td>
<td>464</td>
</tr>
<tr>
<td>2:bhorner:pay1:J4</td>
<td>Job</td>
<td>Done</td>
<td>465</td>
</tr>
<tr>
<td>2:bhorner:pay1:pay2</td>
<td>SubFlow</td>
<td>Done</td>
<td></td>
</tr>
<tr>
<td>2:bhorner:pay1:pay2:J1</td>
<td>Job</td>
<td>Done</td>
<td>466</td>
</tr>
<tr>
<td>2:bhorner:pay1:pay2:J2</td>
<td>Job</td>
<td>Done</td>
<td>467</td>
</tr>
<tr>
<td>2:bhorner:pay1:pay2:J3</td>
<td>Job</td>
<td>Done</td>
<td>468</td>
</tr>
</tbody>
</table>
EXAMPLES

% jflows -f myflow

Displays all flows associated with the flow definition myflow
**jhist**

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

**SYNOPSIS**

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

**DESCRIPTION**

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

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

**OPTIONS**

- **-C category**
  Specifies the type of object for which you want to see history. Choose from the following values:
  - `alarm`—displays historical information about one or more alarms
  - `calendar`—displays historical information about one or more calendars
  - `daemon`—displays historical information about Process Manager Server
  - `flowdef`—displays historical information about one or more flow definitions
  - `flow`—displays historical information about one or more flows
  - `job`—displays historical information about one or more jobs or job arrays
  You can specify more than one category by separating categories with a comma (,).

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

- **-t start_time,end_time**
  Specifies the span of time for which you want to display the history. If you do not specify a start time, the start time is assumed to be 7 days prior to the time the `jhist` command is issued. If you do not specify an end time, the end time is assumed to be now.
  Specify the times in the format "yyyy/mm/dd/HH:MM". Do not specify spaces in the time interval string.
  The time interval can be specified in many ways. For more specific syntax and examples of time formats, see “TIME INTERVAL FORMAT” on page 202.
-c calendar_name
Specifications the name of the calendar for which to display historical information. If you do not specify a calendar name when displaying calendars, information is displayed for all calendars owned by the specified user.
Valid only when used with the calendar category.

-f flow_name
Specifies the name of the flow definition for which to display historical information.
Displays flow definition, flow, or job information for flow definitions with the specified name.
Valid only with the flowdef, flow, and job categories.

-i flow_ID
Specifies the ID of the flow for which to display historical information. Displays flow and job information for flows with the specified ID.
Valid only with the flow and job categories.

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

-h
Prints the command usage to stderr and exits.

-v
Prints the Process Manager release version to stderr and exits.

USAGE

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

-C calendar
Displays the times when calendars are added or deleted.

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

-C flowdef
Displays information about when a flow definition state is:
◆ Submit—When a flow definition is submitted
◆ SubmitAndRun—When a flow runs immediately
◆ Remove—When a flow definition is removed from the system
◆ Release—When a flow definition is released from on hold
◆ Hold—When a flow definition is placed on hold
◆ Trigger—When a flow definition is triggered manually or by an event
- **Instantiate**—When a flow is created

---

**-C flow**

Displays information about when a flow state is:
- **Start**—When a flow is started
- **Kill**—When a flow is killed
- **Suspend**—When a flow is suspended
- **Resume**—When a flow is resumed from the Suspended state
- **Finished**—When a flow is completed

---

**-C job**

Displays information about when a job or job array is:
- **Started**
- **Killed**
- **Suspended**
- **Resumed**
- **Finished**

---

**TIME INTERVAL FORMAT**

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

Specify the time interval is follows:

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

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

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

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

**start_time,end_time**

Specifies both the start and end times of the interval.

**start_time,**

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

**,end_time**

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

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

**ABSOLUTE TIME EXAMPLES**

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

- \(1,8\) = May 1 00:00 2005 to May 8 23:59 2005
- \(,4\) = the time of the first occurrence to May 4 23:59 2005
- \(6\) = May 6 00:00 2005 to May 6 23:59 2005
- \(3/\) = Mar 1 00:00 2005 to Mar 31 23:59 2005
- \(/12:\) = May 9 12:00 2005 to May 9 12:59 2005
- \(2/1\) = Feb 1 00:00 2005 to Feb 1 23:59 2005
- \(2/1,\) = Feb 1 00:00 to the current time
- \(,..\) = the time of the first occurrence to the current time
- \(,2/10:\) = the time of the first occurrence to May 2 10:59 2005
- \(2001/12/31,2005/5/1\) = from Dec 31, 2001 00:00:00 to May 1st 2005 23:59:59

**RELATIVE TIME EXAMPLES**

- \(.-9\) = April 30 17:06 2005 to the current time
- \(,.-2/\) = the time of the first occurrence to Mar 7 17:06 2005
- \(.-9,.-2\) = nine days ago to two days ago (April 30, 2005 17:06 to May 7, 2005 17:06)

**OUTPUT**

The following is a sample of the output when \texttt{jhist} is used:

```
\texttt{jhist \textendash C flowdef,flow,job}
```

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>User</th>
<th>Host</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon Aug 12 17:00:01</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Instantiated flow definition bhorner:testflow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FlowId=30</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:01</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Start flow 30:bhorner:testflow</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:01</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Start job 30:bhorner:testflow:J1</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:01</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Instantiated flow definition bhorner:sample3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FlowId=31</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:01</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Start flow 31:bhorner:sample3</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:01</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Start job 31:bhorner:sample3:J1</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:07</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Started job 30:bhorner:testflow:J1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JobId=1189</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:12</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Started job 31:bhorner:sample3:J1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JobId=1190</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:17</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Execute job 30:bhorner:testflow:J1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JobId=1189</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Host=curie</td>
</tr>
<tr>
<td>Mon Aug 12 17:00:27</td>
<td>2005 EST</td>
<td>bhorner</td>
<td></td>
<td>Execute job 31:bhorner:sample3:J1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>JobId=1190</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Host=curie</td>
</tr>
</tbody>
</table>
EXAMPLES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**SYNOPSIS**

```
jhold [-u user_name] flow_name[ flow_name...]
jhold [-h]|[-V]
```

**DESCRIPTION**

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

**OPTIONS**

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

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

- **-h**
  Prints the command usage to `stderr` and exits.

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

**EXAMPLES**

```
% jhold myflow
Places the flow definition `myflow`, which is owned by the current user, on hold.
% jhold -u "user01" payupdt
Places the flow definition `payupdt`, which is owned by user01, on hold.
```

**SEE ALSO**

`jrelease`
**jid**

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

**SYNOPSIS**

```
jid [-h | -v]
```

**DESCRIPTION**

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

**OPTIONS**

- `-h`

  Prints command usage to `stderr` and exits.

- `-v`

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

controls a job in a running flow.

**SYNOPSIS**

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

**DESCRIPTION**

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

**OPTIONS**

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

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

- `-k`  
  Specifies to kill the job.

- `-r`  
  Specifies to run or rerun the job.

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

- `flow_name:subflow_name:manual_job_name`  
  Specifies the name of the job to control. Specify the fully-qualified job name, which is the flow name followed by the subflow name, if applicable, followed by the name of the job. For example:  
  `myflow:print:prtreport`

- `-h`  
  Prints the command usage to *stderr* and exits.

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

**EXAMPLES**

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

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

jmanuals
jkill

kills a flow.

SYNOPSIS

jkill [-u user_name | -u all] [-f flow_name]
jkill flow_id[ flow_id ...] | 0
jkill [-h] | [-V]

DESCRIPTION

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

OPTIONS

-u user_name

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

-f flow_name

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

flow_id

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

0

Specifies to kill all flows.

-h

Prints the command usage to stderr and exits.

-V

Prints the Process Manager release version to stderr and exits.
% jkill -f myflow
Kills all flows associated with the flow definition myflow. Does not affect the flow definition.
jmanuals
displays all manual jobs that have not yet been completed.

SYNOPSIS

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

jmanuals  [-h]      [-V]

DESCRIPTION

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

OPTIONS

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

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

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

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

-r no
Specifies to display only those manual jobs that do not require completion at this time.

-h
Prints the command usage to stderr and exits.

-V
Prints the Process Manager release version to stderr and exits.
OUTPUT

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

```
bhorner@curie-68: jmanuals
```

<table>
<thead>
<tr>
<th>ID</th>
<th>USER</th>
<th>NAME</th>
<th>COMPLETION</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>bhorner</td>
<td>untitled2:M1</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>bhorner</td>
<td>untitled2:Chkprt</td>
<td>Yes</td>
</tr>
</tbody>
</table>

SEE ALSO

`jcomplete`
**jreconfigalarm**

reloads the alarm definitions.

**SYNOPSIS**

```
jreconfigalarm [-h | -V]
```

**DESCRIPTION**

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

**OPTIONS**

- `h`
  
  Prints the command usage to `stderr` and exits.

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

**EXAMPLES**

```
# jreconfigalarm
```

Loads the updated list of Process Manager alarms.

**SEE ALSO**

`jadmin`
**jrelease**

releases a previously held flow definition.

**SYNOPSIS**

```
jrelease [-u user_name] flow_name[ flow_name...]
jrelease [-h] [-V]
```

**DESCRIPTION**

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

**OPTIONS**

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

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

- `-h`
  Prints the command usage to `stderr` and exits.

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

**EXAMPLES**

```
% jrelease myflow
Releases the flow definition `myflow`, which is owned by the current user, from hold.
% jrelease -u "user01" payupdt
Releases the flow definition `payupdt`, which is owned by `user01`, from hold.
```

**SEE ALSO**

`jhold`
**jremove**

removes a previously submitted flow definition from Process Manager.

**SYNOPSIS**

```
jremove [-u user_name] -f flow_name[ flow_name...]
jremove [-h] [-V]
```

**DESCRIPTION**

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

**OPTIONS**

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

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

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

- `-h`
  
  Prints the command usage to `stderr` and exits.

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

**EXAMPLES**

```
% jremove myflow
Removes the definition `myflow` from Process Manager. In this example, `myflow` is owned by the current user.
% jremove -u "user01" payupdt
Removes the definition `payupdt` from Process Manager. In this example, `payupdt` is owned by user01.
```

**SEE ALSO**

`jsub`, `jhold`
**jrerun**

reruns an exited flow.

**SYNOPSIS**

```
jrerun [-v "var=value[;var1=value1;...]"] flow_id[ flow_id...]
jrerun [-h]|[-V]
```

**DESCRIPTION**

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

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

**OPTIONS**

- **-v var=value**

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

- **flow_id**

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

- **-h**

  Prints the command usage to `stderr` and exits.

- **-V**

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

**EXAMPLES**

```
% jrerun 1234
reruns the flow with the flow ID 1234.
% jrerun -v "USER=bhorner" 277
reruns the flow with the flow ID 277 and passes it a value of `jdoe` for the USER variable.
```
jresume
resumes a suspended flow.

SYNOPSIS

jresume [-u user_name|-u all] [-f flow_name]
jresume flow_id[ flow_id..] 0
jresume [-h]|[-V]

DESCRIPTION

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

OPTIONS

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

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

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

0
Specifies to resume all suspended flows.

-h
Prints the command usage to stderr and exits.

-v
Prints the Process Manager release version to stderr and exits.
EXAMPLES

% jresume 14 17 22
Resumes the flows with IDs 14, 17 and 22.

% jresume 0
Resumes all suspended flows owned by the user invoking the command.

% jresume -u all
Resumes all suspended flows owned by all users.

SEE ALSO

jstop
**jrun**

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

**SYNOPSIS**

```
jrun [-v "var=value; var1=value1 ; ..."] flow_file_name
jrun [-h] [-V]
```

**DESCRIPTION**

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

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

**SYNOPSIS**

```
jsetvars -i flow_ID variable=value [variable2=value2...]
jsetvars [-h][-V]
```

**DESCRIPTION**

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

**OPTIONS**

- `-i flow_ID`
  Required. Specifies the ID of the flow in which to change the variable.

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

- `-h`
  Prints the command usage to `stderr` and exits.

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

**EXAMPLES**

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

Changes the value of the priority variable to 10 for the flow with the ID 1234.
Using Platform Process Manager

jsinstall

runs jsinstall, the Platform Process Manager installation and configuration script

SYNOPSIS

jsinstall -f install.config
jsinstall -h

DESCRIPTION

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

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

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

suspends a running flow.

**SYNOPSIS**

```
jstop [-u user_name | -u all] [-f flow_name]
jstop flow_id[ flow_id...] 0
jstop [-h] | [-V]
```

**DESCRIPTION**

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

**OPTIONS**

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

- **-f flow_name**

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

- **flow_id**

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

- **0**

  Specifies to suspend all flows.

- **-h**

  Prints the command usage to `stderr` and exits.

- **-v**

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

% jstop -f "myflow"
Suspends all flows associated with the definition myflow. Does not affect the flow definition.

% jstop 14
Suspends flow ID 14.

% jstop 0
Suspends all flows.

SEE ALSO

jresume
jsub

submits a flow definition to Process Manager.

SYNOPSIS

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

DESCRIPTION

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

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

OPTIONS

- `-H`

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

- `-r`

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

- `-d`

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

- `-T time_event`

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

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

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

**hour:minute**

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

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

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

**%duration**

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

**-F "file_event"**

Specifies to automatically trigger a flow when the specified file events are true. When specifying the file name, you can also specify wildcard characters: * to represent a string or ? to represent a single character. For example, a*.dat* matches abc.dat, another.dat and abc.dat23. S??day* matches Satdays.tar and Sundays.dat. *e matches smile.

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

Specify the file event in one of the following formats:

**arrival(file_location)**

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

**absolute_directory/ filename**
**exist(file_location)**

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

```
absolute_directory/filename
```

! **exist(file_location)**

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

```
absolute_directory/filename
```

**size(file_location) operator size**

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

```
absolute_directory/filename
```

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

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

Specify the size in bytes.

**age(file_location) operator age**

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

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

```
absolute_directory/filename
```

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

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

Specify the age in minutes.

```
-p "proxy_event"
```

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

Specify the proxy event in one of the following formats:

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

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

Valid operators are `>=`, `>`, `<=`, `<`, `!=` and `==`. 
Note: For csh, if you specify != (not equal), you need to precede the operator with a backslash escape character.

Example: on successful completion of J1:

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

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

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

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

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

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

Example: on successful completion of all jobs in Array1:

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

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

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

Example: if more than 3 jobs in A1 exit:

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

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

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

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

Example: on successful completion of all jobs in myflow:

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

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

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

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

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

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

```
-f "flow_event"
```

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

Specify the flow event in one of the following formats:
**done**(*flow_definition_name*)

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

```
user_name:flow_definition
```

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

**end**(*flow_definition_name*)

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

```
user_name:flow_definition
```

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

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

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

```
user_name:flow_definition
```

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

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

For example:

```
numdone(jdoe:payflow) >= 5
```

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

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

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

```
user_name:flow_definition
```

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

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

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

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

```
user_name:flow_definition
```

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

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

For example:

```
umexit(jdoe:payflow) >= 3
```

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

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

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

```
user_name:flow_definition
```
If you do not specify a user name, it defaults to your own.

Valid operators are $\geq$, $>$, $\leq$, $\leq$, $\neq$, $\equiv$.

For example:

```
exit(jdoe:payflow)$\geq$2
```

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

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

```
-C combination_type
```

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

**AND**

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

**OR**

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

```
flow_file_name
```

Specifies the name of the file containing the flow definition.

```
-h
```

Prints the command usage to stderr and exits.

```
-v
```

Prints the Process Manager release version to stderr and exits.

**EXAMPLES**

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

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

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

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

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

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

- The size of /data/tmp.log exceeds 3.5 MB
The file /tmp/1.dat arrives
Any triggering information defined within the flow definition is overwritten. If this flow definition already exists, create a duplicate.
**jtrigger**

manually triggers a previously submitted flow definition.

**SYNOPSIS**

```
jtrigger [-u user_name] [-v "var=value[;var1=value1;...]" flow_name[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 the flow definition is on hold, you can use this command to trigger a flow. If the flow definition is not on hold, this command triggers an additional execution of the flow. If you want to trigger a flow whose definition is not yet stored in Process Manager, use the `jrun` command.

**OPTIONS**

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

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

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

- `-h`
  Prints the command usage to `stderr` and exits.

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

**EXAMPLES**

```
% jtrigger myflow
Triggers the flow definition myflow, which is owned by the current user.
% jtrigger -u "user01" payupdt
Triggers the flow definition payupdt, which is owned by user01.
% jtrigger -v "PMONTH=October" payflow
Triggers the flow definition payflow, which is owned by the current user, and passes it a value of October for the variable PMONTH.
```
SEE ALSO

jrun
Symbols
%I 50
%J 50
( calendar operator 31
) calendar operator 31

A
alarm exception handler 100
alarms
description 17
enabling new 213
listing open 177
raising when exception occurs 109
refreshing list 109
reloading 213
viewing list of 138
viewing open 110
arrays
defining 43
description 43
inserting 43
using variables 44
arrays See also job arrays
attributes
doing a flow, defining 87, 127
automatic trigger 112

B
black border, meaning of 149
blue border, meaning of 149
built-in variables 50
By Event view 138
By State view 137
By User view 136

C
caleditor command 172
Calendar Editor
description 12
starting 13
toolbar 25
user interface 24
calendar expressions, nesting 31
calendars
built-in 22
changing calendar combinations 34
changing dates 34
changing the expression 34
combining dates 31
calendars (continued)
creating 180
for today 26
repeating 27
with complex expressions 30
with specific dates 26
daily, creating 27
deleting 35, 187
description 16
ing 34, 188
eading 32
excluding dates 31
expressions, creating 31
Holidays, limitation 23
how used by LSF 22
intersecting dates 31
list of 24
listing 185
naming, reserved words 183
operators, what they mean 31
shared 22
system
monthly 23
weekly 23
yearly 23
system owned 22
types of 22
weekly, built-in 22
weekly, creating 27
where stored 16
with random dates 26
yearly, creating 27
Cannot Run exception 97
description 97
color, used in Flow Manager 149
command syntax 8
commands
caleditor 172
floweditor 173
flowmanager 174
jalarms 177
jcadd 180
jcals 185
jcdel 187
jcm 188
jcomplete 192
jdefs 194
jflows 197
jid 200
jhold 164, 205
jid 206
jjob 207
commands (continued)
jkill 209
jmanualls 211
jreconfigalarm 213
jreleasue 214
jremove 215
jrerun 216
jresume 217
jrun 132, 219
jsetvars 220
jsinstall 221
jstop 222
jsub 114, 116, 120, 123, 124, 224
jtrigger 231
list of calendar 171
list of calendars 171
list of flow definition 171
list of flow monitor and control 171
pre-execution 80
running 77
completion of flow, specifying 90, 130
connection, verifying to host 206
CPU time, specifying maximum 82

D

daemons, controlling 176
data
   real-time, refreshing 139
   setting refresh rate 139
date dependencies, defining 64
dates
   combining multiple calendars 31
   excluding 31
   intersecting two sets 31
definition of flow, saving 93, 131
dependencies
   all must be met 75
   at least one must be met 75
   bypassing 151
   completing manually 151
defining for job 58
description 14
djob, description 14
dmultiple 75
on a file 60
on date and time 64
on job array 66
on job arrays 66
on proxy flow 73
on proxy job 70
on proxy job array 71
on proxy subflow 72
on subflows
   all jobs end 68
   number of jobs started 69
   number of successful jobs 69
   number of unsuccessful jobs 69
   sum of exit codes 69
dtype of 68
dtypes of for job arrays 66
what it looks like 14
descriptions, for a flow 92, 125

E

days
   changing the label 63
description 15
definition 139
defining for job 58
defining for job 60
defining for proxy job 60
defining for proxy job array 60
defining for proxy subflow 60
defining for subflows 60
defining multiple times to trigger 117
defining types of 15

description 14
depending on 14
depending on another calendar 31
dependencies 75
dependencies, all must be met 75
dependencies, at least one must be met 75

descriptions, for a flow 92, 125

descriptions, for a flow 92, 125

e-mails
   sending automatically 125
turning off 126

dependencies, file, specifying 78
depending on 14
descriptions, for a flow 92, 125
dependencies

events
   changing the label 63
description 15
definition 139
defining for job 58
defining for job 60
defining for proxy job 60
defining for proxy job array 60
defining for proxy subflow 60
defining for subflows 60
defining multiple times to trigger 117
defining types of 15
defining view flows by 138
defining exception handler, description 16
defining exception handlers
   behavior when used 101
built-in 100
user defined 100

definition of flow, saving 93, 131
dependencies
   all must be met 75
   at least one must be met 75
   bypassing 151
   completing manually 151
defining for job 58
description 14
dmultiple 75
on a file 60
on date and time 64
on job array 66
on job arrays 66
on proxy flow 73
on proxy job 70
on proxy job array 71
on proxy subflow 72
on subflows
   all jobs end 68
   number of jobs started 69
   number of successful jobs 69
   number of unsuccessful jobs 69
   sum of exit codes 69
type of 68
dtypes of for job arrays 66
what it looks like 14
descriptions, for a flow 92, 125

E

days
   changing the label 63
description 15
definition 139
defining for job 58
defining for job 60
defining for proxy job 60
defining for proxy job array 60
defining for proxy subflow 60
defining for subflows 60
defining multiple times to trigger 117
defining types of 15
defining view flows by 138
defining exception handler, description 16
defining exception handlers
   behavior when used 101
built-in 100
user defined 100

definition of flow, saving 93, 131
dependencies
   all must be met 75
   at least one must be met 75
   bypassing 151
   completing manually 151
defining for job 58
description 14
dmultiple 75
on a file 60
on date and time 64
on job array 66
on job arrays 66
on proxy flow 73
on proxy job 70
on proxy job array 71
on proxy subflow 72
on subflows
   all jobs end 68
   number of jobs started 69
   number of successful jobs 69
   number of unsuccessful jobs 69
   sum of exit codes 69
type of 68
dtypes of for job arrays 66
what it looks like 14
descriptions, for a flow 92, 125

E

e-mails
   sending automatically 125
turning off 126

dependencies, file, specifying 78
depending on 14
descriptions, for a flow 92, 125
dependencies

events
   changing the label 63
description 15
definition 139
defining for job 58
defining for job 60
defining for proxy job 60
defining for proxy job array 60
defining for proxy subflow 60
defining for subflows 60
defining multiple times to trigger 117
defining types of 15
defining view flows by 138
defining exception handler, description 16
defining exception handlers
   behavior when used 101
built-in 100
user defined 100

definition of flow, saving 93, 131
dependencies
   all must be met 75
   at least one must be met 75
   bypassing 151
   completing manually 151
defining for job 58
description 14
dmultiple 75
on a file 60
on date and time 64
on job array 66
on job arrays 66
on proxy flow 73
on proxy job 70
on proxy job array 71
on proxy subflow 72
on subflows
   all jobs end 68
   number of jobs started 69
   number of successful jobs 69
   number of unsuccessful jobs 69
   sum of exit codes 69
type of 68
dtypes of for job arrays 66
what it looks like 14
descriptions, for a flow 92, 125

E

e-mails
   sending automatically 125
turning off 126

dependencies, file, specifying 78
depending on 14
descriptions, for a flow 92, 125
dependencies

events
   changing the label 63
description 15
definition 139
defining for job 58
defining for job 60
defining for proxy job 60
defining for proxy job array 60
defining for proxy subflow 60
defining for subflows 60
defining multiple times to trigger 117
defining types of 15
defining view flows by 138
defining exception handler, description 16
defining exception handlers
   behavior when used 101
built-in 100
user defined 100

definition of flow, saving 93, 131
dependencies
   all must be met 75
   at least one must be met 75
   bypassing 151
   completing manually 151
defining for job 58
description 14
dmultiple 75
on a file 60
on date and time 64
on job array 66
on job arrays 66
on proxy flow 73
on proxy job 70
on proxy job array 71
on proxy subflow 72
on subflows
   all jobs end 68
   number of jobs started 69
   number of successful jobs 69
   number of unsuccessful jobs 69
   sum of exit codes 69
type of 68
dtypes of for job arrays 66
what it looks like 14
descriptions, for a flow 92, 125

F

fairshare groups, assigning job to 80
file dependencies
defining 60
specifying 60
file events
  creating 60
  description 16
  frequency within flow 60

file names
  for a flow definition 93, 131
  Windows 17

File Transfer tab 83
FILEEVENT_NAME 50

files
  appending output 84
  creating dependency on 60
  input, output and error 78
  names, wildcard characters 118, 225
  output, transferring after job runs 84
  size limits 82
  standard input, output and error 78
  to trigger flow 118
  transferring 83

flow definitions
  actions 19
  creating 40
  creating simple 42
  defining descriptions 92, 125
  description 15
  displaying history 200
  holding 164, 205
  inserting job array 43
  inserting subflow 46
  listing 194
  methods of defining 40
  preventing from running 205
  releasing from hold 165, 214
  removing 167
  removing from Process Manager 215
  running once immediately 219
  submitting 224
  triggering 231
  viewing in Flow Manager 166
  where stored 19

Flow Editor
  description 12
  design palette 39
  starting 12
  toolbar 39
  user interface 38

Flow Manager
  description 13
  refreshing data 139
  starting 13
  toolbar 139
  user interface 136

floweditor command 173
flowmanager command 174

flows
  about 18
  actions 19
  all jobs must complete successfully 90, 130
  allowing to trigger 151
  completing 87, 127
  completing dependencies 151
  completing individual jobs 154
flows (continued)
  completing jobs 154
  controlling from Action menu 136
  controlling jobs 207
  default completion behavior 87, 127
  default completion criteria 87, 127
  defining attributes 87, 127
  defining recurring 113
  dependencies on 121
  description 15
  determining status of jobs 149
  determining when complete 20
  Done status 19
  Exit status 19
  external to current flow, triggering on 73
  filtering
    by user 141
    for last hours 141, 142
  forcing job complete 207
  handling exceptions automatically 91
  how completed 19
  killing 158, 209
  killing individual jobs 152
  listing 197
    by trigger time 138
  marking manual job complete 192
  notifying owner 125
  one job must end 90, 130
  passing parameters to 144
  preventing multiples 125
  processing 87, 127
  rerunning 161
  rerunning exited 216
  rerunning jobs 153
  rerunning killed 231
  resuming 160
  resuming suspended 217
  running
    exclusively 125
    immediately 132
    multiple times 113
    once 132
    one at a time 125
    when other flow completes 121
    when other flow fails 122
    when proxy job completes 123
  running individual jobs 153
  running when other flow fails 122
  sample, using 41
  saving definition 93, 131
  scheduling 112
  setting exit condition 87, 127
  setting refresh rate in Flow Manager 139
  submitting 113
  suspended, rerunning jobs 153
  suspending 159, 222
  to recover from exceptions 101
  triggering 231
    at particular time 115
    automatically 112
    based on file 118
    file arrives 119
flows (continued)
  triggering 231
    file does not exist 119
    file exists 119
    file reaches age 119
    file reaches size 119
    manually 112, 144
    with file 118
    with single time event 115
    with variables 144
  turning off notification 126
  viewing 145
    definition 166
    events that trigger 147
    history 162
    history of 163
    history of definition 162
    while running 145
    with manual trigger 112, 114, 144
    with no schedule 112

G
  General tab 76
  global variables, user 55
  green border, meaning of 149
  grey border, meaning of 149
  group, fairshare 80

H
  history
    displaying 200
    of a flow, viewing 162
  hold, description 18
  host, verifying connection 206
  host limits, specifying 82
  host types, specifying 81
  hosts
    moving to another on failure 79
    running exclusively 79
    running on same as another job 80
    specifying to run on 79

I
  In common with, calendar operator 31
  index expressions, creating 43
  input file, specifying 78
  installation, verifying connection 206
  instructions, adding to job 86

J
  jadmin command 176
  jalarms command 177
  jcadd command 180
  jcals command 185
  jcdel command 187
  jcmod command 188
  jcomplete command 157, 192
  jdefs command 194
  jflows command 197
  jhist command 200
  jhold command 205
  jid command 206
  jjob command 207
  jkill command 158, 209
  jmanuals command 155, 211
  job arrays
    creating index expression 43
    defining 43
    dependency, specifying 66
    description 14, 43
    displaying history 200
    external to current flow, triggering on 71
    handling exceptions 104
    inserting in drawing 43
    runtime attributes, viewing 145
    undefined 40
    using variables 44
    viewing history 163
    viewing state of 149
  job definitions, viewing in Flow Manager 145
  job dependencies
    completing manually 151
    defining 58
    description 14
    ignoring 151
  job flow See flow
  jobs
    actions 19
    appending output file 84
    appending output to local file 84
    assigning priority 80
    assigning to fairshare group 80
    border colors, meaning of 149
    completing 154, 207
    controlling 207
    creating dependencies on 58
    creating manual 47
    creating time dependencies 64
    defining 76
    defining simple 42
    dependency on 123
    dependency on job array 66
    dependency on subflow
      failed jobs 69
      number of jobs started 69
      number of successful jobs 69
      sum of exit codes 69
    description 14
    determining status 149
    displaying history 200
    drawing on workspace 42
    exit, impact on flow 20
    external to current flow, triggering on 70
    forcing complete 207
    handling exceptions 104
    holding on submission 77
    in subflow, viewing 146
    killing 152
    manual, description 15
    marking complete 207
    naming 76
jobs (continued)
notifying user
- when ends 78
- when starts 78
- when starts and ends 78
preventing from running 77
red border around 40
rerunning 153
rerunning in suspended flow 153
rerunning on another host 79
running 153
- as part of a project 77
- at specific time 64
- based on file 60
- command before job 80
- under another user ID 78
running when a job array ends 67
running when a job array has an exit code 67
running when another job has exit code 59
- starts 59
running when number of jobs in job array have started 67
running when number of jobs in job array is unsuccessful 67
running when successful job array jobs is 67
running with exclusive use of host 79
runtime attributes, viewing 145
specifying
- command to run 77
- details 76
- host 79
- host limits 82
- host type 81
- input, output, error files 78
- instructions 86
- limits 82
- login shell 77
- number of licenses required 81
- number of processors 80
- resources required 81
- swap space 82
submitting on hold 77
submitting to queue 77
to recover from exceptions 101
transferring local files for 83
transferring output files after use 84
triggering on proxy flow 73
triggering on proxy job 70
triggering on proxy job array 71
triggering on proxy subflow 72
undefined 40
viewing history 163
viewing state of 149
jreconfigalarm command 213
jrelease command 165, 214
jremove command 167, 215
jrerun command 161, 216
jresume command 160, 217
jrun command 219
JS_FLOW_ID 50, 53
JS_FLOW_NAME 50, 53
JS_ITERATION_COUNTER 50, 53
jsetvars command 220
jinstall command 221
jstop command 159, 222
js command 224
jtrigger command 144, 231
K
Kill exception handler 100
L
labels
- on events, changing 63
legend, description 136
licenses, specifying number required 81
limitations, for a job 82
Limits tab 82
link events, description 16
local variables 55
- setting at runtime 220
login shell, specifying 77
Isfinstall 221
M
manual jobs
- completing 155, 192
- defining 47
- description 15
- displaying list of 155
- displaying uncompleted 211
- viewing list of 139
manual trigger 112
maximum CPU time, specifying 82
maximum run time, specifying 82
menus, what they affect 136
Misschedule exception 96
- description 96
multiple dependencies, defining 75
N
name of job 76
Not, calendar operator 31
Not in common with, calendar operator 31
notification
- on state of flow 125
- stopping 126
number of jobs in flow, how calculated 123
O
operators, meaning of 31
orange border, meaning of 149
output files
- specifying 78
- transferring after job runs 84
Overrun exception 96
- description 96
P
parallel jobs, specifying number of processors 80
parameters, passing to flow 144
Plus calendar operator 31
priorities, assigning for a job 80
Process Manager status, viewing 138
Processing tab 79
processors, specifying number for parallel jobs 80
projects, specifying for job 77
proxy, defining dependency on 70, 71, 72, 73
proxy events
description 16
duration 121, 124
Q
queues, specifying 77
R
recovery flow exception handler 101
recovery job exception handler 101
red border, meaning of 149
red job in Flow Editor 40
refresh rate, setting 139
release, description 18
remove, description 18
Rerun exception handler 100
resources, specifying for job 81
Resources tab 81
run time, specifying maximum 82
runtime attributes, viewing 145, 150
S
sample flows, using 41
server
displaying host name 206
installing 221
Server status, viewing 138
setting, in Windows 56
Start Failed exception 96
description 96
status, viewing 138
subflows
creating dependencies on 68
dependencies
on all jobs ending 68
on exit code 69
on number of jobs started 69
on number of successful jobs 69
on number of unsuccessful jobs 69
description 15
displaying history 200
external to current flow, triggering on 72
handling exceptions 105
inserting into flow definition 46
runtime attributes, viewing 145
viewing exit conditions 146
viewing exit conditions of 146
viewing jobs in 146
viewing state of 149
submit, definition 18
swap space, specifying 82
system calendars 22
absolute dates 181, 189
creating expressions 181, 188
merging expressions 181, 189
recurring daily 181, 189
recurring monthly 182, 189
recurring weekly 181, 189
recurring yearly 182, 190
T
time dependencies
creating 64
run after a time 64
run before a time 64
time events
description 16
frequency within flow 64
single 115
types of 64
time expression, specifying 117
toolbars
Calendar Editor 25
for Calendar Editor 25
for Flow Editor 39
for Flow Manager 139
trigger, description 19
triggers, manual 112, 114
typographic conventions 8
U
Underrun exception, description 96
UNIX, wildcard behavior 225
user ID, changing for job 78
user variables 56
before using 55
global 55
local 55
multiple 55
queues 55
setting, in UNIX 55
using 57
valid fields 56
users, notifying when job runs 78
V
variables
built-in 50
description 17
in array index expression 44
local, setting values for at runtime 220
setting
in UNIX 55
in Windows 56
restriction 56
triggering flows with
user
before using 55
global 55
local 55
multiples 55
queues 55
variables (continued)
  user
    using 57
  valid fields 56
  when evaluated 57
views, description 138

W
warnings, deleting calendars 35
wildcard characters 118, 225
difference between UNIX and Windows 225
Windows, wildcard behavior 225

Y
yellow border, meaning of 149