

Flo for Spring XD


Documentation

Published: 14 March 2019

Table of Contents

Table of Contents	2
Flo for Spring XD	3
Installing Flo	5
Guidelines for Building Composed Jobs	6
Providing Feedback, Requesting Features, and Reporting Bugs	14

Flo for Spring XD

 **Note:** Flo for Spring XD is no longer supported. To stay up to date with the latest software and security updates, upgrade to a supported version.

Flo is an HTML5 canvas application that runs on the Spring XD runtime, offering a graphical interface for creating and monitoring streaming and batch data pipelines.

Drag and Drop Interface

Flo allows you to create real-time streaming and batch pipelines either with the textual shell or the ‘drag and drop’ interface. For more information, see the [Building Streams](#) and [Advanced Stream Creation](#).

Flo and Streaming Pipelines

The textual shell offers content-assist for stream definitions. Whether it is a source, processor, sink, or job, content-assist offers completion of module names and module options. This allows for rapid development, and is similar to the content-assist experience when writing code in an IDE. For more information, see the [Content Assist and Validation](#) tutorial video.

The textual and graphical views are kept in sync automatically. While working on a stream definition you can switch them and still find an up-to-date representation of your pipelines.

Flo includes a palette with all out-of-the-box adapters available in Spring XD. The palette also includes a smart search filter for quick module lookup. Once you find your desired module, you can drag and drop the module to the canvas to create a new definition or augment an existing one.

Pipelines are composed of source, processor (0..n), and sink modules. Flo expects this structure and offers immediate feedback on any issues it finds. The textual shell displays error messages for syntax errors and other problems with stream definitions, such as if a module is specified with an unsupported option.

Health Monitoring

Spring XD uses Spring Boot’s monitoring and management support over HTTP and JMX along with Spring Integration’s MBean Exporters. Flo leverages these endpoints to provide visual health monitoring.

The states of pipelines in Spring XD are represented visually by different colors. Green, orange, grey, and red indicate `Deployed`, `Incomplete`, `Undeployed`, and `Failed`, respectively.

Small colored circles under each module represent the states of that module’s instances. White indicates that an instance is waiting to be deployed, and green indicates successful deployment. Hovering over a circle provides information specific to the corresponding instance, such as where it is running. If you request a large number of instances in this way, the view switches from circles to a simple numeric summary of how many are successfully deployed versus how many have been requested.

Metrics

Flo overlays the stream view of each module with the combined message throughput for all instances of that module. Where applicable, the input rate is displayed to the left of the module and the output rate is displayed on the right.

For more information, see the [Health Metrics](#) tutorial video.

Flo and Batch Pipelines

Flo also offers a textual shell and graphical view for batch pipelines. As with the textual shell and graphical view for streaming pipelines, they are kept in sync automatically. Flo for Batch pipelines in Spring XD is available under **Jobs > Create Composed Job**.

Unlike the palette for streaming pipelines, the palette for batch pipelines does not include out-of-the-box job modules. Job definitions for batch pipelines must first be created, and are then available in the palette, along with any already-defined composed jobs.

Flo for Batch pipeline builds upon the newly supported Batch DSL in Spring XD, which can be used to create composite batch workflows involving sequential jobs, parallel jobs or even the combination of the two.


The Flo canvas textual shell and drag-and-drop interface both support the Spring XD batch DSL. Job definitions in the DSL will be computed automatically for batch jobs composed in the drag-and-drop interface.


Existing composite batch workflows can be viewed by clicking **Jobs > Definitions**. In future releases, Flo will include comprehensive features for batch pipelines such as metrics, statuses and distributed tracing of batch jobs.

Installing Flo

This topic provides instructions for installing Flo for Spring XD.

1. Install or update to the latest Spring XD GA release, or the release of your choice.
2. Visit the [Flo product page](#) and download `flo-spring-xd-admin-ui-client-[RELEASE_OF_YOUR_CHOICE].zip`.
3. Navigate to the `XD_HOME/lib` directory and back up your original `spring-xd-admin-ui-client-[RELEASE].jar` file.
4. While still in the `XD_HOME/lib` directory, unzip `flo-spring-xd-admin-ui-client-[RELEASE_OF_YOUR_CHOICE].zip` to replace the existing `spring-xd-admin-ui-client-[RELEASE].jar`.
5. Clear your browser cache.
6. Access Flo for Spring XD at the following URI endpoints:
 - Main dashboard: `http://HOST_NAME:PORT/admin-ui`
 - Stream creation: `http://HOST_NAME:PORT/admin-ui/#/streams/create`
 - Health monitoring: `http://HOST_NAME:PORT/admin-ui/#/streams/definitions`
 - Module throughput rates are visible at `http://HOST_NAME:PORT/admin-ui/#/streams/definitions`.

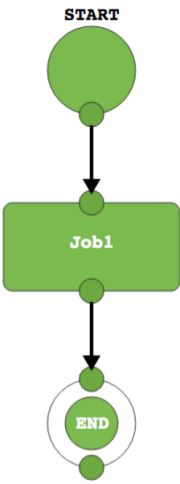
 **Note:** If unchanged, the default PORT points to 9393.

 **Note:** To see throughput rates for modules, you must enable JMX in your Spring XD cluster. By default it is disabled.

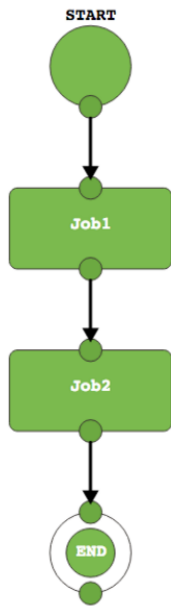
Guidelines for Building Composed Jobs

This topic provides instructions for building composed jobs.

- The basic building block is a flow, which is a sequence of one or more jobs.
- Flows are self contained and there is effectively one entry point and all exits from a flow exit to the same next target.
- You can link the end of one flow to the start of a new flow but you can't link between intermediate nodes in two flows.
- A split indicates a number of flows run in parallel.
- A composed job can specify a timeout, this is a global setting for the job. To set a global setting for a job open the properties for the START node and set it there.
- Although jobs in a flow are expressed such that one runs after another, a conditional transition can be setup to cause something different to happen when a particular job finishes. The conditional transition specifies the Job Exit Status which triggers it and then what happens next: this might be another job or it might be immediately ENDing or FAILing that flow.
- The wildcard "*" value for a conditional transition job exit status represents everything not mapped by other transitions from the same job.
- For the last job in a flow, if the exit space is being mapped in any way (if there are any conditional transitions out of it) then it is important to map the entire exit space by ensuring you have a conditional transition for "*" (meaning 'any exit status').
- If your graph looks like spaghetti, consider decomposing it further. You can build a simple composed job then use that as a building block in another composed job.

Basic composed job	
 <pre> graph TD START((START)) --> Job1[Job1] Job1 --> END((END)) END --> END </pre>	<p>A very basic composed job consisting of a single job.</p> <p>DSL definition: Job1</p>

Multiple jobs

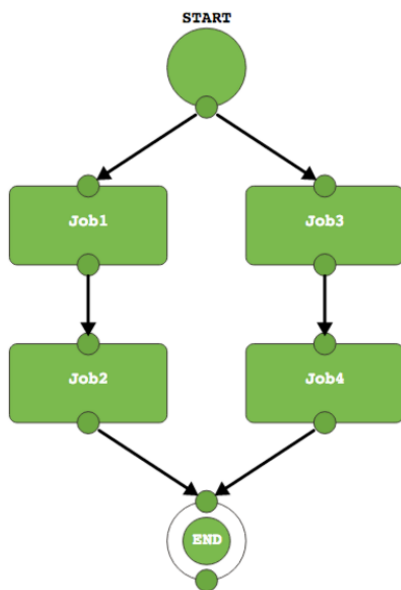


Composed job consisting of two jobs. Job1 runs then Job2 runs.

DSL definition:

Job1 || Job2

Simple Split

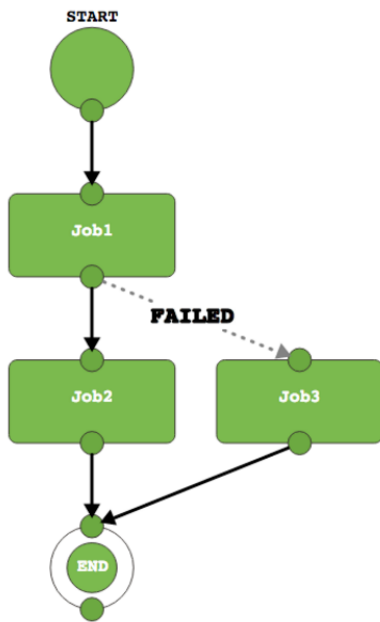


There are two flows here. The first flow is Job1 and Job2. The second flow is Job3 and Job4. These two flows are run in parallel. When Job3 and Job4 complete, the entire composed job completes.

DSL definition:

<Job1 || Job2 & Job3 || Job4>

Composed job with conditional transition

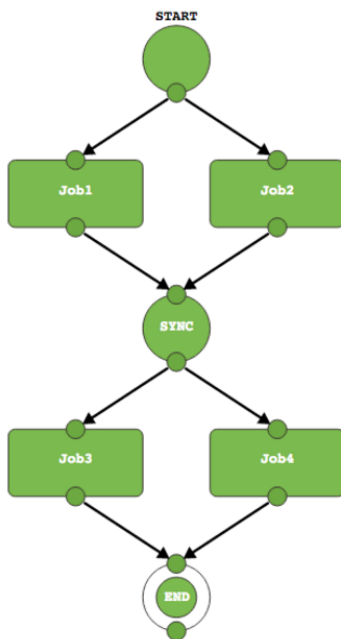


When Job1 finishes, if it exits normally then Job2 will run. If Job1 exits with status FAILED, processing will continue with Job3. When either Job2 or Job3 completes (whichever was chosen), the entire composed job completes.

DSL definition:

```
Job1 | FAILED=Job3 || Job2
```

Double Splits

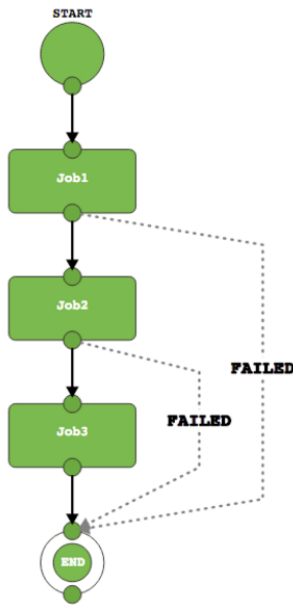


This contains two splits. The first split will run Job1 and Job2 in parallel. The second split will run Job3 and Job4 in parallel. The SYNC node is used to join the threads of execution from the first split before they branch out into the secondary flow. Without SYNC it isn't clear what you could connect the output of Job1 and Job2 to in order to achieve this. There is no real job called SYNC, it is merely a special control node in the graph.

DSL definition:

```
<Job1 & Job2> || <Job3 & Job4>
```


Flow with multiple exit points

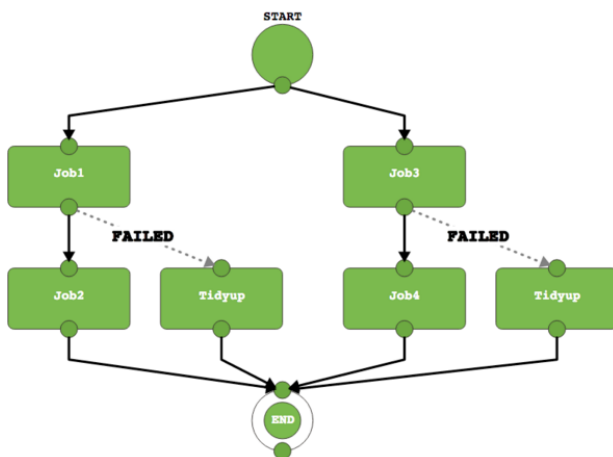


Here it is a basic job sequence (Job1 then Job2 then Job3) but if either Job1 or Job2 exit with a FAILED job exit status, the flow terminates early (skipping any remaining jobs).

DSL definition:

```
Job1 | FAILED=$END || Job2 | FAILED=$END ||
Job3
```

Parallel flows with transitions

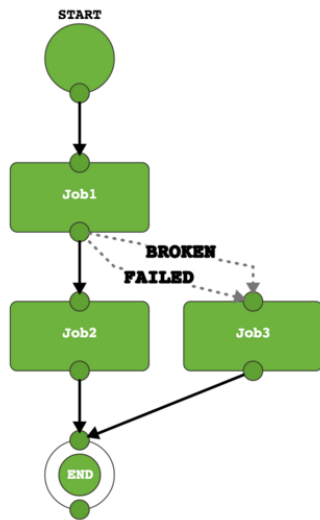


Here is a split with two flows running in parallel. The first job in each flow may exit in a FAILED state in which case the Tidyup job will run rather than the secondary job in each case.

DSL definition:

```
<Job1 | FAILED=Tidyup || Job2 & Job3 |
FAILED=Tidyup || Job4>
```

Multiple transitions to same target

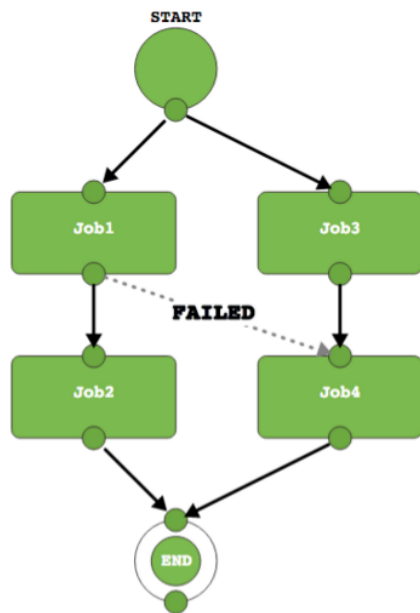


If Job1 exits with a status of BROKEN or a status of FAILED then Job3 will run, otherwise Job2 will run.

DSL definition:

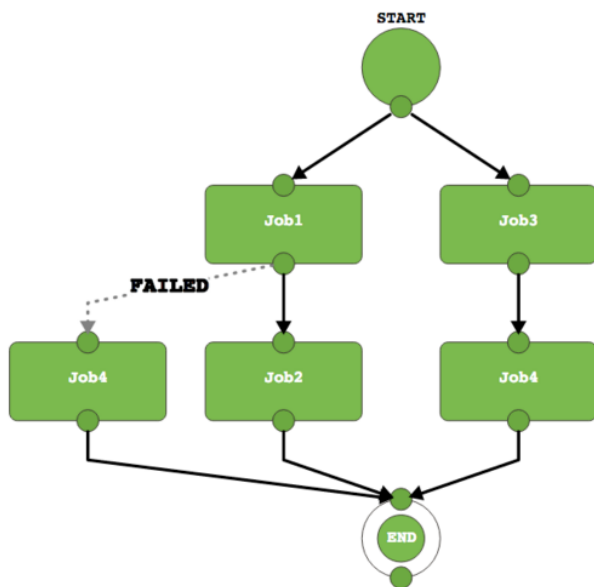
```
Job1 | BROKEN=Job3 | FAILED=Job3 || Job2
```

Parallel flows with transitions



This graph is not valid, it violates a rule - it is not possible to link across flows. This is a split with two parallel flows running at the same time. The flows should be self contained.

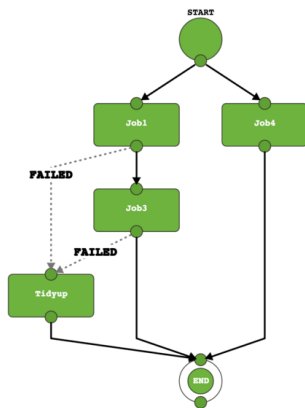
DSL definition:
not valid



This is the correct version. There is still a split and the left and right sides are self contained flows.

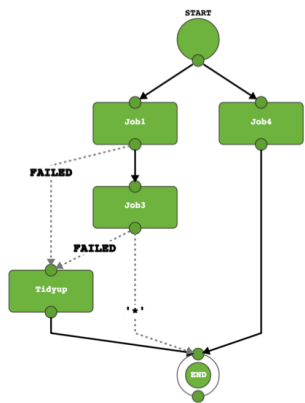
DSL definition:
<Job1 | FAILED=Job4 || Job2 & Job3 || Job4>

Mapping the exit space



This graph is not valid. There is a split and on the left is a flow consisting of Job1 and Job3. Because the exit space of Job3 is partially mapped (mapping FAILED to Tidyup) the rest of it must also be mapped using the wildcard transition rather than the solid line.

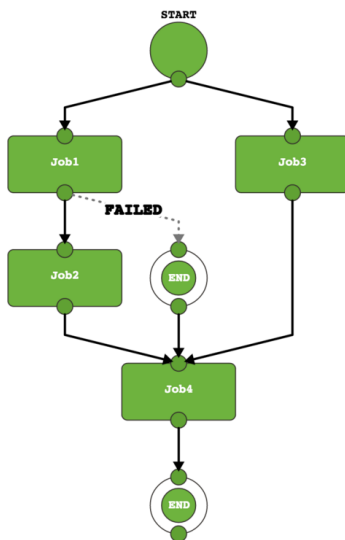
DSL definition:
not valid



This is the correct version. The exit space of Job3 has been completely mapped by setting '*' on the connection from Job3 to END.

DSL definition:
<Job1 | FAILED= Tidyup || Job3 | FAILED=Tidyup
| '*'=\$END & Job2>

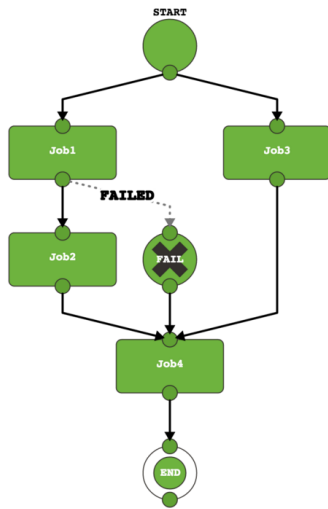
Finishing a flow early



The END node can be used to indicate the end of the entire job or the end of a particular flow. Here there is a split running Job1/Job2 in parallel with Job3. However, if Job1 exits with a FAILED exit status, the flow immediately ends (indicated by the END node) and when this flow finishes, or when the parallel flow containing Job3 finishes, then Job4 will run.

DSL definition:
<Job1 | FAILED= \$END || Job3 & Job2> || Job4

Failing a flow early



The FAIL node can be used to indicate earlier failure of a flow. Here there is a split running Job1/Job2 in parallel with Job3. However, if Job1 exits with a FAILED exit status, the flow immediately ends (indicated by the FAIL node). When this flow finishes either because Job2 finishes, or the FAIL node is hit, or Job3 finishes, then Job4 will run. Note, using the FAIL node does not mean the whole job fails, only that flow.

DSL definition:

```
<Job1 | FAILED =$FAIL || Job3 & Job2> || Job4
```

Providing Feedback, Requesting Features, and Reporting Bugs

Use JIRA to provide feedback, request a feature, or report a bug. Ensure that you enter the following information in any [Spring XD JIRA ticket](#) that you create:

- **Epic Link:** Flo for Spring XD
- **Summary:** [Flo] My issue...
- **Description:** As a Flo for Spring XD user, I...

Issue Type*

Bug

?

Epic Link

Flo for Spring XD

Choose an epic to assign this issue to.

Summary*

[Flo] ..

Priority

Minor

?

Component/s

Start typing to get a list of possible matches or press down to select.

Affects Version/s

1.2 GA x

Start typing to get a list of possible matches or press down to select.

Story Points*

Measurement of complexity and/or size of a requirement.

Reporter*

sabby

Type username of the reporter.

Description

Style

B

I

U

A

A

As a Flo for Spring XD user, I ..