Frequently Asked Questions

When will Batch 1.0 be out? My understanding is that the current product can be used on projects, is that correct?

We expect rc1 before 1.0 final, so realistically not soon. You can track the progress and planning in JIRA (http://opensource.atlassian.com/projects/spring/browse/BATCH). But it is already usable at m3 as long as there is an understanding that upgrading might take a small amount of effort - there are 5 or 6 projects using it, with at least one in production. That project upgraded from m1 to m2 in less than a day. We intend to keep changes from here to 1.0 as close to internal as possible, so the client touch points are pretty stable.

[top]

How stable are the interfaces in Spring Batch?

We are still in the milestone release phase (1.0-m5 is just out). This means that we are still adding functionality that we want to be part of a 1.0 release. We do not rule out changes to package and interface names in this phase, but that said we think the basic domain concepts in Spring Batch are sound enough to survive significant re-factoring. The bulk of the application developer "touch points" have been stable for quite some time now, and we have several early adopter projects already using snapshot releases.

The process from here is to collect feedback from the community and use that to decide on what extra features need to be added to get us to 1.0. When we are feature complete we will move to the "release candidate" phase, and the first release in that phase will be 1.0-rc1. We only expect one release candidate, but if there is enough demand for new features, or significant problems occur in rc1, then we might need an rc2. We aim for 2 weeks elapsed time between release candidates (and between the last milestone and rc1).

[top]

When will support for the more complex job execution classes appear? Our Client has a number of multi-threaded batch jobs.

Multi-threaded execution in a single VM is perfectly possible with 1.0 - but we recommend exercising caution in the analysis of such requirements (is it really necessary?). Several people have tried it and it works as long as the step is intrinsically restartable (idempotent effectively). The parellel job sample shows how it might work in practice - this uses a "process indicator" pattern to mark input records as complete, inside the business transaction.

[top]

There are 4 main layers of the architecture (application, core, execution, and infrastructure), what is the vision for how the execution layer might be used in future? What happened to the "container" layer?

The "layers" described are nicely segregated in terms of dependency. Each layer only depends (at compile time) on layers below it.

We recognised that what we used to call the container layer actually is composed of two distinct contexts, "Core" and "Execution". So the full catalogue of contexts is:

- **Application** is the business logic. It is written by the application developer - the client of Spring Batch - and only depends on the other Core interfaces for compilation and configuration.
- **Core** is the public API of Spring Batch, including the core batch domain of Job, Step, configuration and Executor interfaces.
- **Execution** is the deployment, execution and management concerns. Different execution environments (e.g. in a JEE container, out of container) are configured differently, but can execute the same application business logic.
- **Infrastructure** is a set of low level tools, that are used to implement the execution and parts of the core layers.

The "execution" layer is fertile ground for collaboration and contributions from the community and from projects in the field. The central interface is `JobLauncher` with methods for starting and stopping jobs. The vision for this is that there can be multiple implementations of `JobLauncher` providing different architectural patterns, and delivering different levels of scalability and robustness, without changing either the business logic or the job configuration.

[top]

What is the Spring Batch philosophy on the use of flexible strategies and default implementations?

There are a great many extension points in Spring Batch for the framework developer (as opposed to the implementor of business logic). We expect clients to create their own more specific strategies that can be plugged in to control things like commit intervals (`CompletionPolicy`), rules about how to deal with exceptions (`ExceptionHandler`), and many others.

[top]

How does Spring Batch differ from Quartz? Is there a place for them both in a solution?

Spring Batch and Quartz have different goals. Spring Batch provides functionality for processing large volumes of data and Quartz provides functionality for scheduling tasks. So Quartz could complement Spring Batch, but are not excluding technologies. A common combination would be to use Quartz as a trigger for a Spring Batch job using a Cron expression and the Spring Core convenience `SchedulerFactoryBean` .

[top]

How do I schedule a job with Spring Batch?

Use a scheduling tool. There are plenty of them out there. Examples: Quartz, Control-M, Autosys. Quartz doesn't have all the features of Control-M or Autosys - it is supposed to be lightweight. If you want something even more lightweight you can just use the OS (cron, at, etc.).

Simple sequential dependencies can be implemented using the job-steps model of Spring Batch. We think this is quite common. And in fact it makes it easier to correct a common mis-use of scehdulers - having hundreds of jobs configured, many of which are not independent, but only depend on one other.

[top]

How will Spring Batch allow project to optimize for performance and scalability (through parallel processing or other)?

We see this as one of the roles of the Execution layer. A specific implementation (or implementations) of the `StepExecutor` can deal with the concern of breaking apart the business logic and sharing it efficiently between parallel processes or processors. There are a number of technologies that could play a role here. The essence is just a set of concurrent remote calls to distributed agents that can handle some business processing. Since the business processing is already typically modularised - e.g. input an item, process it -

Spring Batch can strategise the distribution in a number of ways. One implementation that we have had some experience with (and have a prototype for) is a set of remote EJBs handling the business processing. We switch off Home caching in the container and then send a specific range of primary keys for the inputs to each of a number of remote calls. The same basic strategy would work with any of the Spring Remoting protocols (plain RMI, HttpInvoker, JMS, Hessian etc.) with little more than a couple of lines change in the execution layer configuration.

[top]

What are the key concepts in the Spring Batch core domain?

In a nutshell: A JobConfiguration with a list of StepConfigurations is passed to a JobExecutor. From this a Job is constructed consisting of a series of Steps, each of which is executed by a StepExecutor. The StepExecutor contains all the strategies for deciding when to complete, when to commit, when to abort and when to continue.

Many Jobs in practice consist of a single Step. Step is very useful and best practice for breaking a Job down into logical units, rather than having to execute separate Jobs (potentially in separate OS processes) which have no obvious logical connection.

Jobs can be executed once, or many times with different logical identifiers (JobIdentifier). It is also possible to restart a failed Job with the same or a modified input source, and identify the resulting JobExecution as a separate entity. In this way the progress of a Job and its history of successful and failed executions can easily be tracked. The same argument applies to Steps, which have their corresponding StepExecution entity.

[top]

How can messaging be used to scale batch architectures?

There is a good deal of practical evidence from existing projects that a pipeline approach to batch processing is highly beneficial, leading to resilience and high throughput. We are often faced with mission-critical applications where audit trails are essential, and guaranteed processing is demanded, but where there are extremely tight limits on performance under load, or where high throughput gives a competitive advantage. Matt Welsh's work shows that a Staged Event Driven Architecture (SEDA) has enormous benefits over more rigid processing architectures, and message-oriented middleware (JMS, AQ, MQ, Tibco etc.) gives us a lot of resilience out of the box. There are particular benefits in a system where there is feedback between downstream and upstream stages, so the number of consumers can be adjusted to account for the amount of demand. So how does this fit into Spring Batch? Well it's a good example of an `StepExecutor` or (more broadly) execution runtime if the deployment is grid- or cluster-based, or in any way involves multiple OS processes.

[top]

How can I contribute to Spring Batch?

Use JIRA and the forum to get involved in discussions about the product and its design. There is a process for contributions and eventually becoming a committer. The process is pretty standard for all Apache-licensed projects. You make contributions through JIRA (so sign up now); you assign the copyright of any contributions using a standard Apache-like CLA (see the Apache one for example - ours might be slightly different); when the contributions reach a certain level, or you convince us otherwise that you are going to be committed long term, even if part time, then you can become a committer.

[top]