Recap: Step Definition

Discussion: Job Definition

Discussion: Checkpoint Model

Discussion: CDI

List for Next Meeting
Recap: Step Definition (so far)

- Annotation-based
- @Step at class level
- Callbacks at method level
  - @CreateStep (factory?)
  - @BeginStep/@EndStep
  - @SetProperties
  - @RunStep
  - @GetReturnCode
Things you can specify at step level:

- Name `@Step(name=...)`
- Parallelization `@Step(...,parallel=true)`
- Properties `@Step(...,properties={ ... })`

Not yet addressed:

- Relationship of step to readers/writers
- Relationship of step to checkpoint policy
- Separate discussion required
Annotations or DSL (JCL?) or Both
- Probably XML
- Nothing wrong with both – spec overkill?
- If we picked one, does it prevent the other?
- Pick one and allow implementer freedom?

JCL – internal or external to application?
Discussion: Job Definition

- JCL – internal or external (or both??)

```
<job name="Job1">
<step name="Step1" next="Step2"/>
<step name="Step2"/>
</job>
```

```
<job name="JobX">
<step name="Step1"/>
<step name="Step2"/>
<step name="Step3"/>
</job>
```

“submit” – sends a job.xml to the container to run

“launch” – tells container to run pre-configured job
Container-centric or Reader Centric

Container-centric
- Loop: container<>step
  - Container checkpoints according to policy
- Pros: flexible support of multiple readers/writers
- Cons: non-obvious loop (unnatural)

Reader-centric
- Loop: step<>reader
  - Reader checkpoints according to policy
- Pros: obvious loop (natural)
- Cons: rigid support for multiple readers/writers
Proposal: CDI should be available for developers but not an inherent part of the spec.

Developers can access CDI to instantiate
- Jobs, Steps, Readers/Writers
List for Next Meeting

- **Job Initiation** (submit command, launchers, etc)
- **Step Concurrency**
- **Readers/Writers**
- **Metrics**
- **What else?**