# JSR-352 – The Future of Java Batch and WebSphere Compute Grid

*Session 16384*

*David Follis*

*IBM*


#SHAREorg

SHARE is an independent volunteer-run information technology association that provides **education**, professional **networking** and industry **influence**.

SHARE in Seattle 2015

# Please Note

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion. Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.

The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.

# WebSphere Application Server

| Session | Title | Time | Room | Speaker |
|---|---|---|---|---|
| 16379 | WebSphere Liberty Profle, Windows and z/OS, Hands-on Lab | Monday 4:30 | Redwood | Follis/Stephen |
| 16380 | z/OS Connect: Opening up z/OS Assets to the Cloud and Mobile Worlds | Tuesday 1:45 | Virginia | David Follis |
| 16381 | WebSphere Liberty Profile and Traditional WebSphere Application Server – What's New? | Tuesday 3:15 | University | Follis/Stephen |
| 16509 | Debug 101-Using ISA Tools for Apps in WebSphere Application Server z/OS | Wednesday 3:15 | Virginia | Mike Stephen, Joran Siu |
| 16383 | IBM Installation Manager for z/OS System Programmers: Web-based Installs, Fix Packs, and How iFixes Really Work. | Thursday 8:30 | University | Don Bagwell, Bryant Panyarachun |
| 16384 | JSR 352 - The Future of Java Batch and WebSphere Compute Grid | Thursday 10:00 | University | David Follis |
| 16382 | Common Problems and Other Things You Should Know about WAS on z/OS | Thursday 4:30 | Virginia | Mike Stephen |
| 16385 | Configuring Timeouts for WebSphere Application Server on z/OS | Friday 10:00 | Virginia | Follis/Stephen |

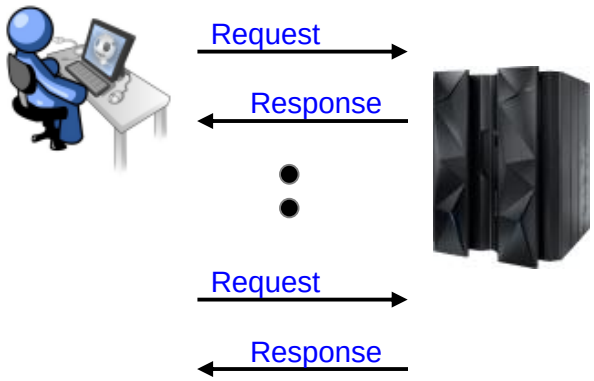
# Overview

Setting the stage for the discussion of modern batch solutions

# Batch (or Bulk) Processing

Many definitions exist ... they all have in common the relative lack of human interaction, and the expectation of results at a future time rather than immediately:

## Online Processing




*In general:*

- Interaction is *one-for-one* ... that is, request with matching response
- Expectation is for response to follow request in a *near-immediate* span time frame

## Bulk Processing


*In general:*

- Interaction is *one-for-many*... that is, initial request results in many results from processing
- Expectation is for results to finish within some determined *non-immediate* time frame

Examples of bulk processing …

# Some Examples of Batch (or Bulk) Processing

Just to set some context for the upcoming discussion

## Month-End Tax or Fee Calculation and Billing

Customer records are processed with tax or other calculations processed against activity. This may be part of a larger process of calculating amount owed and formulating billing.

## Period-End Statements and Reports

An example is investment portfolio results and reporting.

## Data Transformation

Raw data records are processed with goal of transforming some aspect of the data content or layout. Result is a second set of data with the intended formatting.

## Data Analysis

Data records are analyzed to determine trends or patterns. Data mining to find new potential markets is one example. Analyzing vast quantities of seismic data is another.

Point is these are activities where processing in bulk is a better way to maximize efficiency of data access and computer resources. The completion time is determined but not immediate.

# What's Behind This?

Every business has different motivators. The common ones we've seen:

## Batch Window Compression

The window in time for batch processing is shrinking. There's a need to better manage online and batch processing concurrently within the same system.

## Java Skills and Common Tooling

Java skills are more common that traditional programming skills. Further, Java tooling for online work is powerful and capable of being used for batch programming as well.

## GP cycles and Offload to Specialty Engines

Cost pressures are creating a need to explore ways of offloading processing cycles from GP to specialty engines such as zAAP or zAAP-on-zIIP.

Other motivators may exist. The key is that these motivators are real and they are driving exploration of modern batch.

# Approaches to "Batch Modernization"

Generally speaking, we see two basic approaches:

## Preserve Existing ... Java Batch for New

Leave existing batch processes as they are today, but as new requirements come up then engineer them into the Java batch model

## Re-Engineer *Some* Existing Batch Processes

Identify existing non-Java as candidates and then re-engineer them to operate in a Java batch environment

Typical starting approach: identify batch processes with fewer interdependencies and then work out from there

What we don't see is a "rip-and-replace" strategy. That's too costly and too risky. Always a reasoned incremental approach.

# History of Java Batch in WebSphere

## WebSphere has a long history of supporting Batch applications written in Java

- In 2006 the WebSphere Extended Deployment suite of products included “Business Grid”, which provided support for Java Batch applications.
- It was later renamed “Compute Grid” and released as a separate product. The latest release is Compute Grid v8 and it supports WAS v7 and v8.
- In WAS v8.5, the Compute Grid functionality was merged with the Application Server. The functionality today remains the same in both the Compute Grid Feature Pack and WAS v8.5+.
- In 2012, IBM led the development of the Java Batch Open Standard (JSR-352) through the Java Community Process.
  - ✓The JSR-352 specification was approved in 2013
  - ✓IBM contributed the Reference Implementation
- IBM has released beta functionality for JSR-352 **and more** for the WebSphere Liberty Profile.

Statement of Direction

- IBM intends to support the JSR-352 framework, along with additional functionality, in **WebSphere Liberty Profile**
- IBM intends to support the JSR-352 framework, along with additional functionality, in **WebSphere Application Server Full Profile** release.

# Open Standard Java Batch

## A look at the JSR 352 specification

# JSR 352 - "Batch Applications for the Java Platform"

The V1.0 final release is dated April 18th of 2013. IBM served as specification lead on this JSR. It is available for download from the web at the following URL:

`http://jcp.org/aboutJava/communityprocess/final/jsr352/index.html`


The document has a very nice section titled "Domain Language of Batch" which provides an overview of key concepts and key terminology

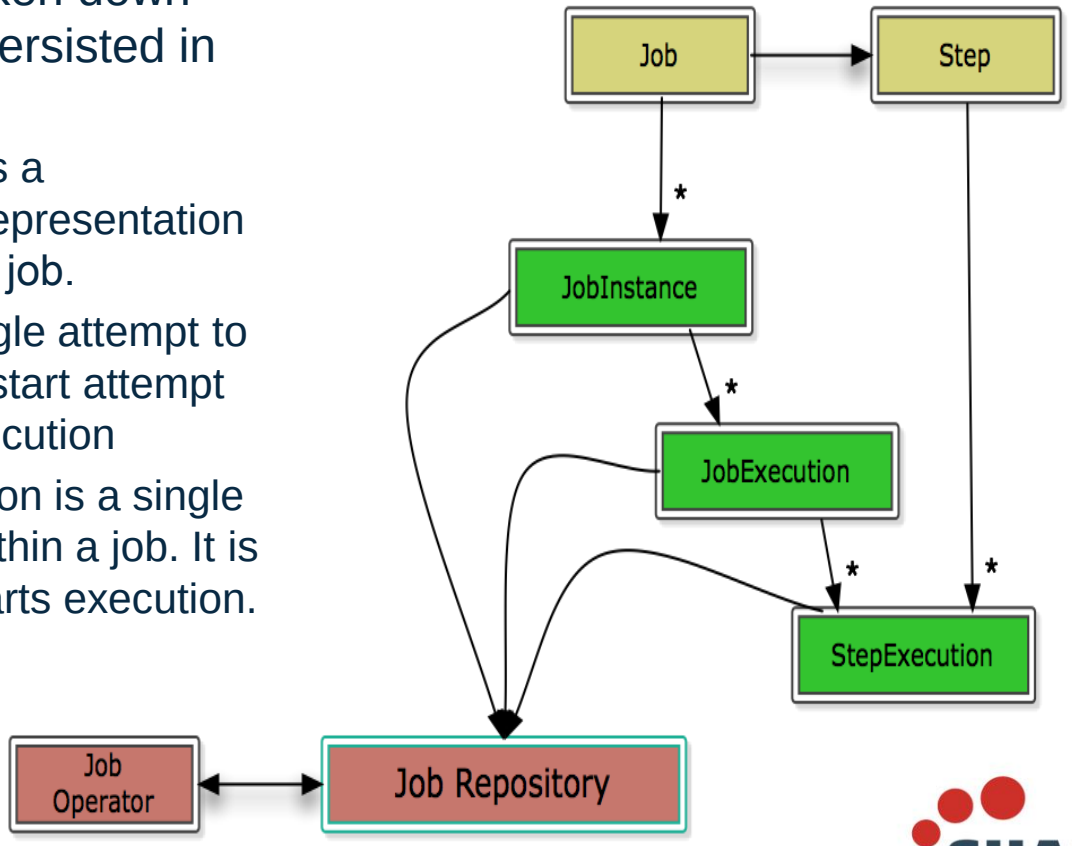
As well as detailed sections on the specification interfaces and other details of the specification

If interested in this JSR, download and review is encouraged

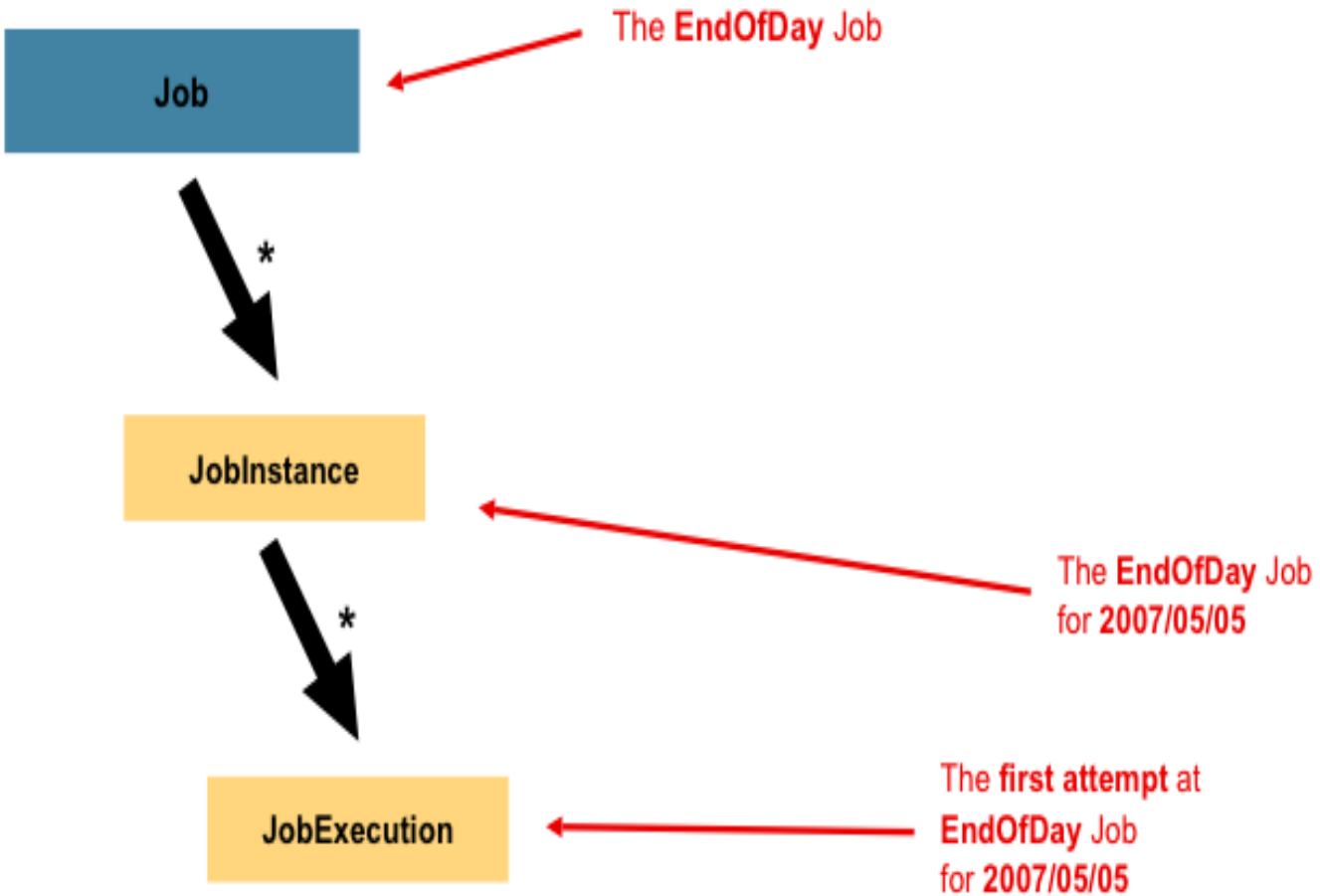
# Fundamental Concepts

# Execution: JobInstance, JobExecution, and StepExecution

- The state of a job is broken down into various parts, and persisted in the repository
  - Submitting a job creates a JobInstance, a logical representation of a particular “run” of a job.
  - A JobExecution is a single attempt to run a JobInstance. A restart attempt creates another JobExecution
  - Similarly, a StepExecution is a single attempt to run a step within a job. It is created when a step starts execution.



# • Job, JobInstance, JobExecution example

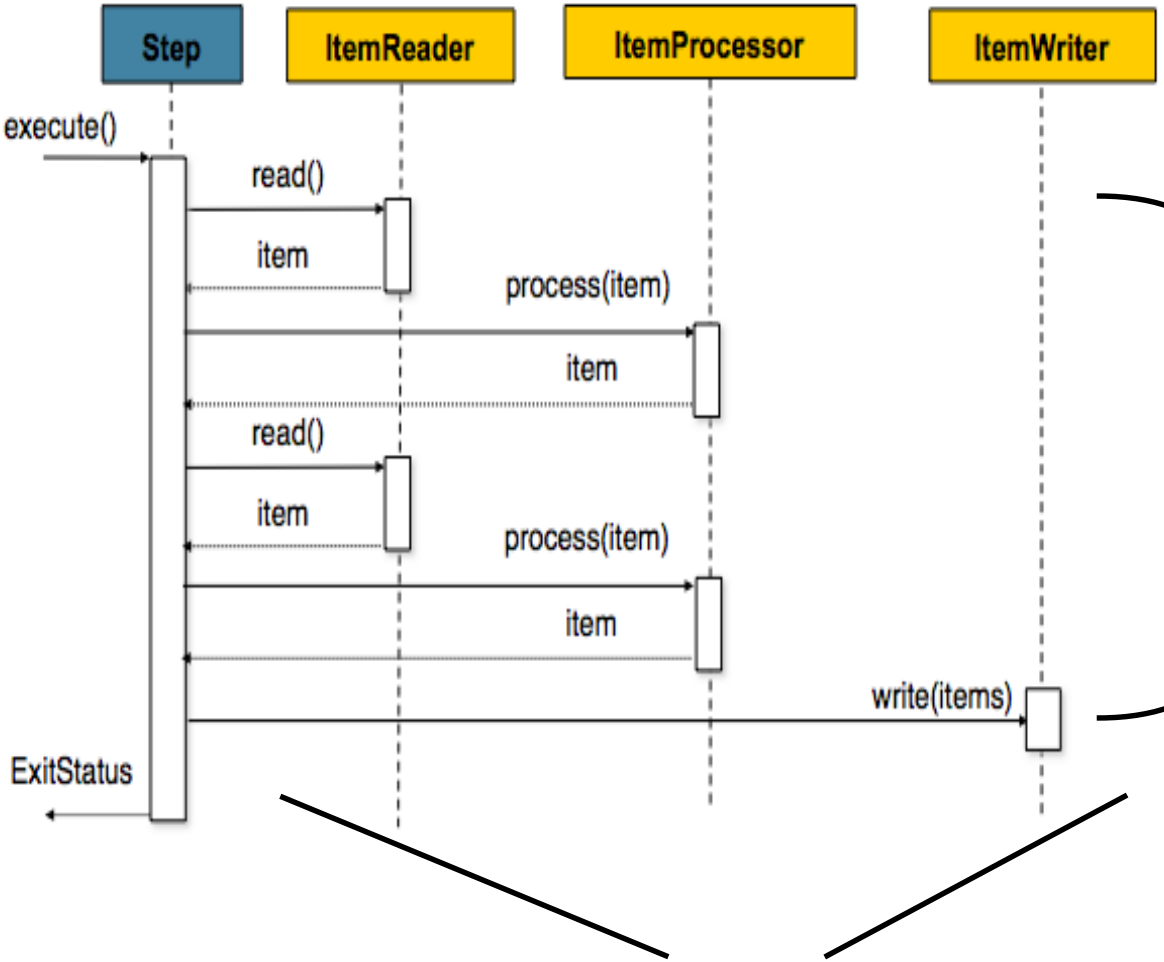


# Facilitates Restart

- Job may not run to completion because of:
  - Invalid data
  - Failed to obtain lock
  - Batch window closed – online work required priority

- On restart, resume where you left off:
  - within the job - don't rerun already completed steps
  - within the step - pick up within data set at last "checkpoint"

# Chunk Loop

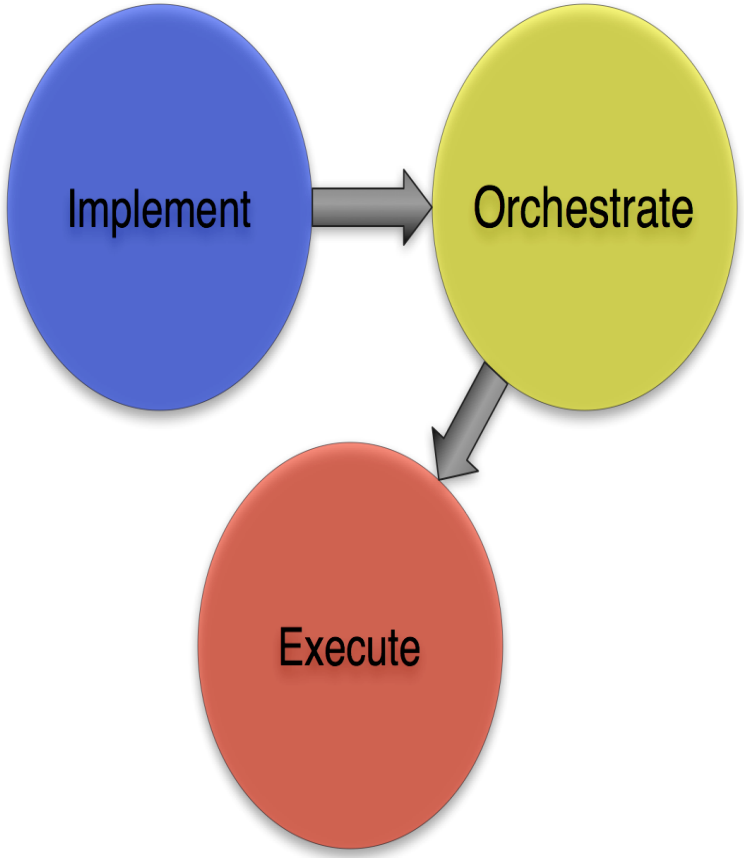



In EE this "chunk" is performed in a single global transaction

Repeat chunk until reader says no more data

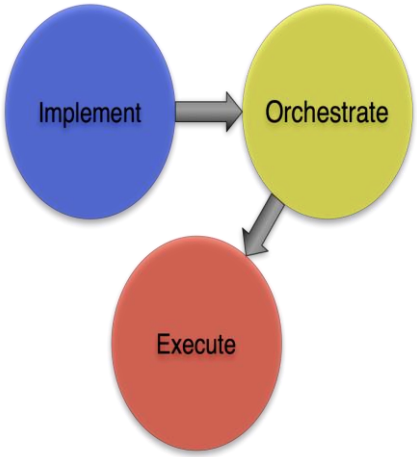
# Three Key Concepts (Roles) …

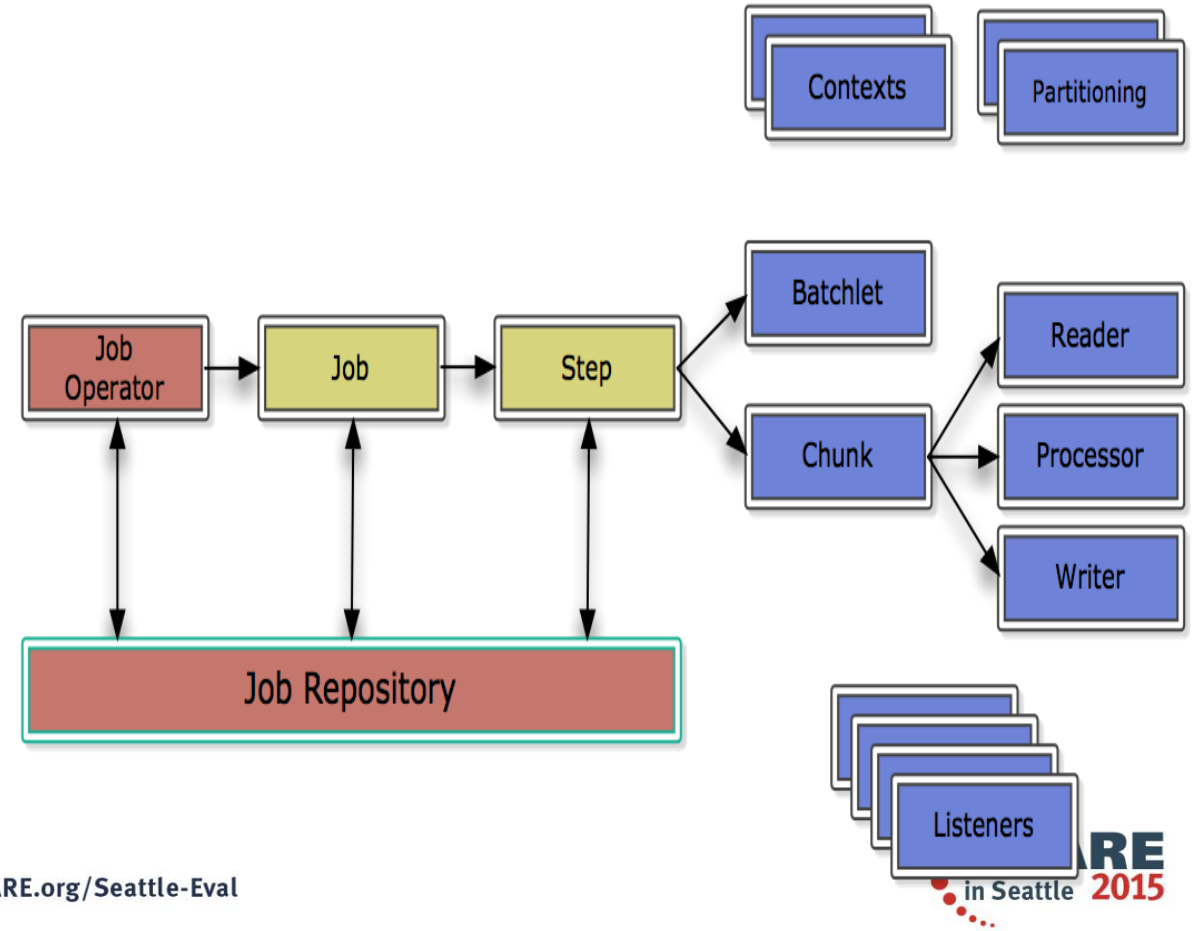
- JSR 352 defines
  - Implementation: A programming model for implementing the artifacts
  - Orchestration: A Job Specification Language, which orchestrates the execution of a batch artifacts within a job.
  - Execution: A runtime environment for executing batch application, according to a defined lifecycle.
- Note: “key” concepts, not “new” concepts!
  - Roles and abstractions should be familiar to SOA and JavaEE developers



# Anatomy of JSR352

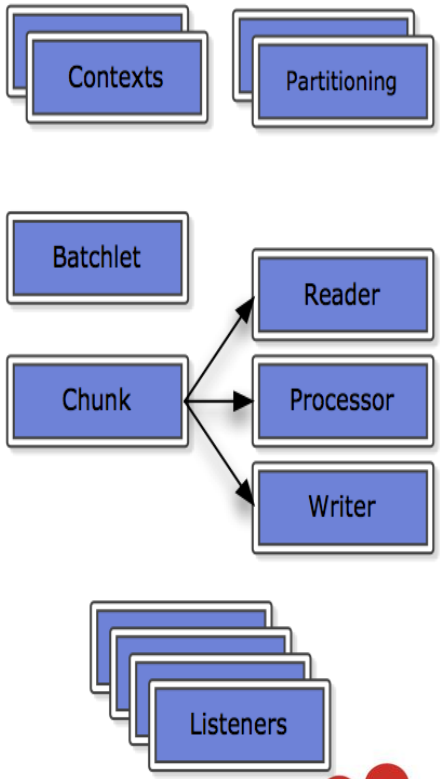
- Those concepts define the anatomy of JSR 352: Batch Applications for the Java Platform...



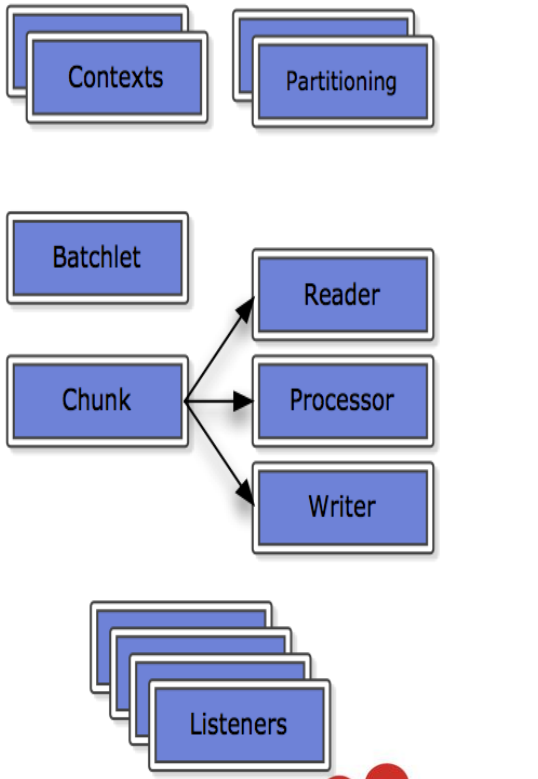
# Implementation: The programming model

- Chunk and Batchlet provide models for implementing a step.
- Contexts provide Job- and Step- level runtime information, and provide interim data persistence.
- Listeners provide callback hooks to respond to lifecycle events on batch artifacts.
- Partitioning provides a mechanism imposing parallel processing on jobs and steps



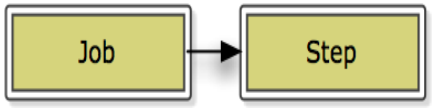
# Implementation: The programming model

- Chunk vs Batchlet
- Both are implementations of a step within a batch job
- The chunk model
  - Encapsulates a very common pattern: ETL
  - Single “reader”, “processor” and “writer”
  - Reader/Processor combination is invoked until an entire “chunk” of data is processed
  - Output “chunk” is written atomically
- Batchlet provides a “roll your own” step type
  - Invoked and runs to completion, producing a return code upon exit.



# Orchestration: The Job Specification Language (JSL)

- The JSL defines a batch job as an XML document
- Describes a step as an assemblage of batch artifacts
- Provides for the description of steps, step groupings, and execution sequencing

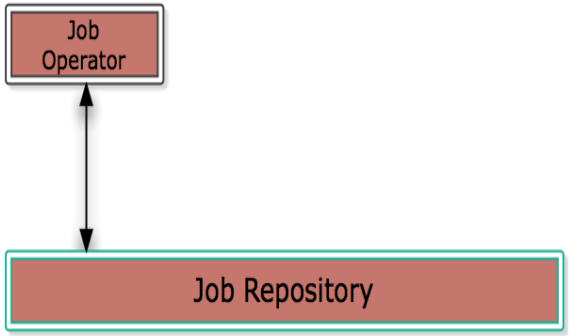



batch.xml | sample.xml

```
<job id="SampleBatchApp" restartable="true" version="1.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee">
    <step id="step1" >
        <chunk checkpoint-policy="item" item-count="10000">
            <reader ref="myChunkReader">
                <properties>
                    <property name="fileName" value="/usr/share/dict/words"/>
                </properties>
            </reader>
            <processor ref="myChunkProcessor"/>
            <writer ref="myChunkWriter">
                <properties>
                    <property name="fileName" value="/tmp/output.txt"/>
                </properties>
            </writer>
        </chunk>
    </step>
</job>
```

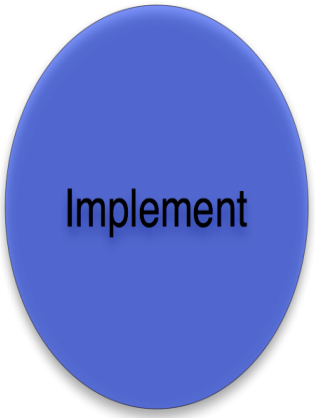
# Execution: The JobOperator and Repository

- JobOperator is the runtime interface for job management, including start, stop, restart and job repository related commands
- The Job Repository holds information about completed and executing jobs
- To start a batch job, get a JobOperator instance use it to start a job described (described by JSL).



# ItemReader / ItemProcessor

- An ItemReader encapsulates the data access and deserialization of a record.
  - No restriction on data access paradigm: use DAO patterns, JDBC, JPA, Hibernate, Spring Data, etc!
  - Checkpoint/Restart data provided as Serializable argument to “open” and from “checkpointInfo” methods.
- An ItemProcessor encapsulates the business logic applied to each record



# ItemReader code - abstracted

```
public class AnalysisFileReader implements ItemReader {

  private int numRecordsProcessed++;

  // Read and maintain current position
  public Object readItem()  {
    numRecordsProcessed++;
    return Integer.parseInt(bufferedReader.readLine());    // Or return null if no more
  }

  // Return current position when asked by container
  public Serializable checkpointInfo()  {
    return numRecordsProcessed;
  }

  // Position reader based on container-persisted value
  public void open(Serializable checkpoint) n {
    if (checkpoint != null)
      numRecordsProcessed = (Integer)checkpoint;
    else
      numRecordsProcessed = 0;

    for (int i = 0; i < numRecordsProcessed; i++) {
      bufferedReader.readLine();
    }
}
```

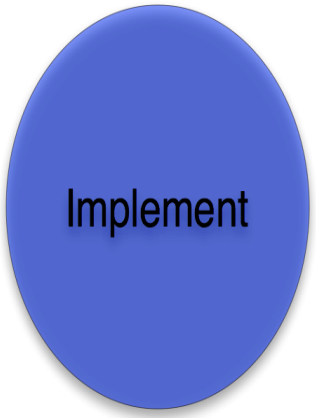
Responsible for:

- Reading
- Positioning

# ItemWriter

StringWriter.java

```
package com.timfanelli.javaone.batchdemo;

import java.io.BufferedWriter;

public class StringWriter implements ItemWriter {
    private BufferedWriter writer;
    private int lineNumber;

    @Inject
    @BatchProperty
    private String fileName;

    @Override
    public Serializable checkpointInfo() throws Exception {
        return lineNumber;
    }

    @Override
    public void close() throws Exception {
        writer.close();
    }

    @Override
    public void open(Serializable arg0) throws Exception {
        writer = new BufferedWriter(new FileWriter(new File(fileName)));
    }

    @Override
    public void writeItems(List<Object> arg0) throws Exception {
        for (Object line : arg0) {
            writer.write(line + "\n");
        }
    }
}
```




- An ItemWriter is the output counterpart to ItemReader
- Primary difference is that writeItems accepts a “chunk” of output objects (as a list) to serialize.
- Again, no restriction on data access paradigm!

# The Batch Descriptor and Job Specification

batch.xml | sample.xml

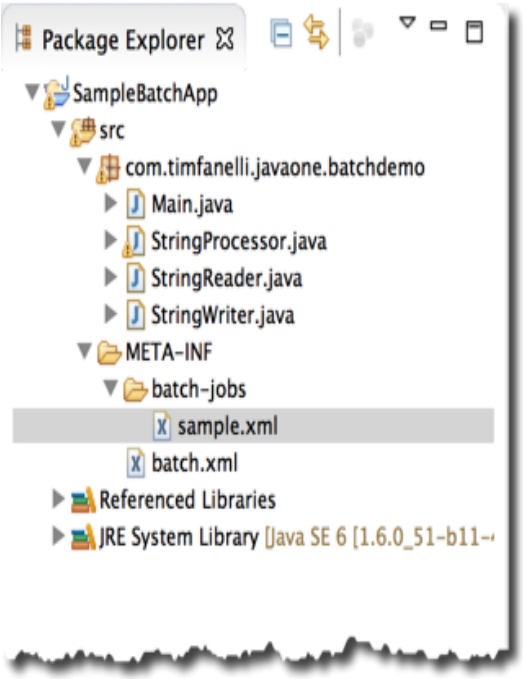
```
<batch-artifacts xmlns="http://xmlns.jcp.org/xml/ns/javaee">
    <ref id="myChunkReader" class="com.timfanelli.javaone.batchdemo.StringReader"/>
    <ref id="myChunkProcessor" class="com.timfanelli.javaone.batchdemo.StringProcessor"/>
    <ref id="myChunkWriter" class="com.timfanelli.javaone.batchdemo.StringWriter"/>
</batch-artifacts>
```

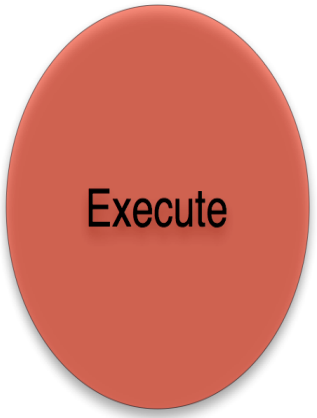

- batch.xml defines and names the default JSL ref-to-Java artifact mapping
- sample.xml is an example JSL document for SampleBatchApp

batch.xml | sample.xml

```
<job id="SampleBatchApp" restartable="true" version="1.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee">
    <step id="step1" >
        <chunk checkpoint-policy="item" item-count="10000">
            <reader ref="myChunkReader">
                <properties>
                    <property name="fileName" value="/usr/share/dict/words"/>
                </properties>
            </reader>
            <processor ref="myChunkProcessor"/>
            <writer ref="myChunkWriter">
                <properties>
                    <property name="fileName" value="/tmp/output.txt"/>
                </properties>
            </writer>
        </chunk>
    </step>
</job>
```

# The Execution




- Package the application as a standard JAR or WAR for deployment in JavaSE or EE environments
  - batch.xml goes in META-INF or WEB-INF/classes/META-INF
  - JSL may go in META-INF/batch-jobs, or submitted from an external source (up to the provider!)

# •Skip and retry

- Job Designer (Orchestrator) may decide to tolerate a certain number of failures via JSL include/exclude of specific-typed exceptions

- Similarly, certain exceptions can be declared in JSL to be handled by the container re-starting process of the current chunk (retry-with-rollback) or the current item (retry no-rollback)

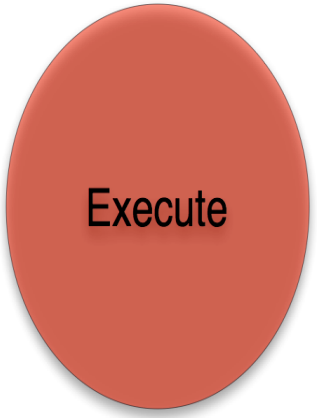
# Conditional Execution (transitioning)

```
<step id="step3">
    <batchlet ...
    <next on="RC0" to="step2"/>
    <next on="RC4" to="step2.prep"/>
    <fail on="RC-*"/> <!--  Fail on negative RC -->
</step>
```


- Exit Status provides a string based return code from step
  - Job designer can use this to control flow of execution between steps (glob patterns supported)

# Job Management - More

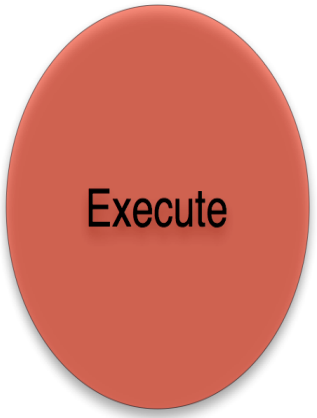
- JobOperator exposes stop() to stop currently-running job (to be restarted later).




- The door is left open for more advanced batch job management systems to be built!
  - Integration into existing enterprise schedulers
  - Plenty of options, but currently left to the provider to implement

# Java EE Integration

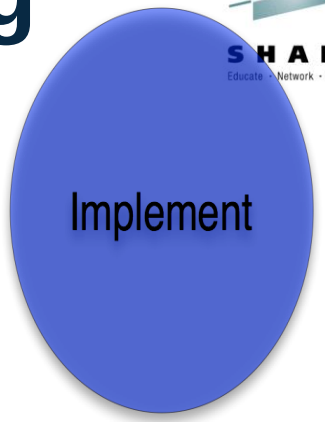
- JSR-352: Java Batch is included in Java EE 7
- Provides EE clustering, security, resource management, etc to Java Batch applications
- Performance benefits to dispatching into long-running, reusable container
  - JIT compilation through the first couple runs
  - Eliminates overhead of starting / stopping JVM



# Parallel Job Processing

- Splits and Flows provide a mechanism for executing job steps concurrently at the orchestration layer
- A flow is a sequence of one or more steps which execute sequentially, but as a single unit.
- A Split is a collection of flows that may execute concurrently
  - A split may only contain “flows”; a step is not implicitly a flow
- This is done entirely in the JSL descriptor
  - Imposed on the batch application with no code changes!

# Parallel Job Processing




- Step-level parallelism can be achieved programmatically using step partitioning
- A partitioned step runs as multiple instances with distinct property sets
- PartitionMapper defines the number of partitions, and property values for each partition
  - Can be a fixed set of partitions in JSL
  - Can be dynamic using a PartitionMapper implementation

# No new Java artifacts

- Not necessarily the general case
- Might have to
  - Coalesce Exit Status (PartitionAnalyzer)
  - Process Intermediate results on parent thread (PartitionCollector->PartitionAnalyzer)
  - Perform other tasks on end of partition (PartitionReducer)
- Also might want to programmatically partition (PartitionMapper) rather than via JSL

# •Parallel Job Processing

partitioned.xml | parallel.xml | Main.java | FilesInFolderPa | StringReader.ja

```
<job id="SampleBatchApp" restartable="true" version="1.0" xmlns="http://xmlns.jcp.org/xml/ns/javaee">
    <step id="step1" >
        <chunk checkpoint-policy="item" item-count="10000">
            <reader ref="myChunkReader">
                <properties>
                    <property name="fileName" value="#{partitionPlan['input.fileName']}"/>
                </properties>
            </reader>
            <processor ref="myChunkProcessor"/>
            <writer ref="myChunkWriter">
                <properties>
                    <property name="fileName" value="#{partitionPlan['output.fileName']}"/>
                </properties>
            </writer>
        </chunk>
        <partition>
            <mapper ref="myPartitionMapper">
                <properties>
                    <property name="filePath" value="/Users/timfanelli/BatchDemoFiles/in"/>
                </properties>
            </mapper>
        </partition>
    </step>
</job>
```

partition plan property substitutions

partition mapper reference and properties

Design | Source

partitioned.xml | parallel.xml | Main.java | FilesInFolderPa | StringReader.ja

```
package com.timfanelli.javaone.batchdemo;

import java.io.File;

public class FilesInFolderPartiionMapper implements PartitionMapper {
    @Inject
    @BatchProperty
    private String filePath;

    @Inject
    @BatchProperty
    private String outputPath;

    @Override
    public PartitionPlan mapPartitions() throws Exception {
        File path = new File(filePath);

        List<String> entryPaths = new ArrayList<String>();
        for ( String entry : path.list() ) {
            String entryPath = filePath + File.separator + entry;
            if ( new File(entryPath).isFile() )
                entryPaths.add(entryPath);
        }

        int i = 0;
        PartitionPlan plan = new PartitionPlanImpl();
        Properties[] properties = new Properties[entryPaths.size()];

        for ( String entry : entryPaths ) {
            properties[i] = new Properties();
            properties[i].setProperty("input.fileName", entry);
            properties[i].setProperty("output.fileName", outputPath + i + ".txt");

            ++i;
        }

        plan.setPartitions( properties.length );
        plan.setPartitionProperties(properties);

        return plan;
    }

}
```

- Some subtleties

# • Context gotchas

- JobContext/StepContext is sort of “thread-local” (yet partitions run on their own thread)
  - No built-in way to simply write to JobContext in step 1 and then access the data from partitions in partitioned step 2
  - If this seems crazy, consider partitions running in separate JVMs
- Careful with transient data in JobContext. If we restart job in the middle of step 2 then transient data from step 1 won’t be there

# Why Java Batch and Liberty?

- We have wrapped all the qualities of service of the Liberty profile around the batch programming model
  - Dynamic configuration
  - Operational management
  - Transactions
  - Logging
  - High availability
  - Scalability
  - Tooling

# More about Operational Management

- REST API
  - Java Batch in the Liberty profile provides an easy to use rest interface to remotely manage your batch jobs
  - Ability to start, stop, restart, view job instance and execution data, and access job logs
- Job Logging
  - Server logs are interleaved with application records for easy debugging
- External scheduler integration
  - Provides the ability to combine enterprise quality scheduling with batch

# More about tooling

- WebSphere Developer Tools
  - Create Java Batch Applications using the JSR-352 programming model and XML job definitions
  - Job Creation Wizards
  - JSL Editor
  - Java class wizards
  - Simple to submit, test, and debug applications
  - Remote deployment and debug capabilities
  - Simple graphical UI to monitor jobs