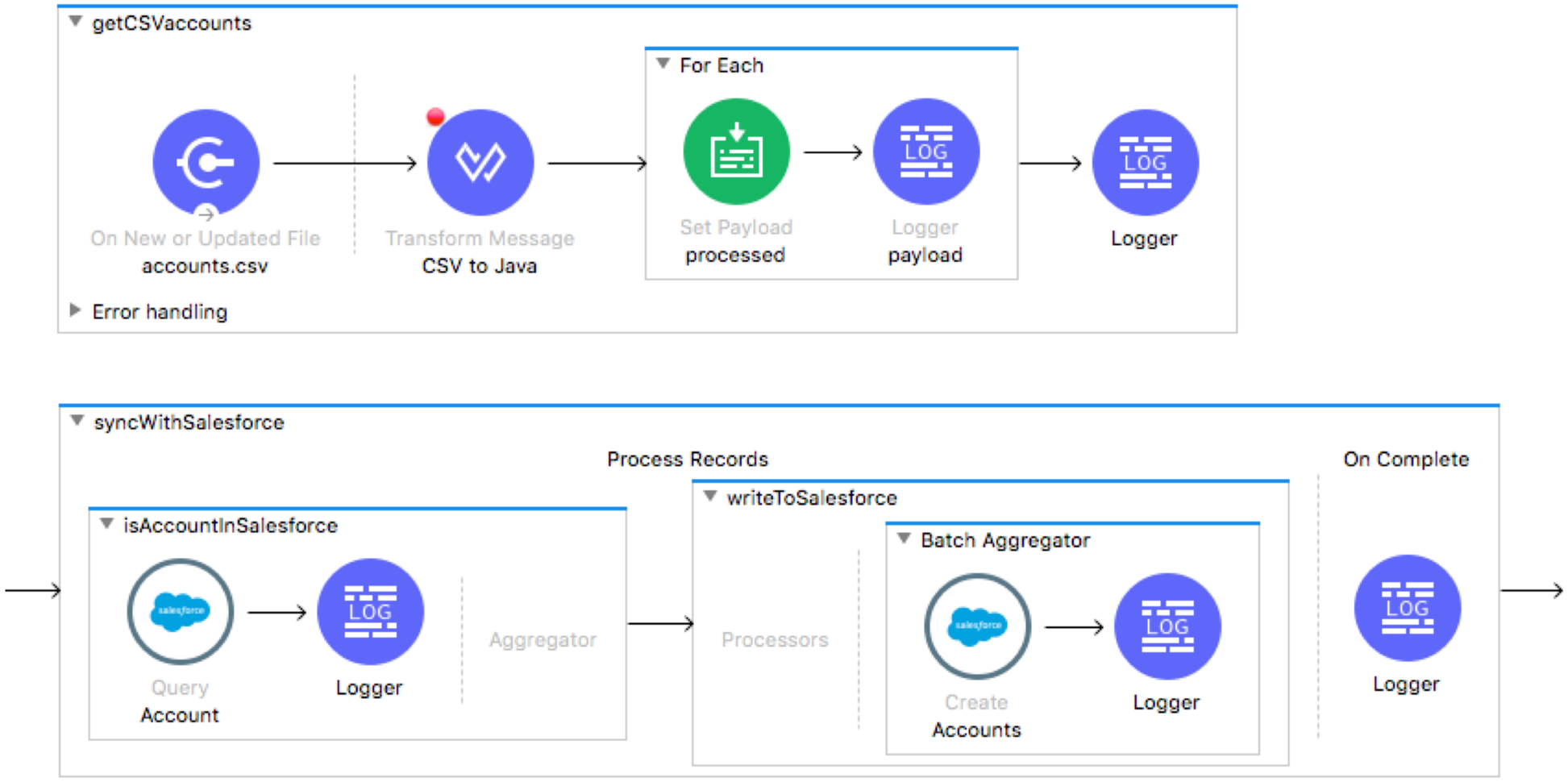
Module 13: Processing Records

Goal



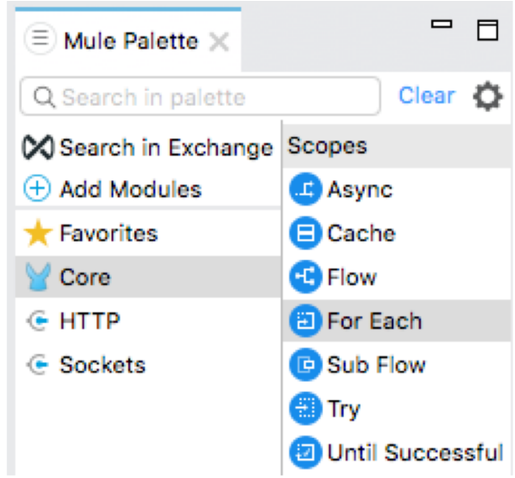
At the end of this module, you should be able to

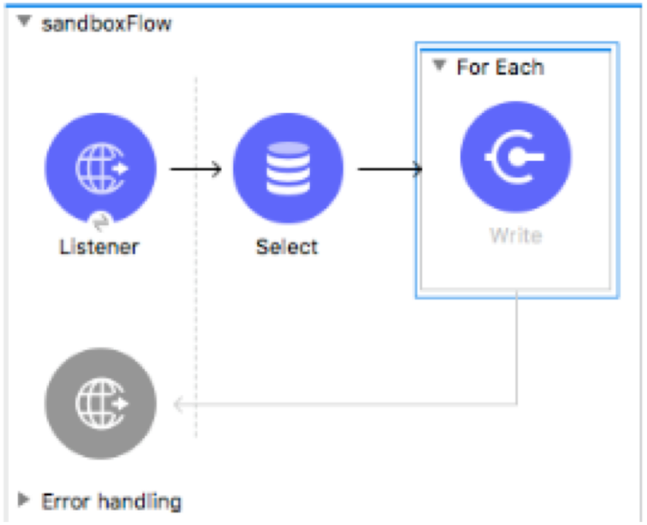
- Process items in a collection using the For Each scope
- Process records using the Batch Job scope
- Use filtering and aggregation in a batch step

Processing items in a collection with the For Each scope

The For Each scope

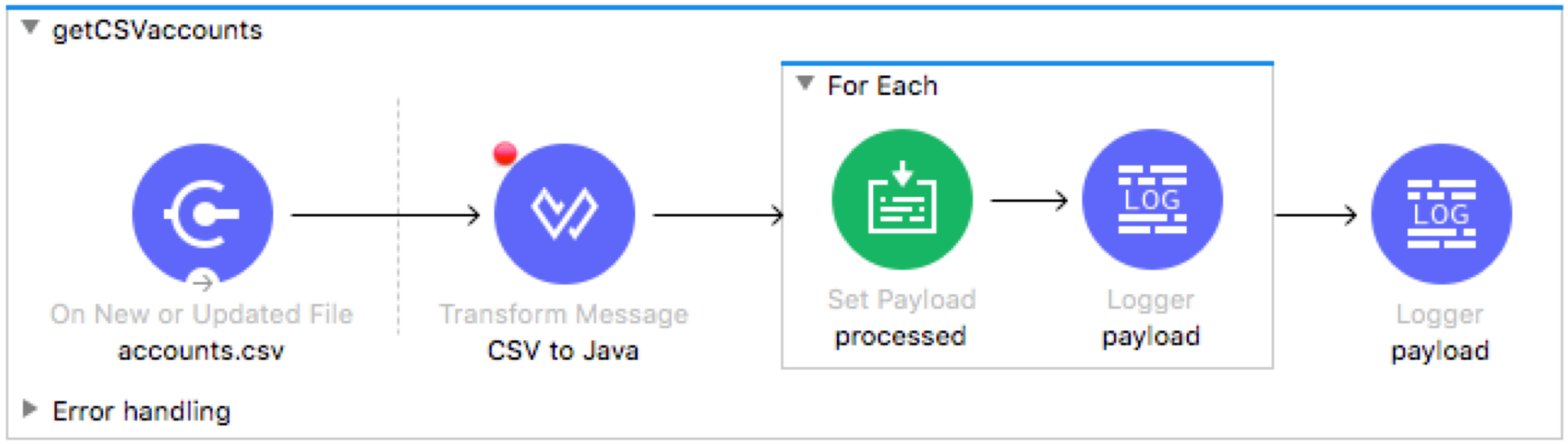
- Splits a payload collection and processes the individual elements
  - Collection can be any supported content type, including application/json, application/java, or application/xml
- Returns the original payload
  - Regardless of any modifications made inside the scope
- Stops processing and invokes an error handler if one element throws an exception





Walkthrough 13-1: Process items in a collection individually

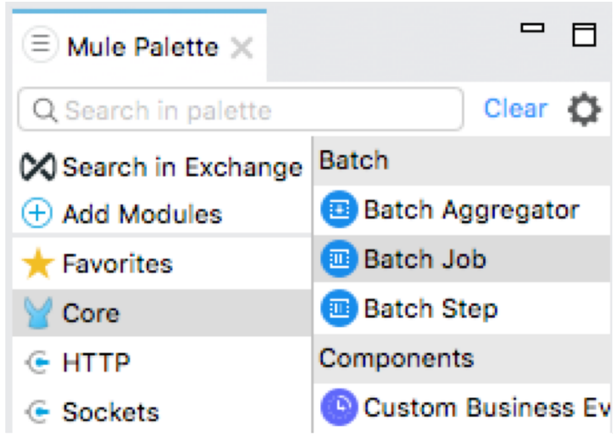
- Use the For Each scope element to process each item in a collection individually
- Change the value of an item inside the scope
- Examine the payload before, during, and after the scope
- Look at the thread used to process each item



Processing records with the Batch Job scope

The Batch Job scope

- Provides ability to split large messages into records that are processed asynchronously in a batch job



- Created especially for processing data sets
  - Splits large or streamed messages into individual records
  - Performs actions upon each record
  - Handles record level failures that occur so batch job is not aborted
  - Reports on the results
  - Potentially pushes the processed output to other systems or queues

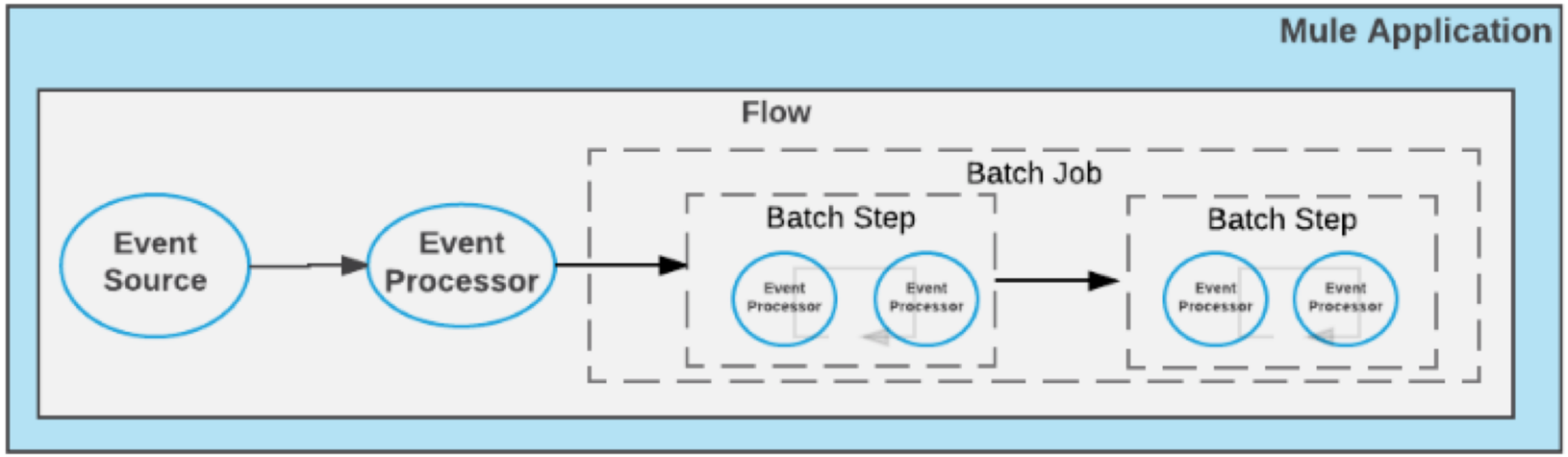
- Enterprise edition only

Example use cases

- Integrating data sets to parallel process records
  - Small or large data sets, streaming or not
- Engineering "near real-time” data integration
  - Synchronizing data sets between business applications
  - Like syncing contacts between NetSuite and Salesforce
- Extracting, transforming, and loading (ETL) info into a target system
  - Like uploading data from a flat file (CSV) to Hadoop
- Handling large quantities of incoming data from an API into a legacy system

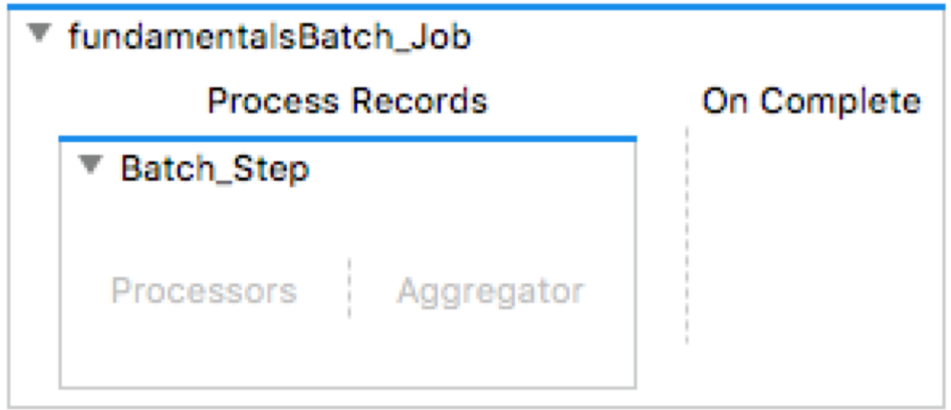
How a batch job works

- A batch job contains one or more batch steps that act upon records as they move through the batch job



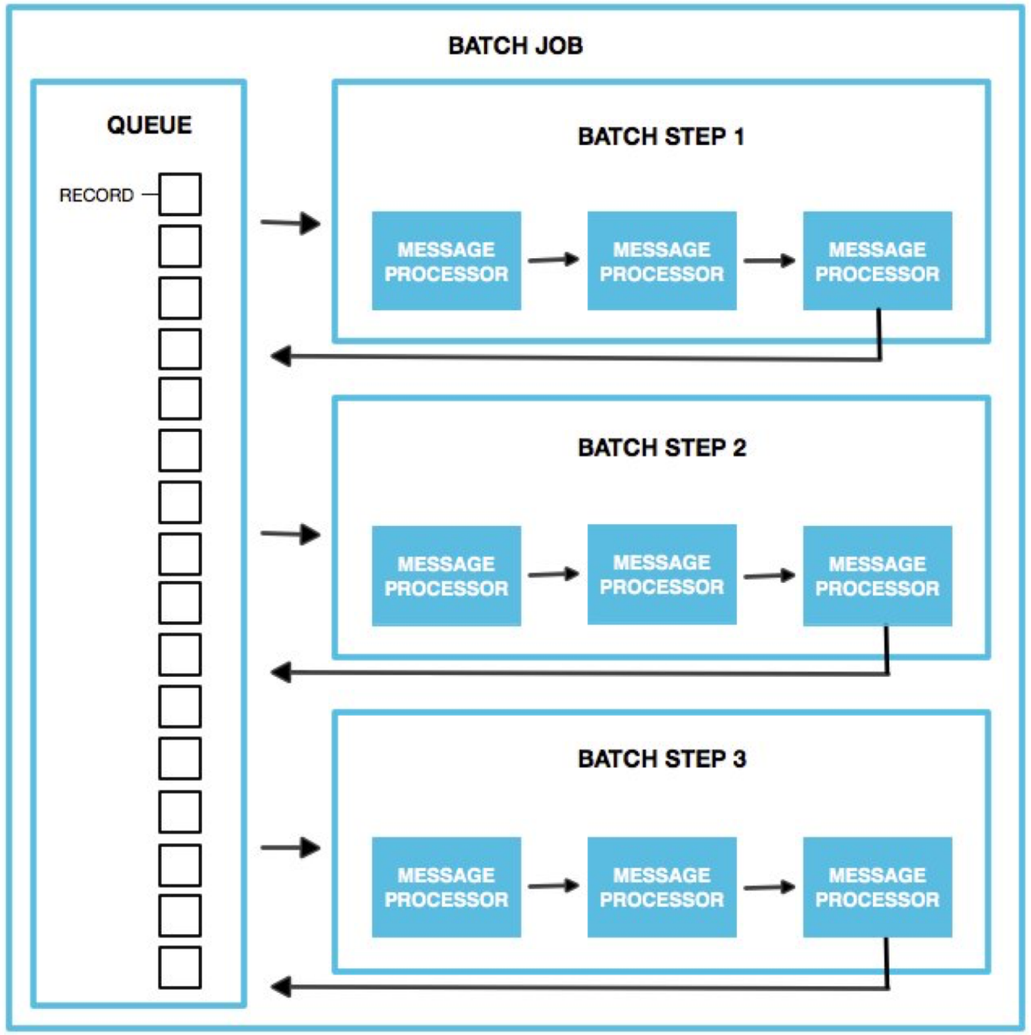
A batch job contains three different phases

- **Load and dispatch** (implicit)
  - Performs “behind-the-scene” work
    - Splits payload into a collection of records
    - Creates a persistent queue and stores each record in it
- **Process** (required)
  - Asynchronously processes the records
  - Contains one or more batch steps
- **On complete** (optional)
  - Reports summary of records processed
  - Provides insight into which records failed so you can address issues



How record processing works

- One queue exists
- Each record
  - Keeps track of what steps it has been processed through
  - Moves through the processors in the first batch step
  - Is sent back to the queue
  - Waits to be processed by the second step
- This repeats until each record has passed through every batch step
- **Note**: All records do not have to finish processing in one step before any of them are sent to the next step



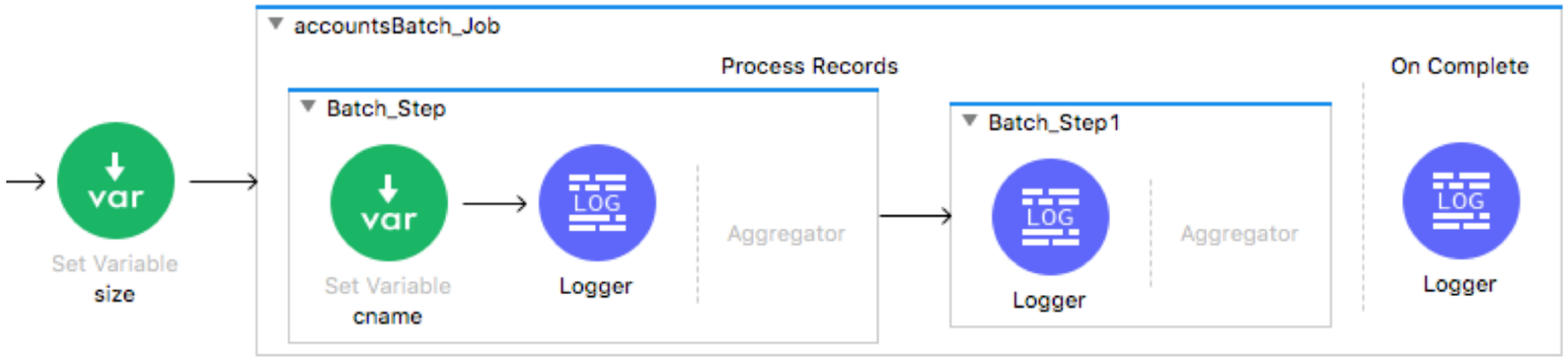
Batch job performance

- Batch records are queued and scheduled in blocks of 100
  - This lessens the amount of I/O requests and improves an operation’s load
- A threading profile of 16 threads per job is used
  - Each of the 16 threads processes a block of 100 records
  - Each thread iterates through that block processing each record, and then each block is queued back and the process continues
- This configuration works for most use cases, but can be customized to improve batch’s performance in certain use cases

Variables in a batch job

- Variables created before a batch job are available in all batch steps

- Variables created inside a batch step are record-specific
  - Persist across all batch steps in the processing phase
  - Commonly used to capture whether or not a record already exists in a database

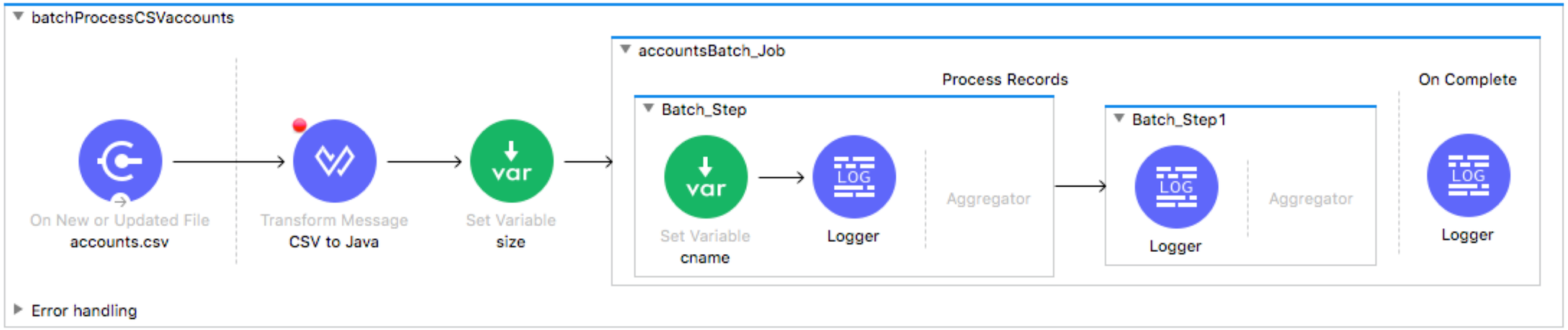


Handling record-level errors during processing

- If a record fails to be processed by a processor in a batch step, there are three options
  - **Stop processing** the entire batch (default)
    - In-flight steps are finished, but all other steps are skipped and the on complete phase is invoked
  - **Continue processing** the batch
    - You need to specify how subsequent batch steps should handle failed records
      - To do this, use batch step filters that are covered in the next section
  - **Continue processing** the batch **until a max number** of failed records is reached
    - At that point, the on complete phase is invoked

Walkthrough 13-2: Create a batch job for records in a file

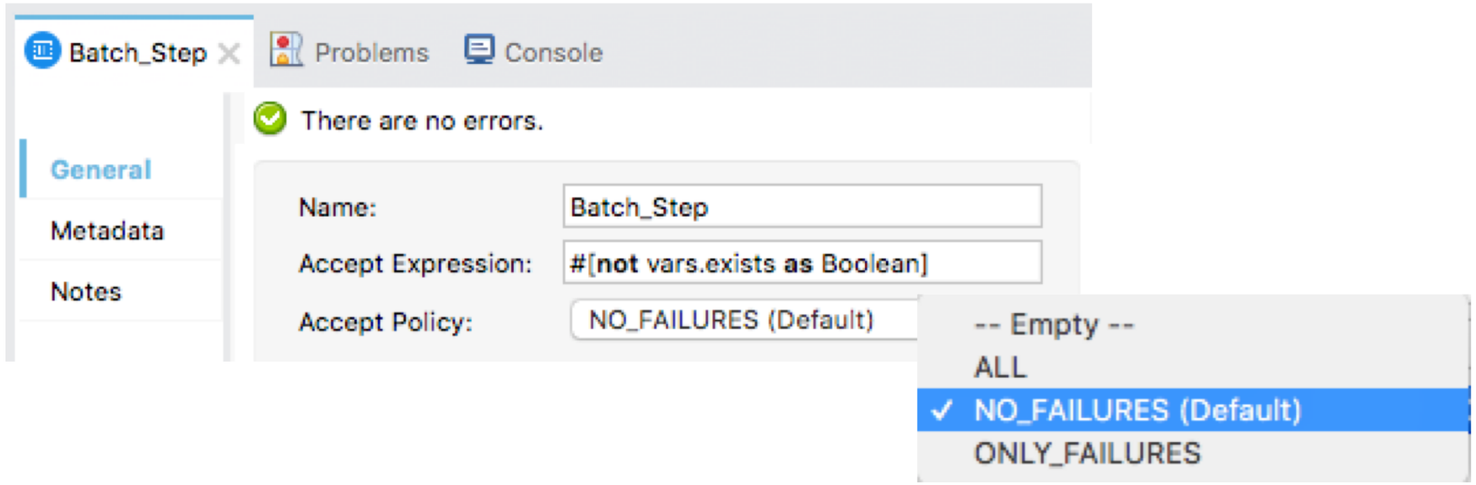
- Use the Batch Job scope to process items in a collection
- Examine the payload as it moves through the batch job
- Explore variable persistence across batch steps and phases
- Examine the payload that contains information about the job in the on complete phase
- Look at the threads used to process the records in each step



Using filtering and aggregation in a batch step

Using filters to specify when a batch step is executed

- A batch job has two attributes to filter the records it processes
  - An accept expression
  - An accept policy

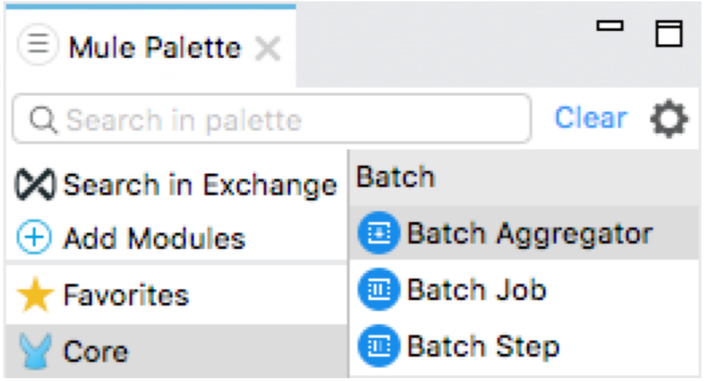


- Examples
  - Prevent a step from processing any records which failed processing in the preceding step
  - In one step, check and see if the record exists in some data store and then in the next only upload it to that data store if it does not exist

Aggregating records in a batch step for for bulk insert

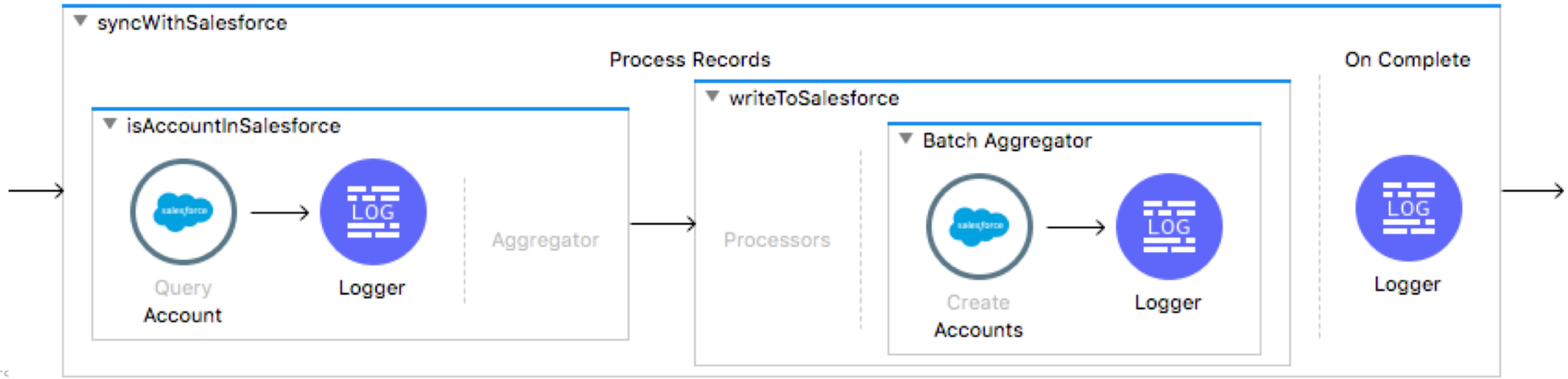
- To accumulate records, use a **Batch Aggregator** scope inside the Aggregator section of a batch step

- For example, instead of using a separate API call to upsert each record to a service, upload them in a batch of 100



Walkthrough 13-3: Use filtering and aggregation in a batch step

- Use a batch job to synchronize database records to Salesforce
- In a first batch step, check to see if the record exists in Salesforce
- In a second batch step, add the record to Salesforce
- Use a batch step filter so the second step is only executed for specific records
- Use a Batch Aggregator scope to commit records in batches



Summary

Summary

- Use the **For Each** scope to process individual collection elements sequentially and return the original payload
- Use the **Batch Job** scope (EE only) for complex batch jobs
  - Created especially for processing data sets
  - Splits messages into individual records and performs actions upon each record
  - Can have multiple batch steps and these can have filters
  - Record-level data is persisted across steps using variables
  - Can handle record level failures so the job is not aborted
  - The batch job returns an object with the results of the job for insight into which records were processed or failed