# DocumentProcessing

(This page is a child of the TaskList page)

## Problem

Solr would benefit from a flexible document processing framework meeting the requirements of enterprise grade content integration. Most search projects have some need for processing the incoming content prior to indexing, for example:

- Language identification
- Text extraction (Tika)
- Entity extraction and classification
- Data normalization and cleansing
- Routing
- 3rd party systems integration (e.g. enrich document from external source)
- etc

The built-in UpdateRequestProcessorChain is capable of doing simple simple processing jobs, but it is only built for local execution on the indexer node in the same thread. This means that any performance heavy processing chains will slow down the indexers without any way to scale out processing independently. We have seen FAST systems with far more servers doing document processing than indexing.

There are many processing pipeline frameworks from which to get inspiration, such as the one in FAST ESP, OpenPipeline, OpenPipe (now on GitHub), Pypes, UIMA, Eclipse SMILA, Apache commons pipeline, Piped, Behemoth, Findwise's yet-to-be-announced pipeline and others. Indeed, some of these are already being used with Solr as a pre-processing server.

A choice of technologies is good, but it can be a bit too much and fragmented as well...

There have recently been interest within the search community for a true open source pipeline with a healthy community behind it and a rich pool of processors. See this presentation from Lucene Eurocon 2010 as well as this blog post for thoughts from FindWise, as well as the recent solr-user thread Pipeline for Solr and Cominvent's talk at Lucene Eurocon 2011 Improving Solr's Update Chain. In addition to developing a true open source preferred solution, it should also be possible to improve interoperability and compatibility.

Here are a few things that we could consider in order to ease this situation:

- Start talking together and try find common ground, places to cooperate, consolidate etc
- Develop a common Java interface which models a document processor, enabling cross-pipeline use of the same processor
- Develop a Java wrapper for executing Python processors (reuse of ESP processors, Pypes processors and Piped processors) in a Java pipeline
- Specify a common "Document" model which may be serialized between various components (Avro based?)
- Establish a source repository (outside of the ASF) of reusable processors, maintained by a large community

*Update*: At Lucene Eurocon 2011 in Barcelona, we formed an interest group for pipelines which has its home at http://www.meetup.com/SearchPipelines/

## Wishes for a Lucene targeted pipeline

Here are some thoughts and wishes for a new pipeline project mainly target at Lucene based search enginens (including Solr, ElasticSearch and Lucene itself). It should probably build upon/fork one of the existing projects and best practices.

### Key requirements

#### Must

- Apache licensed
- Java based
- Lightweight (not over-engineered)
- Support for multiple named pipelines, addressable at document ingestion
- Support for a rich document format, including token streams (pre-analyzed content)
- Support for metadata on document and field level (e.g. tokenized=true, language=en)
- Well defined dead-simple API and SDK for the processing stages
- Easy configuration of pipelines through separate config and GUI
- Run standalone as well as embedded in another framework (such as Solr's UpdateChain)
- Do not directly depend on Solr, but allow easy, tight integration with either Lucene or Solr

#### Should

- SDK for stage developers - to encourage stage development
- Easily debuggable and testable
- Separate stages repository (e.g. a gitHub space, outside of ASF svn) to encourage sharing
- Integration points for UIMA, LingPipe, OpenNLP etc
- Be able to run Lucene's Tokenizers and Token Filters directly and ship this to Lucene as the new "pre-analyzed" field (see SOLR-1535)
- Support for writing stages in JVM scripting languages such as Jython

### Could

- GUI for configuring pipelines
- Hot pluggable pipelines
- Wrappers for custom FAST ESP stages to work with minor modification
- Wrappers for custom UpdateProcessor stages to work with minor modification
- Robust - if a batch fails, it should re-schedule to another processor
- Optimize for performance through e.g. batch support
- Allow scaling out processing to multiple dedicated servers for heavy tasks. Cloud-friendly
- Support status callbacks to the client

## Anti-patterns

- Do not over-architecture like Eclipse SMILA and others have done going crazy with ESB etc
- Do not try to be a connector framework as well. Let ManifoldCF do that job. Focuson on the pipeline!
- Do not keep the source private (although Apache licensed) as DieselPoint did with OpenPipeline - create a community!

## Proposed architecture

Jan Høydahl: I think OpenPipe is a hot candidate to fork as a new open source framework. It already supports most of the above, is Apache licensed, and is abandoned by its original developers.

## Risks

TBD

## Q&A

### Your question here

- Q: Is there a JIRA issue that tracks the Solr-side development of this?
- A: Not yet
- Q: How is this related to https://issues.apache.org/jira/browse/SOLR-2129?
- A: SOLR-2129 is an UpdateProcessor for UIMA (see SolrUIMA). Here we're talking about a new standalone framework and a way to integrate this and other existing pipelines cleanly with Solr/Lucene.
- Q: Will the pipelines have to be linear. For instance, could we implement a first stage in the pipeline that would be a splitter. The splitter could, for example, break up a large XML document into chapters, then push each chapter to the next stage where other processing will take place. In the end, the Lucene index would have one document per chapter.
- A: The new framework can be however we want it. If you talk about the Solr UpdateChain, we suggest in SOLR-2841 a way to support non linear chains. For splitting in chapters however, I think that a UpdateRequestHandler may be a better choice, see http://wiki.apache.org/solr/XsltUpdateRequestHandler
- Q: How will the pipelines support compound files, e.g. archives, e-mail messages with attachments (which could be archives), etc.? This could be a problem if pipelines are linear.
- A: This is an open question. For the new pipeline framework, there are many possibilities, which must be discussed. If you're thinking about the Solr UpdateChain, you have a choice whether your UpdateRequestHandler should understand the input format and do the splitting for you. But it should also be possible to write an UpdateProcessor which splits the incoming SolrInputDocument into multiple sub documents - generating unique IDs for each. You would somehow need to inject these sub documents again, either by using SolrJ from your UpdateProcessor or by instantiating a "sub chain" in another thread to push the sub docs into the index. This is however, left as an exercise for the user 🙂