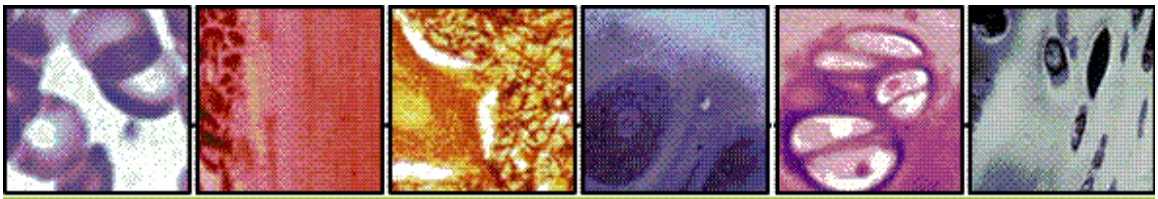
Developer Guide v2.8

SPINScrubberDeveloperGuide

Title:

Author: Andrew McMurry

Contact: Andrew_McMurry@hms.harvard.edu



Intended Audience :

Developers looking to extend, customize, or contribute to this scrubber utility.
It is assumed the reader has already reviewed the *Scrubber User Guide.*

Open Source:
We are fully dedicated to open source and use Apache Development Tools:

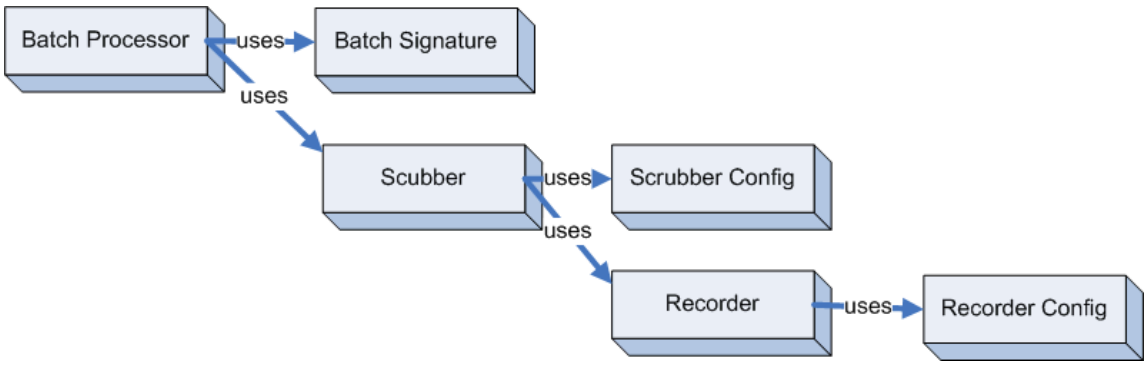
Subversion (source code)
http://scm.chip.org/svn/repos/spin/scrubber/

Maven Repository (download binaries)
http://repo.open.med.harvard.edu/nexus/content/repositories/releases/org/spin/scrubber/scrubber-core/2.8/

JIRA (Bug / Feature Tracking, open to the public)
http://open.med.harvard.edu/jira/browse/SCRUBBER

BAMBOO (Continuous Integration)
http://open.med.harvard.edu/bamboo

Domain Object Relationships



BatchProcessor:
Process a batch (collection) of input files at once.
The batch process accepts any type of scrubber provided (text, xml, etc....)

BatchSignature:
Signs each batch output file with a signature describing the resources (files) were used to process the input.

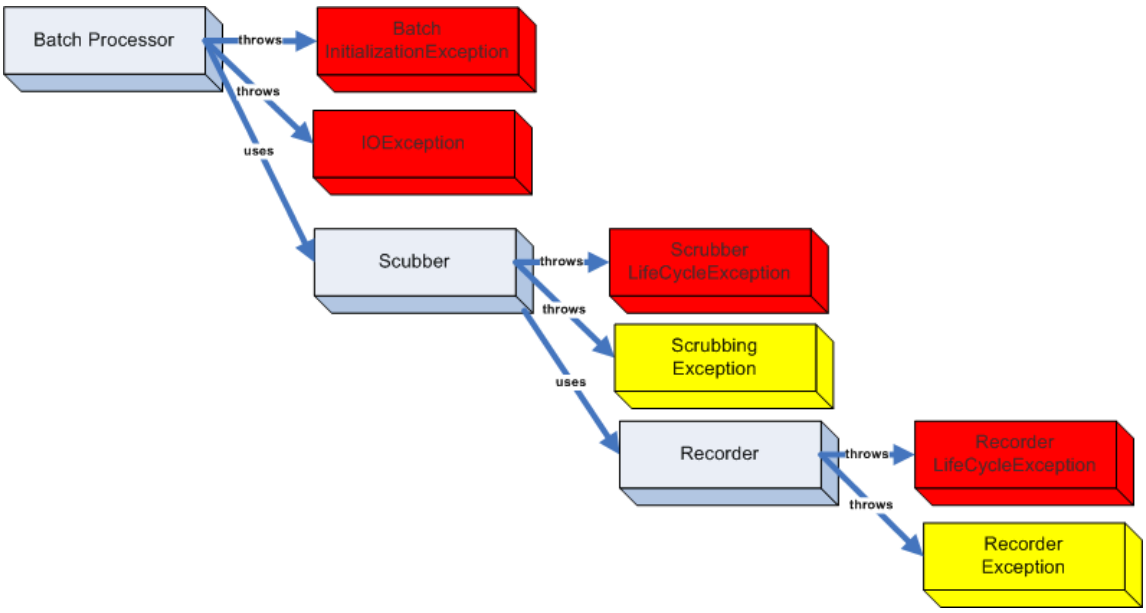
Scrubber:
Removes identifiers and regular expressions from text.

Scrubber Config:
Defines the identifiers and regular expressions to be removed.

Recorder:
Record the matches found by evaluating the identifiers and regular expressions.

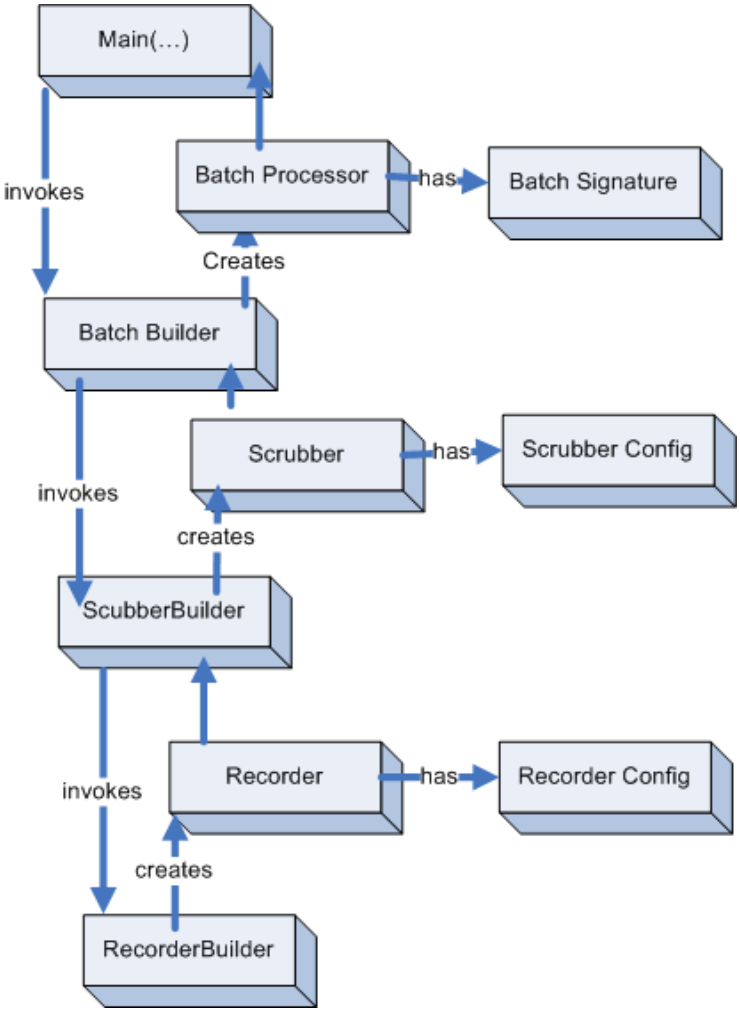
Recorder Config:
Used to initialize a recorder implementation

Exception Handling Model



The **BatchProcessor** continues if a single file fails, but will abort if there is an error during initialization of the batch, scrubber, and recorder. The diagram above depicts the catched exceptions as *warnings* in YELLOW and the *f atal* exceptions in RED .

Creating Scrubber Objects *via* Chained Builders



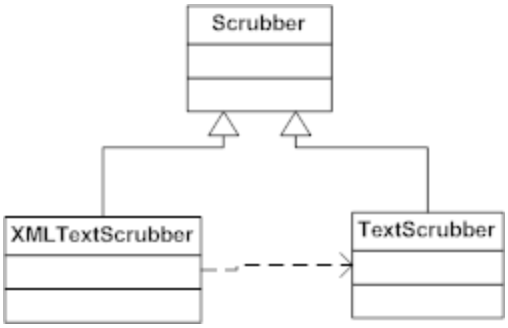
The configuration XML can be used to assemble a runtime collection of scrubber objects.
In this diagram, each builder uses and consumes the product of the builder immediately below it.

Customizing the Scrubber

The scrubber configuration allows for tremendous flexibility.

If further customization is needed, we recommend *extending the existing scrubbers* rather than writing a whole new implementation.

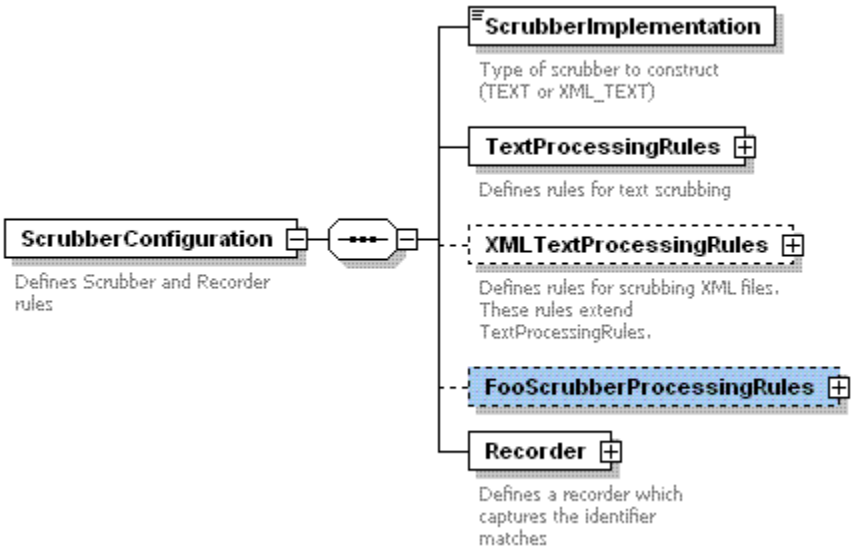
This is how XMLTextScrubber was developed.
It implements the scrubber interface and uses the *visitor pattern* over the XML tree to *apply the TextScrubber*.



To integrate a new scrubber into the application, you must update ScrubberBuilder.makeScrubber(…) to include a call to makeMyFooScrubber(…)

At runtime, set the configuration property “ScrubberConfiguration\ScrubberImplementation” to “MyFooScrubber”.

Any resources required by the new Scrubber should be defined in the configuration file like this:



Custom Recorders

It should be trivial to create a new recorder by simply extending the abstract class **“Recorder”** and defining an implementation of init(), shutdown(), and handle() methods.

Much like defining a new scrubber, you must update the RecorderBuilder.makeRecorder(…) to include a call to makeMyFooRecorder(…)

At runtime, set the configuration property “ScrubberConfiguration\Recorder\Implementation” to “MyFooRecorer”.

Any resources required by the Recorder should be defined in the configuration file. [ScrubberConfiguration\Recorder\ConfigParam(s)]

Customizing the Batch Process

The default program entry point is through the *DefaultBatchRunner* class which invokes the BatchBuilder. To make a new kind of batch process, update the BatchBuilder.makeBatch(…) to include a call to makeMyFooBatch(….)

Testing Regular Expressions

The regular expression configuration file regex.txt should be fined tuned to your particular medical record structure. An extensive library of regular expressions exist at http://regexlib.com/ and a Java-based online regular expression testing tool is located at: http://myregexp.com/