Medici Extractor in Java

This page will describe how to create an extractor using java. A lot of this code can be copied and pasted. The only major part is the processFile function. You can find this example also in our code repository at: https://opensource.ncsa.illinois.edu/stash/projects/MMDB/repos/extractors-examples/browse/java

To get started we define all required jar files needed in a maven pom file

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>


	<groupId>edu.illinois.ncsa.medici</groupId>
	<artifactId>example-extractor</artifactId>
	<version>1.0.0-SNAPSHOT</version>

	<properties>
		<exec.mainClass>edu.illinois.ncsa.medici.extractor.example.WordCount</exec.mainClass>
	</properties>


	<dependencies>
		<dependency>
			<groupId>com.rabbitmq</groupId>
			<artifactId>amqp-client</artifactId>
			<version>3.0.2</version>
		</dependency>
		<dependency>
			<groupId>commons-io</groupId>
			<artifactId>commons-io</artifactId>
			<version>2.4</version>
		</dependency>
		<dependency>
			<groupId>com.fasterxml.jackson.core</groupId>
			<artifactId>jackson-databind</artifactId>
			<version>2.4.1.1</version>
		</dependency>
		<dependency>
			<groupId>commons-logging</groupId>
			<artifactId>commons-logging</artifactId>
			<version>1.1.3</version>
		</dependency>
		<dependency>
			<groupId>log4j</groupId>
			<artifactId>log4j</artifactId>
			<version>1.2.17</version>
			<scope>runtime</scope>
		</dependency>
	</dependencies>
</project>
```

The first step in the example is to setup some global variables that will be used in the rest of the code.

configuration

```
    // name where rabbitmq is running
    private static String rabbitmqhost = "localhost";

    // name to show in rabbitmq queue list
    private static String exchange = "medici";

    // name to show in rabbitmq queue list
    private static String extractorName = "wordCount";

    // username and password to connect to rabbitmq
    private static String username = null;
    private static String password = null;

    // accept any type of file that is text
    private static String messageType = "*.file.text.#";

    // secret key used to connect to medici, this will eventually be
    // part of the message received.
    private static String secretKey = "r1ek3rs";
```

A convenience function to return status messages to Medici

statusUpdate

```
    private void statusUpdate(Channel channel, AMQP.BasicProperties header, String fileid, String status)
throws IOException {
        logger.debug("[" + fileid + "] : " + status);
        Map<String, Object> statusreport = new HashMap<String, Object>();
        statusreport.put("file_id", fileid);
        statusreport.put("extractor_id", extractorName);
        statusreport.put("status", status);
        statusreport.put("start", dateformat.format(new Date()));
        AMQP.BasicProperties props = new AMQP.BasicProperties.Builder().correlationId(header.
getCorrelationId()).build();
        channel.basicPublish(exchange, header.getReplyTo(), props, mapper.writeValueAsBytes(statusreport));
    }
```

The main method will listen for messages on the message bus.

main

```
     public static void main(String[] argv) throws Exception {
        // setup connection parameters
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(rabbitmqhost);
        if ((username != null) && (password != null)) {
            factory.setUsername(username);
            factory.setPassword(password);
        }
        // connect to rabitmq
        Connection connection = factory.newConnection();
        // connect to channel
        final Channel channel = connection.createChannel();
        // declare the exchange
        channel.exchangeDeclare(exchange, "topic", true);
        // declare the queue
        channel.queueDeclare(extractorName, true, false, false, null);
        // connect queue and exchange
        channel.queueBind(extractorName, exchange, messageType);
        // create listener
        channel.basicConsume(extractorName, false, "", new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties header, byte
[] body) throws IOException {
                WordCount wc = new WordCount();
                wc.onMessage(channel, envelope.getDeliveryTag(), header, new String(body));
            }
        });
        // start listening
        logger.info("[*] Waiting for messages. To exit press CTRL+C");
        while (true) {
            Thread.sleep(1000);
        }
    }
```

Next the function to handle a message. This will call a function to download the data, and process the data. Only if download and process_file are handled without any errors, this function will ack the message on the message bus, telling rabbitMQ the file has been processed, and nobody else should process the file again. If this ack did not happen (because the process died in the middle), the next extractor listening on this channel will pick up the message and process the file.

onMessage

```
    public void onMessage(Channel channel, long tag, AMQP.BasicProperties header, String body) {
        File inputfile = null;
        String fileid = "";

        try {
            @SuppressWarnings("unchecked")
            Map<String, Object> jbody = mapper.readValue(body, Map.class);
            String host = jbody.get("host").toString();
            fileid = jbody.get("id").toString();
            String intermediatefileid = jbody.get("intermediateId").toString();
            if (!host.endsWith("/")) {
                host += "/";
            }
            statusUpdate(channel, header, fileid, "Started processing file");

            // download the file
            inputfile = downloadFile(channel, header, host, secretKey, fileid, intermediatefileid);

            // process file
            processFile(channel, header, host, secretKey, fileid, intermediatefileid, inputfile);

            // send ack that we are done
            channel.basicAck(tag, false);
        } catch (Throwable thr) {
            logger.error("Error processing file", thr);
            try {
                statusUpdate(channel, header, fileid, "Error processing file : " + thr.getMessage());
            } catch (IOException e) {
                logger.warn("Could not send status update.", e);
            }
        } finally {
            try {
                statusUpdate(channel, header, fileid, "Done");
            } catch (IOException e) {
                logger.warn("Could not send status update.", e);
            }
            if (inputfile != null) {
                inputfile.delete();
            }
        }
    }
```

This function will download the actual file from medici so it can be processed by the local extractor. The on_message function will take care of removing the file after the processing of data is complete.

downloadFile

```
    private File downloadFile(Channel channel, AMQP.BasicProperties header, String host, String key, String
fileid, String intermediatefileid) throws IOException {
        statusUpdate(channel, header, fileid, "Downloading file");
        URL source = new URL(host + "api/files/" + intermediatefileid + "?key=" + key);
        File outputfile = File.createTempFile("medici", ".tmp");
        outputfile.deleteOnExit();
        FileUtils.copyURLToFile(source, outputfile);
        return outputfile;
    }
```

Finally this is the main block of code. This will process the data, and send any extracted metadata, or previews back to medici. The following example will send back extracted metadata back to medici.

processFile

```
    private void processFile(Channel channel, AMQP.BasicProperties header, String host, String key, String
fileid, String intermediatefileid, File inputfile) throws IOException {
        statusUpdate(channel, header, fileid, "Counting words in file.");
        // implementation of word count (unix wc utility)
        int lines = 0, words = 0, characters = 0;
        BufferedReader br = new BufferedReader(new FileReader(inputfile));
        String line;
        while ((line = br.readLine()) != null) {
            lines++;
            String[] pieces = line.split("\\s+");
            words += pieces.length;
            for (String s : pieces) {
                characters += s.length();
            }
        }
        br.close();

        // store results as metadata
        Map<String, Object> metadata = new HashMap<String, Object>();
        metadata.put("lines", lines);
        metadata.put("words", words);
        metadata.put("characters", characters);
        postMetaData(host, key, fileid, metadata);
    }
```

Finally there is the postMetaData function which will post metadata back to medici at a specific URL

postMetaData

```
    private String postMetaData(String host, String key, String fileid, Map<String, Object> metadata) throws
IOException {
        URL url = new URL(host + "api/files/" + fileid + "/metadata?key=" + key);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-Type", "application/json");
        conn.setDoOutput(true);

        DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
        mapper.writeValue(wr, metadata);
        wr.flush();
        wr.close();

        int responseCode = conn.getResponseCode();
        if (responseCode != 200) {
            throw (new IOException("Error uploading metadata [code=" + responseCode + "]"));
        }

        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();
        return response.toString();
    }
```