Medici Extractor in Python

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

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
rabbitmqhost = "localhost"

# name to show in rabbitmq queue list
exchange = "medici"

# name to show in rabbitmq queue list
extractorName = "wordCount"

# username and password to connect to rabbitmq
username = None
password = None

# accept any type of file that is text
messageType = "*.file.text.#"

# secret key used to connect to medici, this will eventually be
# part of the message received.
secretKey = "r1ek3rs"
```

A convenience function to return status messages to Medici

status_update

```
def status_update(channel, header, fileid, status):
    """Send notification on message bus with update"""
    global extractorName, exchange
    logger.debug("[%s] : %s", fileid, status)
    statusreport = {}
    statusreport['file_id'] = fileid
    statusreport['extractor_id'] = extractorName
    statusreport['status'] = status
    statusreport['start'] = time.strftime('%Y-%m-%dT%H:%M:%S')
    channel.basic_publish(exchange=exchange,
                          routing_key=header.reply_to,
                          properties=pika.BasicProperties(correlation_id = header.correlation_id),
                          body=json.dumps(statusreport))
```

Next we define a function that will listen for messages on the message bus

connect_message_bus

```
def connect_message_bus():
   """Connect to message bus and wait for messages"""
   global extractorName, username, password, messageType, exchange
   # connect to rabbitmq using input username and password
   if (username is None or password is None):
       connection = pika.BlockingConnection()
   else:
       credentials = pika.PlainCredentials(username, password)
       parameters = pika.ConnectionParameters(host=rabbitmqhost, credentials=credentials)
       connection = pika.BlockingConnection(parameters)

   # connect to channel
   channel = connection.channel()

   # declare the exchange in case it does not exist
   channel.exchange_declare(exchange=exchange, exchange_type='topic', durable=True)

   # declare the queue in case it does not exist
   channel.queue_declare(queue=extractorName, durable=True)
   # connect queue and exchange
   channel.queue_bind(queue=extractorName, exchange=exchange, routing_key=messageType)
   # create listener
   channel.basic_consume(on_message, queue=extractorName, no_ack=False)
   # start listening
   logger.info("Waiting for messages. To exit press CTRL+C")
   try:
       channel.start_consuming()
   except KeyboardInterrupt:
       channel.stop_consuming()
   # close connection
   connection.close()
```

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.

on_message

```
def on_message(channel, method, header, body):
    """When message is received do the following steps:
    1. download the file
    2. launch extractor function"""

    global logger, extractorName
    inputfile=None
    fileid=0

    try:
        # parse body back from json
        jbody=json.loads(body)
        host=jbody['host']
        fileid=jbody['id']
        intermediatefileid=jbody['intermediateId']
        if not (host.endswith('/')):
            host += '/'

        # tell everybody we are starting to process the file
        status_update(channel, header, fileid, "Started processing file")

        # download file
        inputfile = download_file(channel, header, host, secretKey, fileid, intermediatefileid)
        # call actual extractor function
        process_file(channel, header, host, secretKey, fileid, intermediatefileid, inputfile)

        # notify rabbitMQ we are done processsing message
        channel.basic_ack(method.delivery_tag)
    except subprocess.CalledProcessError as e:
        msg = str.format("Error processing [exit code=%d]\n%s", e.returncode, e.output)
        logger.exception("[%s] %s", fileid, msg)
        status_update(channel, header, fileid, msg)
    except:
        logger.exception("[%s] error processing", fileid)
        status_update(channel, header, fileid, "Error processing")
    finally:
        status_update(channel, header, fileid, "Done")
        if inputfile is not None:
            try:
                os.remove(inputfile)
            except OSError:
                pass
            except UnboundLocalError:
                pass
```

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.

download_file

```
def download_file(channel, header, host, key, fileid, intermediatefileid):
    """Download file to be processed from Medici"""

    status_update(channel, header, fileid, "Downloading file.")

    # fetch data
    url=host + 'api/files/' + intermediatefileid + '?key=' + key
    r=requests.get('%sapi/files/%s?key=%s' % (host, intermediatefileid, key),
                   stream=True)
    r.raise_for_status()
    (fd, inputfile)=tempfile.mkstemp()
    with os.fdopen(fd, "w") as f:
        for chunk in r.iter_content(chunk_size=10*1024):
            f.write(chunk)
    return inputfile
```

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.

process_file

```
def process_file(channel, header, host, key, fileid, intermediatefileid, inputfile):
    """Count the number of words in text file"""
    status_update(channel, header, fileid, "Counting words in file.")
    # call actual program
    result = subprocess.check_output(['wc', inputfile], stderr=subprocess.STDOUT)
    (lines, words, characters, filename) = result.split()

    # store results as metadata
    metadata={}
    metadata['lines']=lines
    metadata['words']=words
    metadata['characters']=characters
    headers={'Content-Type': 'application/json'}
    r = requests.post('%sapi/files/%s/metadata?key=%s' % (host, fileid, key),
                      headers=headers,
                      data=json.dumps(metadata));
    r.raise_for_status()
```

The following code will process an incoming video and upload a preview of the video

create_image_thumbnail

```
def create_image_thumbnail(inputfile, ext, size, host, fileid, *args):
        global logger

        (fd, thumbnailfile)=tempfile.mkstemp(suffix='.' + ext)
        try:
                # convert image to right size
                subprocess.check_output(['convert',  inputfile, '-resize', size, thumbnailfile],
stderr=subprocess.STDOUT)

                if(os.path.getsize(thumbnailfile) == 0):
                        raise Exception("File is empty.")

                # upload preview image
                url=host + 'api/fileThumbnail?key=' + playserverKey
                r = requests.post(url, files={"File" : open(thumbnailfile, 'rb')})
                r.raise_for_status()
                thumbnailid = r.json()['id']

                # associate uploaded thumbnail with original file
                url=host + 'api/files/' + fileid + '/thumbnails/' + thumbnailid + '?key=' + playserverKey
                r = requests.post(url);
                r.raise_for_status()

                logger.debug("[%s] created thumbnail of type %s", fileid, ext)
        finally:
                try:
                        os.remove(thumbnailfile)
                except:
                        pass
```