RecursiveMetadata

Index

Introduction

After the MetadataDiscussion page was created, Jukka Zitting offered an example of how to get to recursive metadata when parsing with an AutoDetectParser, and later updated that example with how to get both text and metadata for nested documents using the AutoDetectParser.

If you parse an archive (zip, tar, etc.) the parsed document contains other documents, and any of those documents could also be archives containing other documents, and so on. The example on this page shows you how to do the following:

- Set up the parse context so nested documents will be parsed.
- Wrap the AutoDetectParser so you can get the text and metadata for each nested document.

Jukka's Example

Here is the full source for Jukka's example for how to get access to nested metadata and document body text. This example writes the metadata and body text for each nested document to standard output. More details about how Jukka's example works are in subsections below.

```
public static void main(String[] args) throws Exception {
     Parser parser = new RecursiveMetadataParser(new AutoDetectParser());
     ParseContext context = new ParseContext();
     context.set(Parser.class, parser);

     ContentHandler handler = new DefaultHandler();
     Metadata metadata = new Metadata();

     InputStream stream = TikaInputStream.get(new File(args[0]));
     try {
         parser.parse(stream, handler, metadata, context);
     } finally {
         stream.close();
     }
 }

 private static class RecursiveMetadataParser extends ParserDecorator {

     public RecursiveMetadataParser(Parser parser) {
         super(parser);
     }

     @Override
     public void parse(
             InputStream stream, ContentHandler ignore,
             Metadata metadata, ParseContext context)
             throws IOException, SAXException, TikaException {
         ContentHandler content = new BodyContentHandler();
         super.parse(stream, content, metadata, context);

         System.out.println("----");
         System.out.println(metadata);
         System.out.println("----");
         System.out.println(content.toString());
     }
 }
```

Main from Jukka's Example

Setting up Recursive Parsing

```
public static void main(String[] args) throws Exception {
     Parser parser = new RecursiveMetadataParser(new AutoDetectParser());
     ParseContext context = new ParseContext();
     context.set(Parser.class, parser);
```

The example starts by setting up recursive parsing. If you are parsing text files, word documents, etc. then you'll never notice if recursive parsing is enable or not. If you are parsing containers like zip files and tar.gz files, the only way to get the text for the files contained by the containers is to enable recursive parsing.

The way to enable recursive parsing is to create a ParseContext and add a parser to it as shown on the line `context.set(Parser.class, parser)`. This is the parser that will be used to parse any nested documents.

In this case the parser is a RecursiveMetadataParser that is a wrapper around an AutoDetectParser. The RecursiveMetadata parser is part of Jukka's example and more details are given below.

Parsing a File

```
        ContentHandler handler = new DefaultHandler();
        Metadata metadata = new Metadata();

        InputStream stream = TikaInputStream.get(new File(args[0]));
        try {
            parser.parse(stream, handler, metadata, context);
        } finally {
            stream.close();
        }
```

The rest of the main function parses a file. The parser used to parse the root document is the same parser that was added to the ParseContext as the parser to use for nested documents.

Looking at the Tika API (http://tika.apache.org/0.7/api/), I don't see a DefaultHandler class or a TikaInputStream. In the place of DefaultHandler you could use BodyContentHandler, and in the place of TikaInputStream you could use FileInputStream.

Jukka's RecursiveMetadata Parser

RecursiveMetadataParser Constructor

```
    private static class RecursiveMetadataParser extends ParserDecorator {
        public RecursiveMetadataParser(Parser parser) {
            super(parser);
        }
```

The RecursiveMetadataParser extends ParserDecorator. All the constructor has to do is let the ParserDecorator superclass know which parser object is being decorated.

RecursiveMetadataParser parse

```
        @Override
        public void parse(
                InputStream stream, ContentHandler ignore,
                Metadata metadata, ParseContext context)
                throws IOException, SAXException, TikaException {
            ContentHandler content = new BodyContentHandler();
            super.parse(stream, content, metadata, context);

            System.out.println("----");
            System.out.println(metadata);
            System.out.println("----");
            System.out.println(content.toString());
        }

    }
```

The parse method is where you get access to the metadata and the body text. When the parser set in ParseContext is used to parse a nested document, a new Metadata object is created and passed to the parse method. Since the example put a RecursiveMetadataParser in the ParseContext, RecursiveMetadataParser's parse method is called. Before calling `super.parse`, the metadata object is empty. After `super.parse` returns, the metadata object contains all of the metadata the decorated parser found and `System.out.println(metadata)` prints all of the metadata to standard output.

By creating a new BodyContentHandler and passing that to `super.parse`, the text for each document is captured without mixing it with text from other documents.

Tracking how far down the Rabbit Hole you have gone

When using the code above, if you have a container format that contains another container, you may wish to keep track of where in the stack you are. To do that, you'd want code something like:

```
    private static class RecursiveTrackingMetadataParser extends ParserDecorator {
        private String location;
        private int unknownCount = 0;

        public RecursiveTrackingMetadataParser(Parser parser, String location) {
            super(parser);
            this.location = location;
            if (! this.location.endsWith("/")) {
               this.location += "/";
            }
        }

        @Override
        public void parse(
                InputStream stream, ContentHandler ignore,
                Metadata metadata, ParseContext context)
                throws IOException, SAXException, TikaException {
            // Work out what this thing is
            String objectName = null;
            if (metadata.get(TikaMetadataKeys.RESOURCE_NAME_KEY) != null) {
               objectName = metadata.get(TikaMetadataKeys.RESOURCE_NAME_KEY);
            } else if (metadata.get(TikaMetadataKeys.EMBEDDED_RELATIONSHIP_ID) != null) {
               objectName = metadata.get(TikaMetadataKeys.EMBEDDED_RELATIONSHIP_ID);
            } else {
               objectName = "embedded-" + (++unknownCount);
            }
            String objectLocation = this.location + objectName;

            // Fetch the contents, and recurse if possible
            ContentHandler content = new BodyContentHandler();
            Parser preContextParser = context.get(Parser.class);
            context.set(Parser.class, new RecursiveTrackingMetadataParser(getWrappedParser(), objectLocation));
            super.parse(stream, content, metadata, context);
            context.set(Parser.class, preContextParser);

            // Report what this one is
            System.out.println("----");
            System.out.println("Resource is " + objectLocation);
            System.out.println("----");
            System.out.println(metadata);
            System.out.println("----");
            System.out.println(content.toString());
        }
    }
```

Surprise! Zips Have Text Too!

The great thing about AutoDetectParser is that it can parse and extract text from almost anything. In particular, it can parse zip, tar, tar.bz2, and other archives that contain documents. If you have a zip file with 100 text files in it, using Jukka's example code you can get the text and metadata for each file nested inside of the zip file. What you might not expect is that you also get metadata and body text for the zip file itself.

Maybe this doesn't surprise you at all. My first reaction when I saw both metadata AND text for the zip file itself was "What text could a zip file possibly have?" My naive assumption was that a zip file wouldn't contain any text, and my assumption was wrong.

I was thinking that a zip, tar, or other archive file was simply a container for other files, and so didn't have any text of its own. Tika looks at archives differently; Tika sees an archive as being like a directory in a file system, and the text for an archive is a list of the contents of the archive.

If you have a zip file that contains 100 text files, after using the code on this page to get the text and metadata for each file, you will get the text and metadata for 101 files: 100 text files, and 1 zip file. The text for the zip file will list the names for each of the 100 text files it contains.

If you aren't interested in seeing text and metadata for the zip file itself, you'll want to take a look at `metadata.get(Metadata.CONTENT_TYPE)` for each file Tika parses so you can skip the archives themselves. For a zip file, the content type is "application/zip".

Integration of the RecursiveParserWrapper into Tika

A RecursiveParserWrapper that is based on Jukka and Nick's example above was added to Tika as of 1.7.

The wrapper returns a list of Metadata objects – the first contains the metadata+content for the container document and the rest contain the metadata+content for each embedded document. The content of each document is stored in "X-TIKA:content", and the embedded document's location in the container document is stored in "X-TIKA:embedded_resource_path" (e.g. "embedded-1/embed1.zip/embed2.zip/embed3.pdf").

A downside to the wrapper is that it breaks the Tika goal of streaming output – the wrapper caches all metadata+content in memory. This wrapper must be used with care.

As of Tika 1.7, a JSONified view of this output was integrated into tika-app (the -J option) and tika-server ("/rmeta").

This format serves as the basis for the upcoming tika-eval module that will help with comparisons of the output of different versions of Tika or other content extractors.