ModifyingContentWithHandlersAndMetadataFilters

Modifying Content with the AutoDetectParserConfig, Handlers, Metadata Filters and Metadata WriteFilters

There are several ways to modify, limit or process content during or after the parse. In the following, we describe the four main categories.

1. ContentHandlers

These are applied during the parse by classes that implement `org.xml.saxContentHandler`. A small handful may cache contents in memory. One small risk for these is that there's no guarantee that parsers will pass in meaningful amounts of text in the call to `characters()`; theoretically, a parser could write one character at a time, which would render a regex matching handler useless.

Programmatically, users have control to use any of the `ContentHandlers` in `tika-core` or they can write their own `ContentHandlers`. If doing this, make sure to consider the `ContentHandlerDecorator` which allows overriding only the methods you need; also consider using the `TeeContentHandler`, which allows multiple handlers to be run during the parse.

An example of using the `TeeContentHandler` to add a language detection handler to the regular `ToXMLContentHandler`:

TeeContentHandler

```
...
ContentHandler xmlHandler = new ToXMLContentHandler();
LanguageHandler langHandler = new LanguageHandler();
ContentHandler tee = new TeeContentHandler(xmlHandler, langHandler);
parser.parse(stream, tee, metadata, context);
LanguageResult result = langHandler.getLanguage();
metadata.set(TikaCoreProperties.LANGUAGE, result.getLanguage());
...
```

Some common content handlers are specified for tika-server's `/tika json output` and the `/rmeta` endpoint by appending "/xml", "/text", "/html", "/body" or "/ignore" to the endpoint.

To set custom `ContentHandlerDecorators` via `tika-config.xml`, set the `ContentHandlerDecoratorFactory` in the `<autoDetectParserConfig/>` element in `tika-config.xml`.

In this example, we're calling a test class that simply upcases all characters in the content handler.

ContentHandlerDecoratorFactory

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <!-- we're including the <parsers/> element to show that it is a separate element from the
       autoDetectParserConfig element.  If it is not included, the standard default parser will
       be used -->
  <parsers>
    <parser class="org.apache.tika.parser.DefaultParser">
      <parser-exclude class="org.apache.tika.parser.microsoft.OfficeParser"/>
    </parser>
    <parser class="org.apache.tika.parser.microsoft.OfficeParser">
      <params>
        <param name="byteArrayMaxOverride" type="int">700000000</param>
      </params>
    </parser>
  </parsers>
  <!-- note that the autoDetectParserConfig element is separate from the <parsers/> element.
       The composite parser built in the <parsers/> element is used as the base parser
       for the AutoDetectParser. -->
  <autoDetectParserConfig>
    <!-- note that this is a test class only available in tika-core's test-jar as an example.
         Specify your own custom factory here -->
    <contentHandlerDecoratorFactory class="org.apache.tika.sax.UpcasingContentHandlerDecoratorFactory"/>
  </autoDetectParserConfig>
</properties>
```

2. Metadata Filters

These are applied at the end of the parse. These are intended to modify the contents of a metadata object for different purposes:

1. Enrich the data (similar to a ContentHandler) -- these metadata filters might run language detection on the cached contents at the end of the parse.
2. Modify the metadata contents – one might want to run a regex over a specific field and extract only the information matching a regex, for example.
3. Modify the metadata keys – If you need to rename metadata keys before emitting the object to, say, OpenSearch, you can use the FieldNameMappingFilter
4. Limit the metadata fields -- let's say you only want `dc:title` and `text`, you can use these: ExcludeFieldMetadataFilter or IncludeFieldMetadataFilter **NOTE:** these were created before we had MetadataWriteFilters; those are probably a better option for this behavior.

Metadata filters are specified in the `<metadataFilters/>` element in `tika-config.xml`. They are run in order, and order matters.

See TikaServerEndpointsCompared for which endpoints apply metadataFilters in tika-server. Metadata filters are applied in tika-pipes and tika-app when using the -J option. MetadataFilters are not applied when Tika streams output.

FieldNameMappingFilter

This is used to select fields to include and to rename fields from the Tika names to preferred names. This was initially designed for modifying field names before emitting document to OpenSearch or Solr.

FieldNameMappingFilter

```
<?xml version="1.0" encoding="UTF-8" ?>
<properties>
  <metadataFilters>
    <metadataFilter class="org.apache.tika.metadata.filter.FieldNameMappingFilter">
      <params>
        <excludeUnmapped>true</excludeUnmapped>
        <mappings>
          <mapping from="X-TIKA:content" to="content"/>
          <mapping from="Content-Length" to="length"/>
          <mapping from="dc:creator" to="creators"/>
          <mapping from="dc:title" to="title"/>
          <mapping from="Content-Type" to="mime"/>
          <mapping from="X-TIKA:EXCEPTION:container_exception" to="tika_exception"/>
        </mappings>
      </params>
    </metadataFilter>
  </metadataFilters>
</properties>
```

DateNormalizingMetadataFilter

Some file formats store timezone, others don't. By default, OpenSearch and Solr need timezones. This filter respects dates with timezones, and blindly adds a UTC timezone to dates that do not have a time zone.

DateNormalizingMetadataFilter

```
<?xml version="1.0" encoding="UTF-8" ?>
<properties>
  <metadataFilters>
    <!-- depending on the file format, some dates do not have a timezone. This
     filter arbitrarily assumes dates have a UTC timezone and will format all
     dates as yyyy-MM-dd'T'HH:mm:ss'Z' whether or not they actually have a timezone.
     -->
    <metadataFilter class="org.apache.tika.metadata.filter.DateNormalizingMetadataFilter"/>
  </metadataFilters>
</properties>
```

GeoPointMetadataFilter

If a metadata object has a `TikaCoreProperties.LATITUDE` and a `TikaCoreProperties.LONGITUDE`, this concatenates those fields with a comma delimiter as `LATITUDE,LONGITUDE` and adds that value to the field specified by `geoPointFieldName`. **Note:** This was added in Tika 2.5.1.

GeoPointMetadataFilter

```
<?xml version="1.0" encoding="UTF-8" ?>
<properties>
  <metadataFilters>
    <metadataFilter class="org.apache.tika.metadata.filter.GeoPointMetadataFilter">
      <params>
        <-- default: "location" -->
        <geoPointFieldName>myGeoPoint</geoPointFieldName>
      </params>
    </metadataFilter>
  </metadataFilters>
</properties>
```

TikaEvalMetadataFilter

If the tika-eval-core jar is on the classpath, this filter should be added automatically. Users may specify it as below. This runs Tika's custom version of OpenNLP's language detector and includes counts for tokens, unique tokens, alphabetic tokens and the "oov" (% out of vocabulary) statistic. See TikaEval for more details on the `tika-eval-app`.

TikaEvalMetadataFilter

```
<?xml version="1.0" encoding="UTF-8" ?>
<properties>
  <metadataFilters>
    <metadataFilter class="org.apache.tika.eval.core.metadata.TikaEvalMetadataFilter"/>
  </metadataFilters>
</properties>
```

LanguageDetection

Two language detectors have a metadata filter option (OpenNLPMetadataFilter and the OptimaizeMetadataFilter). These are applied to the `X-TIKA:content` field at the end of the parse. This is an example of specifying the OptimaizeLanguageDetector. The language id will be added to the metadata object with the TikaCoreProperties.TIKA_DETECTED_LANGUAGE key.

LanguageDetection

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <metadataFilters>
    <metadataFilter class="org.apache.tika.langdetect.optimaize.metadatafilter.OptimaizeMetadataFilter">
      <params>
        <maxCharsForDetection>10000</maxCharsForDetection>
      </params>
    </metadataFilter>
  </metaFilters>
</properties>
```

ClearByMimeMetadataFilter

When using the `RecursiveParserWrapper` (the `/rmeta` endpoint in `tika-server` or the `-J` option in `tika-app`), you can delete metadata objects for specific file types.

ClearByMimeMetadataFilter

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <metadataFilters>
    <metadataFilter class="org.apache.tika.metadata.filter.ClearByMimeMetadataFilter">
      <params>
                <!-- this will remove metadata objects for jpegs and pdfs; more seriously,
              this may be useful for image files or emf or wmf depending on your use case -->
        <mimes>
          <mime>image/jpeg</mime>
          <mime>application/pdf</mime>
        </mimes>
      </params>
    </metadataFilter>
  </metaFilters>
</properties>
```

IncludeFieldMetadataFilter

This removes all other metadata fields after the parse except those specified here.

MetadataFilters

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <metadataFilters>
    <metadataFilter class="org.apache.tika.metadata.filter.IncludeFieldMetadataFilter">
      <params>
        <include>
          <field>X-TIKA:content</field>
          <field>extended-properties:Application</field>
          <field>Content-Type</field>
        </include>
      </params>
    </metadataFilter>
  </metadataFilters>
</properties>
```

3. Metadata Write Filters

These filters are applied during the parse.

The primary goal of the metadata write filters is to limit the the amount of data written to a metadata object for two purposes:

1. Limit the total number of bytes written to a metadata objects (prevent DoS from files with large amounts of metadata)
2. Limit the fields written to a metadata object (decrease bytes held in memory during the parse and decrease the bytes sent over the wire/written to a file after the parse

To configure the StandardWriteFilter, set the properties in its factory in the `<autoDetectParserConfig/>` element in the `tika-config.xml` file.

StandardWriteFilter

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <autoDetectParserConfig>
    <metadataWriteFilterFactory class="org.apache.tika.metadata.writefilter.StandardWriteFilterFactory">
      <params>
                <!-- all measurements are in UTF-16 bytes. If any values are truncated,
                        TikaCoreProperties.TRUNCATED_METADATA is set to true in the metadata object -->

        <!-- the maximum size for a metadata key. -->
        <maxKeySize>1000</maxKeySize>

        <!-- max total size for a field in UTF-16 bytes.  If a field has multiple values,
                        their lengths are summed to calculate the field size. -->
        <maxFieldSize>10000</maxFieldSize>

        <!-- max total estimated byte is a sum of the key sizes and values -->
        <maxTotalEstimatedBytes>100000</maxTotalEstimatedBytes>

        <!-- limit the count of values for multi-valued fields -->
        <maxValuesPerField>100</maxValuesPerField>
        <!-- include only these fields. NOTE, however that there a several fields that are
                         important to the parse process and these fields are always allowed in addition
                         (see ALWAYS_SET_FIELDS and ALWAYS_ADD_FIELDS in the StandardWriteFilter -->
        <includeFields>
          <field>dc:creator</field>
          <field>dc:title</field>
        </includeFields>
      </params>
    </metadataWriteFilterFactory>
  </autoDetectParserConfig>
</properties>
```

If you need different behavior, implement a `WriteFilterFactory`, add it to your classpath and specify it in the `tika-config.xml`.

4. AutoDetectParserConfig

We've mentioned briefly above some of the factories that can be modified in the `AutoDetectParserConfig`. There are other parameters that can be used to modify the behavior of the `AutoDetectParser` via the `tika-config.xml`. The `AutoDetectParser` is built from/contains the `<parsers/>` element (or SPI if no `<parsers/>` element is specified) in the `tika-config`. Because of this, the configuration of the `AutoDetectParser` differs from the component parsers that it wraps – the `AutoDetectParser` uses its own `<autoDetectParserConfig/>` element at the main level inside the `<properties/>` element.

AutoDetectParserConfig

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <autoDetectParserConfig>
    <params>
      <!-- if the incoming metadata object has a ContentLength entry and it is larger than this
           value, spool the file to disk; this is useful for some file formats that are more efficiently
           processed via a file instead of an InputStream -->
      <spoolToDisk>100000</spoolToDisk>
      <!-- the next four are parameters for the SecureContentHandler -->
      <!-- threshold used in zip bomb detection. This many characters must be written
           before the maximum compression ratio is calculated -->
      <outputThreshold>10000</outputThreshold>
      <!-- maximum compression ratio between output characters and input bytes -->
      <maximumCompressionRatio>100</maximumCompressionRatio>
      <!-- maximum XML element nesting level -->
      <maximumDepth>100</maximumDepth>
      <!-- maximum embedded file depth -->
      <maximumPackageEntryDepth>100</maximumPackageEntryDepth>
      <!-- as of Tika &gt; 2.7.0, you can skip the check and exception for a zero-byte inputstream-->
      <throwOnZeroBytes>false</throwOnZeroBytes>
    </params>
    <!-- as of Tika 2.5.x, this is the preferred way to configure digests -->
    <digesterFactory class="org.apache.tika.parser.digestutils.CommonsDigesterFactory">
      <params>
        <markLimit>100000</markLimit>
        <!-- this specifies SHA256, base32 and MD5 -->
        <algorithmString>sha256:32,md5</algorithmString>
      </params>
    </digesterFactory>
  </autoDetectParserConfig>
</properties>
```

TODO: add an example of the EmbeddedDocumentExtractorFactory

TODO: add a 5th? section for writelimiting