This page provides a number of examples on how to use the various Tika APIs. All of the examples shown are also available in the Tika Example module in SVN.

Parsing

Tika provides a number of different ways to parse a file. These provide different levels of control, flexibility, and complexity.

Parsing using the Tika Facade

The Tika facade, provides a number of very quick and easy ways to have your content parsed by Tika, and return the resulting plain text

```
public String parseToStringExample() throws IOException, SAXException,
TikaException {
    Tika tika = new Tika();
    try (InputStream stream =
ParsingExample.class.getResourceAsStream("test.doc")) {
        return tika.parseToString(stream);
    }
}
```

Parsing using the Auto-Detect Parser

For more control, you can call the Tika Parsers directly. Most likely, you'll want to start out using the Auto-Detect Parser, which automatically figures out what kind of content you have, then calls the appropriate parser for you.

```
public String parseExample() throws IOException, SAXException, TikaException {
    AutoDetectParser parser = new AutoDetectParser();
    BodyContentHandler handler = new BodyContentHandler();
    Metadata metadata = new Metadata();
    try (InputStream stream =
ParsingExample.class.getResourceAsStream("test.doc")) {
        parser.parse(stream, handler, metadata);
        return handler.toString();
    }
}
```

Picking different output formats

With Tika, you can get the textual content of your files returned in a number of different formats. These can be plain text, html, xhtml, xhtml of one part of the file etc. This is controlled based on the ContentHandler you supply to the Parser.

Parsing to Plain Text

By using the BodyContentHandler, you can request that Tika return only the content of the document's body as a plain-text string.

```
public String parseToPlainText() throws IOException, SAXException, TikaException
{
    BodyContentHandler handler = new BodyContentHandler();

    AutoDetectParser parser = new AutoDetectParser();
    Metadata metadata = new Metadata();
    try (InputStream stream =
ContentHandlerExample.class.getResourceAsStream("test.doc")) {
        parser.parse(stream, handler, metadata);
        return handler.toString();
    }
}
```

Parsing to XHTML

By using the ToXMLContentHandler, you can get the XHTML content of the whole document as a string.

```
public String parseToHTML() throws IOException, SAXException, TikaException {
    ContentHandler handler = new ToXMLContentHandler();

    AutoDetectParser parser = new AutoDetectParser();
    Metadata metadata = new Metadata();
    try (InputStream stream =
ContentHandlerExample.class.getResourceAsStream("test.doc")) {
        parser.parse(stream, handler, metadata);
        return handler.toString();
    }
}
```

If you just want the body of the xhtml document, without the header, you can chain together a BodyContentHandler and a ToXMLContentHandler as shown:

```
public String parseBodyToHTML() throws IOException, SAXException, TikaException {
    ContentHandler handler = new BodyContentHandler(
            new ToXMLContentHandler());

    AutoDetectParser parser = new AutoDetectParser();
    Metadata metadata = new Metadata();
    try (InputStream stream =
ContentHandlerExample.class.getResourceAsStream("test.doc")) {
        parser.parse(stream, handler, metadata);
        return handler.toString();
    }
}
```

Fetching just certain bits of the XHTML

It possible to execute XPath queries on the parse results, to fetch only certain bits of the XHTML.

```
public String parseOnePartToHTML() throws IOException, SAXException, TikaException {
    // Only get things under html -> body -> div (class=header)
    XPathParser xhtmlParser = new XPathParser("xhtml", XHTMLContentHandler.XHTML);
    Matcher divContentMatcher =
xhtmlParser.parse("/xhtml:html/xhtml:body/xhtml:div/descendant::node()");
    ContentHandler handler = new MatchingContentHandler(
            new ToXMLContentHandler(), divContentMatcher);

    AutoDetectParser parser = new AutoDetectParser();
    Metadata metadata = new Metadata();
    try (InputStream stream =
ContentHandlerExample.class.getResourceAsStream("test2.doc")) {
        parser.parse(stream, handler, metadata);
        return handler.toString();
    }
}
```

Custom Content Handlers

The textual output of parsing a file with Tika is returned via the SAX ContentHandler you pass to the parse method. It is possible to customise your parsing by supplying your own ContentHandler which does special things.

Extract Phone Numbers from Content into the Metadata

By using the PhoneExtractingContentHandler, you can have any phone numbers found in the textual content of the document extracted and placed into the Metadata object for you.

Streaming the plain text in chunks

Sometimes, you want to chunk the resulting text up, perhaps to output as you go minimising memory use, perhaps to output to HDFS files, or any other reason! With a small custom content handler, you can do that.

```
public List<String> parseToPlainTextChunks() throws IOException, SAXException,
TikaException {
    final List<String> chunks = new ArrayList<>();
    chunks.add("");
    ContentHandlerDecorator handler = new ContentHandlerDecorator() {
        @Override
        public void characters(char[] ch, int start, int length) {
            String lastChunk = chunks.get(chunks.size() - 1);
            String thisStr = new String(ch, start, length);

            if (lastChunk.length() + length > MAXIMUM_TEXT_CHUNK_SIZE) {
                chunks.add(thisStr);
            } else {
                chunks.set(chunks.size() - 1, lastChunk + thisStr);
            }
        }
    };

    AutoDetectParser parser = new AutoDetectParser();
    Metadata metadata = new Metadata();
    try (InputStream stream =
ContentHandlerExample.class.getResourceAsStream("test2.doc")) {
        parser.parse(stream, handler, metadata);
        return chunks;
    }
}
```

Translation

Tika provides a pluggable Translation system, which allow you to send the results of parsing off to an external system or program to have the text translated into another language.

Translation using the Microsoft Translation API

In order to use the Microsoft Translation API, you need to sign up for a Microsoft account, get an API key, then pass the key to Tika before translating.

```
public String microsoftTranslateToFrench(String text) {
    MicrosoftTranslator translator = new MicrosoftTranslator();
    // Change the id and secret! See http://msdn.microsoft.com/en-
us/library/hh454950.aspx.
    translator.setId("dummy-id");
    translator.setSecret("dummy-secret");
    try {
        return translator.translate(text, "fr");
    } catch (Exception e) {
        return "Error while translating.";
    }
}
```

Language Identification

Tika provides support for identifying the language of text, through the LanguageIdentifier class.

```
public String identifyLanguage(String text) {
    LanguageIdentifier identifier = new LanguageIdentifier(text);
    return identifier.getLanguage();
}
```

Additional Examples

A number of other examples are also available, including all of the examples from the Tika In Action book. These can all be found in the Tika Example module in SVN.