



**Quanticate**  
A Passion For Excellence

Global Solutions from the World's Largest Data-Focused CRO

# What's with all the 1s and 0s?

Making sense of binary data at scale

# What's with the 1s and 0s?

- Detecting File Types
- Detecting Text
- A quick introduction to Apache Tika
- How things can go wrong at Scale...
- Working out if things are getting better or worse!
- Other tools to consider

# Detecting File Types

[www.quanticate.com](http://www.quanticate.com)

# Isn't detecting simple?

- Surely you just know what a file is on your computer?
- Well, probably on your computer, and maybe elsewhere?
- OK, so maybe people rename things, but it's close, no?
- Ah, the internet... But that's normally right isn't it?
- Hmm, well most web servers tell the truth, right?
- They wouldn't get it that wrong?
- A few percent of the internet, that's hardly that much?
- And people would never rename things by accident?
- Operating Systems would never “help”, would they?

# Filenames

- Filenames – normally, but not always have extensions
- There aren't that many extension combinations
- There are probably more file formats than that
- No official way to reserve an extension
- So everyone just picks a “sensible” one, and hopes that don't have (too many) clashes...
- What happens if you rename a file though?
- Or have a file without one?
- *Quick, but dirty, and may not be right...*

# Mime Magic

- Most file formats have a well known structure
- Most of these have a (mostly) unique pattern near the start
- These are often called Mime Magic Numbers
- In some cases, these are numbers
- More commonly, they're some sort of number / text / bit mask
  
- Ideally located at a fixed offset, even better, right at the start of the file
- *But not always...*

# More on Magic

- PDFs (should) start with `%PDF-`
- Microsoft Office OLE2 docs start with `0xd0cf11e0a1b11ae1`
- Most Zip files start with `PK\003\004`
- AIFF starts with `FORM????AI(FF|FC)` (mask 5-8)
- PE Executables normally have `PE\000\000` at 128 or 240
  
- Not all of these are true constants
- Not all of these are unique - `0xffffe` can be UTF-16LE or MP3
- Container formats – Zip can be Zip, OOXML, iWorks etc

# Container Formats

- Some file formats are actually containers, and can hold lots of different things in them
- For example, a **.zip** file could just be a zip of random files
- Or it could be a Microsoft OOXML file (eg **.docx**, **.pptx**)
- Or it could be an OpenDocument Format file (eg **.ods**)
- Or it could be an iWorks file (Numbers, Pages, Keynote)
- Or it could be an ePub file
- Or....
- A **.ogg** could be audio, video, text, or many!



# Dealing with Containers

- We can use Mime Magic to detect the container itself
- But to go beyond that, we need to actually parse the container, and look for special entries
- All the container based formats have some way of identifying the contents, of varying difficulty
- eg Check zip entries, look for **\_rels/.rels**, it's OOXML, then read that file to get the mimetype
- eg Check Ogg for **CMML**, use that to get the primary stream type, else count stream types

# Patterns inside the file

- Not all file formats have nice Mime Magic near-ish the start
- Many do have some sort of structure, eg repeating patterns within them
- Varies a great deal between file formats, and even between what's in a given collection of files
- If you manually classify a number of files of a given type, you can then use machine learning to build byte histograms and n-gram patterns for them, along with negative patterns too
- Then compare positive and negatives patterns to detect
- Needs training data sets and lots of pre-computation

# Pulling it all together

- Need to combine various techniques
- eg Use Mime Magic to spot it's a container, then use container-specific code to identify the type
- eg Use filename to tell the difference between two very similar different file types
- Using more techniques can increase quality of detection, but slows things down. Consider Speed vs Accuracy
- May need custom weightings, eg from machine learning
- Apache Tika combines different techniques for you via **DefaultDetector**, which auto-loads all available detections

# Detecting Text

[www.quanticate.com](http://www.quanticate.com)

# Encodings 101

- Many different ways to encode text in a file
- “A” could be **0x41** (ascii,utf8 etc), **0x0041** (utf16le), **0x4100** (utf16be), **0xC1** (ebcdic)
- 1-byte-per-character encodings historically very common
- But means that you need lots of different encodings to cope with different languages and character sets
- **0xE1** – could be: **á α c Ɔ ف ۱** (or something else too!)
- Some formats include what encoding they've used
- But many key ones, including plain text, do not!

# Languages

- Different languages have different common patterns of letters, based on words, spellings and patterns
- If you see accents like á è ç then it probably isn't English
- If you see a word starting with an S, it probably isn't Spanish, but if you see lots starting “ES” it might be
- You can look for these patterns, and use those to identify what language some text might be in
- Really needs quite a bit of text to work on though, it's very hard to make meaningful guesses on just a few letters!

# n-grams

- Wikipedia says “An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a  $(n - 1)$ –order Markov model”
- Basically, for us, it's all the possible character combinations (including start + end markers) along with their frequency
- The “n” is the size
- Trigrams of “hello” are [ ]he, hel, ell, llo, lo[ ]
- Quadgrams of “hello” are [ ]hel, ello, llo[ ]
- Can be used to identify both language and encoding

# False Positives, Problems

- For encoding detection to work, your tool (eg Tika) needs to recognise the file as Plain Text
- Too many control characters near the start can cause Tika and friends to decide it isn't Plain Text, so won't detect
- Some encodings are very similar, hard to tell apart
- For short runs of text, very hard to be sure what it is
- Same pattern can crop up in different languages
- Same pattern could occur between different languages when in different encodings



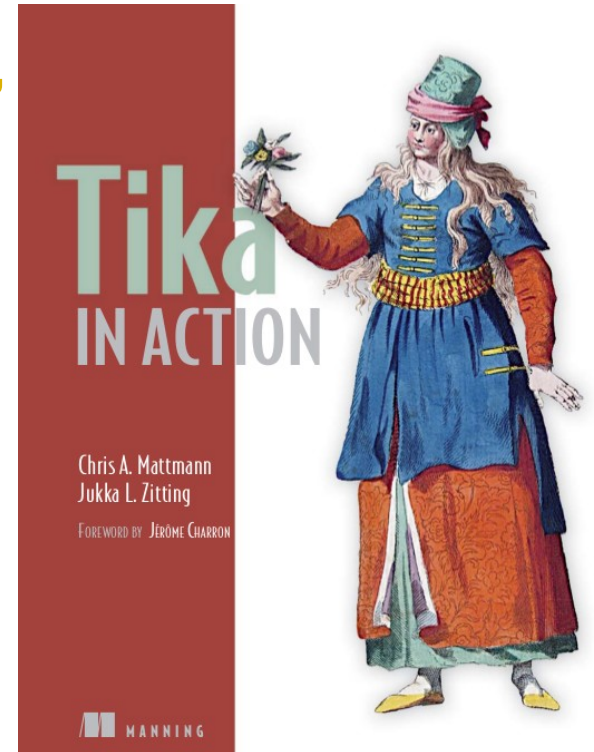
# Apache Tika in a nutshell

[www.quanticate.com](http://www.quanticate.com)

# Apache Tika in a nutshell

“small, yellow and leech-like, and probably the oddest thing in the Universe”

- Like a Babel Fish for content!
- Helps you work out what sort of thing your content (1s & 0s) is
- Helps you extract the metadata from it, in a consistent way
- Lets you get a plain text version of your content, eg for full text indexing
- Provides a rich (XHTML) version too



# (Some) Supported Formats



- Microsoft Office – Word, Excel, PowerPoint, Works, Publisher, Visio – Binary and OOXML formats
- OpenDocument (OpenOffice),
- iWorks – Keynote, Pages, Numbers
- HTML, XHTML, XML, PDF, RTF, Plain Text, CHM Help
- Compression / Archive – Zip, Tar, Ar, 7z, bz2, gz etc
- Atom, RSS, ePub                      Lots of Scientific formats
- Audio – MP3, MP4, Vorbis, Opus, Speex, MIDI, Wav
- Image – JPEG, TIFF, PNG, BMP, GIF, ICO

[www.quanticate.com](http://www.quanticate.com)

# Ways of calling Tika

- Tika-App – command line tool
- Pure Java – Tika Facade – simple way from Java
- Pure Java – Direct – full control over what happens
- OSGi – all your dependencies nicely wrapped up
- Forked Parser – parsing in a different JVM
- JAX-RS Network Server – RESTful interface to Tika
- SOLR Plugin – Tika parsing from within SOLR upload
- Anything you want to write yourself!
- (These are just the main ways)

# Tika App

- Single runnable jar, containing all of Tika + dependencies
- GUI mode, ideal for quick testing and demos
- CLI mode, suitable for calling from non-Java programs
- Detection: `--detect`
- Metadata: `--metadata`
- Plain Text: `--text`
- XHTML: `--xml`
- Information on available mimetypes, parsers etc
- JVM startup costs means not the fastest for lots of docs

# JAX-RS Network Server

- JAX-RS – JSR-311 RESTful network server
- Powered by Apache CXF
- The tika-server jar can be run, starts up the server
- Content is sent via HTTP PUT calls to service URLs
- Supports detection, metadata extraction, plain text, xhtml, and a few other things too, plus info on parsers/formats/etc
- See <https://wiki.apache.org/tika/TikaJAXRS>
- Recommended way to call Tika from non-Java languages
- Runs in a new JVM

# Tika Facade (Java)

- Very simple way to call Tika
- Tika facade class – `org.apache.tika.Tika`
- Detection from streams, bytes, files, urls etc
- Parsing to Strings, Readers, Plain Text or XHTML
- By default, uses all available parsers and detectors
- Requires that all the Tika jars, and their dependencies are present on the classpath, and don't clash
- Harder to check if that hasn't worked properly, eg missing jars or jar clashes

# Java – Direct calls

- Slightly more lines of code, but you get full control
- Normally start with `org.apache.tika.TikaConfig`
- From that fetch Detectors, Parsers, Mime Types list etc
- Typically want to use `AutoDetectParser` to actually parse
- If possible, wrap your content with a `TikaInputStream` before parsing – create from `InputStream` or `File`
  
- Still needs all jars on classpath
- Ask `DefaultParser` what Parsers + mimetypes are available



# Java – Forked Parser

- By default, Tika runs in the same JVM as the rest of your code, so if Tika (or a library it depends on) breaks, so does the rest of your JVM
- This shouldn't happen, but some broken files can trigger parsers to run out of memory, or loop forever
- If you're doing internet scale things, this becomes all the more likely to be hit eventually!
- Forked Parser fires up a new JVM, sends over Tika + dependencies, then calls to that JVM to do the parsing
- If that crashed, doesn't affect your main JVM

# What Tika tries to do

- Tika aims to map file format specific metadata onto a common, consistent set, based on well known standards
- Tika aims to provide semantically meaningful, but not cluttered XHTML
- It isn't supposed to be a perfect rendition of the original file into XHTML
- It is supposed to be simple and clean
- eg Word Parser reports tables and style names, but not Word-style HTML full of fonts +colours
- eg Excel Parser returns what's there, not a CSV

# Extending Tika

- Mimetypes: Tika has >1400 defined, not all with magic
- You can define your own in the same structure, put a [org/apache/tika/mime/custom-mimetypes.xml](http://org.apache.tika.mime/custom-mimetypes.xml) on classpath
- Parsers: Tika has >70 parsers available, some formats have multiple which can be used
- You can write your own quickly and register them for use, see [http://tika.apache.org/1.8/parser\\_guide.html](http://tika.apache.org/1.8/parser_guide.html) for how
- Basic parser is 13 lines of code, work involved depends on what libraries you have available for your file format
- Tika Config XML allows you control of which parser to use

# Embedded Resources

[www.quanticate.com](http://www.quanticate.com)

# Not just containers

- Container formats like Zip contain embedded resources, the files within them
- Many office documents support embedded resources too
- Microsoft Office documents can have other documents embedded within them, eg Excel in PowerPoint
- Most of the document formats support embedding images within the document, and many video too
- Where possible, Tika gives you the embedded resource, and indicates in the XHTML where it came from
- Up to you how / where / if to handle these resources

# Going wrong at Scale...

[www.quanticate.com](http://www.quanticate.com)

# Lots of Data is Junk

- At scale, you're going to hit lots of edge cases
- At scale, you're going to come across lots of junk or corrupted documents
- 1% of a lot is still a lot...
- Bound to find files which are unusual or corrupted enough to be mis-identified
- You need to plan for failures!

# Unusual Types

- If you're working on a big data scale, you're bound to come across lots of valid but unusual + unknown files
- You're never going to be able to add support for all of them!
- May be worth adding support for the more common “uncommon” unsupported types
- Which means you'll need to track something about the files you couldn't understand
- If Tika knows the mimetype but has no parser, just log the mimetype
- If mimetype unknown, maybe log first few bytes



# Failure at Scale

- Tika will sometimes mis-identify something, so sometimes the wrong parser will run and object
- Some files will cause parsers or their underlying libraries to do something silly, such as use lots of memory or get into loops with lots to do
- Some files will cause parsers or their underlying libraries to OOM, or infinite loop, or something else bad
- If a file fails once, will probably fail again, so blindly just re-running that task again won't help

# Failure at Scale continued...



- You'll need approaches that plan for failure
- Consider what will happen if a file locks up your JVM, or kills it with an OOM
- Forked Parser may be worth using
- Running a separate Tika Server could be good
- Depending on work needed, could have a smaller pool of Tika Server instances for big data code to call
- Think about failure modes, then think about retries (or not)
- Track common problems, report and fix them!


[www.quanticate.com](http://www.quanticate.com)

# Getting Better? Or Worse?

[www.quanticate.com](http://www.quanticate.com)

# When things go wrong

Taking a close look at the forest or open meadows reveals that there are often subtle differences in plant species across a wide landscape. Unique micro-climates, exposure to the sun, soil types, moisture availability, and a variety of other factors influence the types of plant species present in any given location. Changes in any of these factors will cause changes to



BGQOTM G IRUYK RUUQ GZ ZNK LUXKYZ UX UVKT SKGJU]Y  
XK\KGRY ZNGZ ZNKXX GXK ULZKT Y[HZRK JOLLKXKTIKY OT VRGTZ  
YVKIOKY GIXUYY G ]OJK RGTJYIGVK% CTOW[K SOIXU-  
IROSGZKY\$K^VUY[XK ZU ZNK Y[T\$ YUOR Z\_VKY\$ SUOYZ[XK  
G\GORGHORZ\_\$GTJ G \GXOKZ\_ UL UZNXK LGIZUXY OTLR[KTIK ZNK  
Z\_VKY UL VRGTZ YVKIOKY VXKYKTZ OT GT\_MO\KT RUIGZOUT%  
4NGTMKY OT GT\_ UL ZNKYK LGIZUXY ]ORR IG[YK INGTMKY ZU

# When things go wrong

You don't know what you can't find...



The image shows a resume with a highlighted section. The highlighted section contains the following text:

**Statement**

**OLS: Office Liquidations Solutions May 2010 – May 2013**

**Experience**

**Bialek Healthcare Environments June 2001 – May 2010**

Below the highlighted section, the resume lists the following information:

Bialek Healthcare Environments June 2001 – May 2010  
Design Associate, Client Services Coordinator  
Furniture bid package review, quotation, response and presentation. Small office design, space planning, and equipment selection. Fine and quality for commercial interiors and furnishings.

# What can go wrong

- Catastrophic failures
  - Out of Memory Errors
  - Infinite Hangs
  - Memory Leaks
- Exceptions: Null Pointer, etc.
- Extraction with loss of fidelity
- Missing text/metadata/attachments
- Extra text (eg placeholder, dummy, hidden, internal etc)
- Garbled text

# Did this change help?

- Unit tests can tell you about major failures
- Unit tests only check the things you know about though...
- And only check a small number of files of each type
- There is a much wider variety of files out there than in even a decent-sized test suite
- File type distributions are uneven, a “minor issue” for me might be a “critical issue” for someone else!
- You can check 10 documents by eye, you can't check 10gb of files by eye, let alone 1tb!

# Did this library change help?



- Large scale testing needed to spot these kinds of problems
- Needs automation of running *and* of analysis!
- Flag up key issues, look at those in detail
  
- Exceptions and Errors are easy (1<sup>st</sup> one per file anyway...)
- Metadata little harder, but sizes small
- Attachment counts easy, contents less so
- Finding junk text harder
- Missing text tougher still

[www.quanticate.com](http://www.quanticate.com)



# Did this library change help?



- Need to store output from many runs to compare
  - Common terms per file can identify problems, if it goes from “differences” to “4NGTMKY” then something's wrong
  - Common terms can help with missing text, but not always
  - Token entropy can show garbage text, or data heavy files!
- 
- Too many errors to fix all of them
  - Need metrics to know which to focus on
  - Govdocs1, Common Crawl output, more datasets needed!

[www.quanticate.com](http://www.quanticate.com)

# Apache Tika at Scale

[www.quanticate.com](http://www.quanticate.com)

# Lots of Data is Junk

- At scale, you're going to hit lots of edge cases
- At scale, you're going to come across lots of junk or corrupted documents
- 1% of a lot is still a lot...
- Bound to find files which are unusual or corrupted enough to be mis-identified
- You need to plan for failures!

# Unusual Types

- If you're working on a big data scale, you're bound to come across lots of valid but unusual + unknown files
- You're never going to be able to add support for all of them!
- May be worth adding support for the more common “uncommon” unsupported types
- Which means you'll need to track something about the files you couldn't understand
- If Tika knows the mimetype but has no parser, just log the mimetype
- If mimetype unknown, maybe log first few bytes

# Failure At Scale

- Tika will sometimes mis-identify something, so sometimes the wrong parser will run and object
- Some files will cause parsers or their underlying libraries to do something silly, such as use lots of memory or get into loops with lots to do
- Some files will cause parsers or their underlying libraries to OOM, or infinite loop, or something else bad
- If a file fails once, will probably fail again, so blindly just re-running that task again won't help
- A library upgrade might help, so track which ones failed

# Failure At Scale Continued



- You'll need approaches that plan for failure
- Consider what will happen if a file locks up your JVM, or kills it with an OOM
- Forked Parser may be worth using
- Running a separate Tika Server could be good
- Depending on work needed, could have a smaller pool of Tika Server instances for big data code to call
- Think about failure modes, then think about retries (or not)
- Track common problems, report and fix them!

[www.quanticate.com](http://www.quanticate.com)

# Tika Eval

- Runs Tika against a corpus of documents, or compares the results of running two different versions on those documents
  - Stack traces and exception counts
  - File id, language id
  - Attachment and metadata counts
  - Top 10 most common words
  - Content length
  - Token length statistics, Token entropy
- Gives reports on key things for a human to check

# Tika Batch

- Tools for running Tika across lots of documents, where some of them are going to fail
- Tika Batch App is complete, used by current Eval work
- WIP: Tika Batch Hadoop and Tika Batch Storm
- <http://wiki.apache.org/tika/TikaInHadoop> has lots of pointers on how to handle Tika on Hadoop right now
- Related: <https://github.com/DigitalPebble/behemoth/>



# Other Projects to know about

[www.quanticate.com](http://www.quanticate.com)

# Apache Projects

- Nutch
  - Storm Crawler
  - Any23
  - OpenNLP
  - UIMA
  - CTAKES
- 
- Any from the audience?

# External Projects

- Hardened Tika
- Behemoth, and Behemoth extensions to Tika
- Google Chrome “Compact Language Detector” (CLD)
- Mozilla's LibCharsetDetect
- ICU4J
- Any from the audience?

# Any Questions?

[www.quanticate.com](http://www.quanticate.com)

# Nick Burch @Gagravarr

[www.quanticate.com](http://www.quanticate.com)