

APACHE  
LUCENE  
EUROCON



## Portable Lucene index format

Andrzej Białecki  
ab@lucidimagination.com

Oct 19, 2011

Presented by

**lucid**  
IMAGINATION

*Lucene*

Apache  
**Solr** 

# whoami

- Started using Lucene in 2003 (1.2-dev...)
- Created Luke – the Lucene Index Toolbox
- Apache Nutch, Hadoop, Lucene/Solr committer, Lucene and Nutch PMC member
- Developer at Lucid Imagination

# Agenda

- Data compatibility challenge
- Portable format design & applications
- Implementation details
- Status & future

# Lucene index format

- Binary
  - **Not human-readable\***
    - \* *except for committer-humans* 😊
- Highly optimized
  - **Efficient random I/O access**
- Compressed
  - **Data-specific compression**
- Strict, well-defined
  - **Details specific to a particular version of Lucene**
  - **Both minor and major differences between versions**

```
NRMÿt | yzqq{x ...
```

# Backward-compatibility

- Compatibility of API
  - Planned obsolescence across major releases
  - Relatively easy (@deprecated APIs)
- Compatibility of data
  - Complex and costly
  - Still required for upgrades to an N+1 release
  - Caveats when re-using old data
    - *E.g. incompatible Analyzer-s, sub-optimal field types*
- Forward-compatibility in Lucene?
  - Forget it – and for good reasons

# Backup and archival

- Do we need to keep old data?
  - Typically index is not a **System Of Record**
- But index backups still occasionally needed
  - If costly to build (time / manual effort / CPU / IO ...)
  - As a fallback copy during upgrades
  - To validate claims or to prove compliance
  - To test performance / quality across updates
- Archival data should use a long-term viable format
  - Timespan longer than shelf-life of a software version
  - Keep all tools, platforms and environments?



Ouch...

# M•N

- Lucene currently supports some backward-compatibility
  - Only latest  $N \leq 2$  releases
  - High maintenance and expertise costs
  - Very fragile
- Ever higher cost of broader back-compat
  - $M$  source formats \*  $N$  target formats
  - Forward-compat? Forget it

2.9	3.x	4.x	
+/+	+/-	-/-	2.9
-/+	+/+?	+/-	3.x
-/-	-/+	+/+?	4.x

# Conclusion



If you need to read data from versions other than N or N-1 you will suffer greatly

There should be a better solution for a broader data compatibility ...

# Portable format idea

- Can we come up with one simple, generic, flexible format that fits data from any version?
- Who cares if it's sub-optimal as long as it's readable!



Use a format so simple that it needs no version-specific tools

# Portable format: goals

- Reduced costs of long-term back-compat

- **M \* portable**
- **Implement only one format right**

2.9	3.x	4.x	
++	++	++	P

- Long-term readability

- **Readable with simple tools (text editor, less)**
- **Easy to post-process with simple tools**

- Extensible

- **Easy to add data and metadata on many levels**
- **Easy to add other index parts, or non-index data**

- *Reducible*

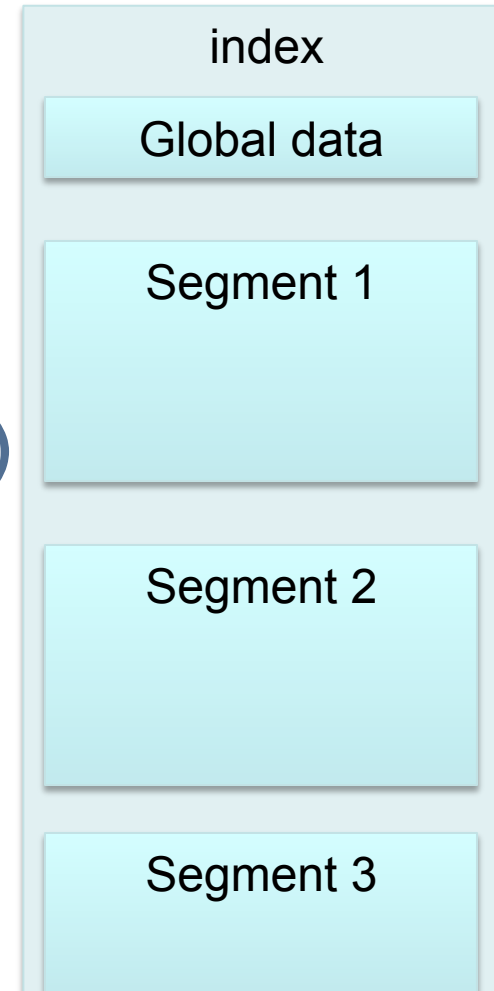
- **Easy to ignore / skip unknown data (forward-compat)**

# Portable format: design

- Focus on function and not form / implementation
  - Follow the logical model of a Lucene index, which is more or less constant across releases
- Relaxed and forgiving
  - Arbitrary number and type of entries
  - Not thwarted by unknown data (ignore / skip)
  - Figure out best defaults if data is missing
- Focus on clarity and not minimal size
  - De-normalize data if it increases portability
- Simple metadata (per item, per section)

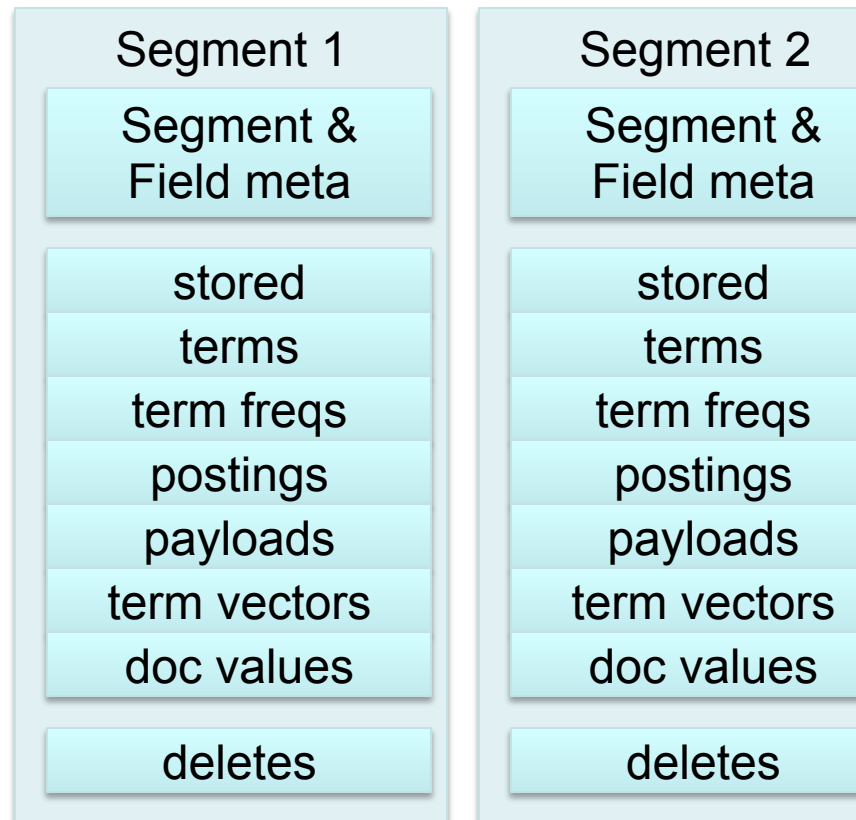
# Index model: sections

- Global index data
  - User data
  - Source format details (informational)
  - List of commit points (and the current commit)
  - List of index sections (segments, other?)
- Segment sections
  - Segment info in metadata
  - List of fields and their types (FieldInfos)
  - Per-segment data



# Per-segment data

- Logical model instead of implementation details
- Per-segment sections
  - Existing known sections
  - Other future sections ...
- This is the standard model!
  - Just be more relaxed
  - And use plain-text formats
- SimpleTextCodec?
  - Not quite (yet?)



# PortableCodec

- Codec API allows to customize on-disk formats
- API is not complete yet ...
  - **Term dict + postings, docvalues, segment infos**
- ... but it's coming: LUCENE-2621
  - **Custom stored fields storage (e.g. NoSQL)**
  - **More efficient term freq. vectors storage**
  - **Eventually will make PortableCodec possible**
    - *Write and read data in portable format*
    - *Reuse / extend SimpleTextCodec*
- In the meantime ... use codec-independent export / import utilities

# Pre-analyzed input

- SOLR-1535 PreAnalyzedField
  - **Allows submitting pre-analyzed documents**
  - **Too limited format**
- “Inverted document” in portable format
  - **Pre-analyzed, pre-inverted (e.g. with term vectors)**
  - **Easy to construct using non-Lucene tools**
- Applications
  - **Integration with external pipelines**
    - *Text analysis, segmentation, stemming, POS-tagging, other NLP – produces pre-analyzed input*
  - **Other ideas**
    - *Transaction log*
    - *Fine-grained shard rebalancing (nano-sharding)*
    - *Rolling updates of a cluster*
    - ...

# Serialization format: goals

- Generic enough:
  - Flexible enough to express all known index parts
- Specific enough
  - Predefined sections common to all versions
  - Well-defined record boundaries (easy to ignore / skip)
  - Well-defined behavior on missing sections / attributes
- Support for arbitrary sections
- Easily parsed, preferably with no external libs
- Only reasonably efficient

# Serialization format: design

- Container - could be a ZIP
  - Simple, well-known, single file, streamable, fast seek, compressible
  - But limited to 4GB (ZIP-64 only in Java 7)
  - Needs un-zipping to process with text utilities
- Or could be just a Lucene Directory
  - That is, a bunch of files with predefined names
    - *You can ZIP them yourself if needed, or TAR, or whatever*
- Simple plain-text formats
  - Line-oriented records, space-separated fields
    - *Or Avro?*
  - Per-section metadata in `java.util.Properties` format

# Implementation

- See LUCENE-3491 for more details
- PortableCodec proof-of-concept
  - Uses extended Codec API in LUCENE-2621
  - Reuses SimpleTextCodec
  - Still some parts are not handled ...
- PortableIndexExporter tool
  - Uses regular Lucene IndexReader API
  - Traverses all data structures for a complete export
  - Mimics the expected data from PortableCodec (to be completed in the future)
- Serialization: plain-text files in Directory

# Example data

- Directory contents

```
120 10 Oct 21:03 _0.docvals
  3 10 Oct 21:03 _0.f1.norms
  3 10 Oct 21:03 _0.f2.norms
152 10 Oct 21:03 _0.fields
1090 10 Oct 21:03 _0.postings
 244 10 Oct 21:03 _0.stored
 457 10 Oct 21:03 _0.terms
1153 10 Oct 21:03 _0.vectors
 701 10 Oct 21:03 commits.1318273420000.meta
 342 10 Oct 21:03 index.meta
1500 10 Oct 21:03 seginfos.meta
```

# Example data

- Index.meta

```
#Mon Oct 17 15:19:35 CEST 2011
#Created with PortableIndexExporter
fieldNames=[f1, f3, f2]
numDeleted=0
numDocs=2
readers=1
readerVersion=1318857574927
```

# Example data

- **Seginfos.meta** (SegmentInfos)

```
version=1318857445991
size=1
files=[_0.nrm, _0.fnm, _0.tvx, _0.docvals, _0.tvd, _0.fdx, _0.tvf
0.name=_0
0.version=4.0
0.files=[_0.nrm, _0.fnm, _0.tvx, _0.docvals, _0.tvd, _0.tvf, _0.f
0.fields=3
0.usesCompound=false
0.field.f1.number=0
0.field.f2.number=1
0.field.f3.number=2
0.field.f1.flags=IdfpVopN-----
0.field.f2.flags=IdfpVopN-Df64
0.field.f3.flags=-----N-Dvin
0.diag.java.vendor=Apple Inc.
0.hasProx=true
0.hasVectors=true
0.docCount=2
0.delCount=0
```

# Example data

- `Commits.1318273420000.meta (IndexCommit)`

```
segment=segments_1
```

```
generation=1
```

```
timestamp=1318857575000
```

```
version=1318857574927
```

```
deleted=false
```

```
optimized=true
```

```
files=[_0.nrm, _0.fnm, _0_1.skp, _0_1.tii, _0.tvd ...
```

# Example data

- `_0.fields` (`FieldInfos` in Luke-like format)

field f1

flags `IdfpVopN-----`

codec `Standard`

field f2

flags `IdfpVopN-Df64`

codec `MockRandom`

field f3

flags `-----N-Dvin`

codec `MockVariableIntBlock`

# Example data

- `_0.stored` (stored fields – LUCENE-2621)

```
doc 2
```

```
field f1
```

```
str this is a test
```

```
field f2
```

```
str another field with test string
```

```
doc 2
```

```
field f1
```

```
str doc two this is a test
```

```
field f2
```

```
str doc two another field with test string
```

# Example data

- `_0.terms` – terms dict (via `SimpleTextCodec`)

```
field f1
  term a
    freq 2
    totalFreq 2
  term doc
    freq 1
    totalFreq 1
  term is
    freq 2
    totalFreq 2
-term test
  freq 2
  totalFreq 2
```

...

```
field f2
  term another
    freq 2
    totalFreq 2
  term doc
    freq 1
    totalFreq 1
  term field
    freq 2
    totalFreq 2
  term string
    freq 2
    totalFreq 2
```

...

# Example data

- `_0.postings` (via `SimpleTextCodec`)

field f1

term a

doc 0

freq 1

pos 2

doc 1

freq 1

pos 4

term doc

doc 1

freq 1

pos 0

...

field f2

term another

doc 0

freq 1

pos 0

doc 1

freq 1

pos 2

term doc

doc 1

freq 1

pos 0

...

# Example data

## ■ `_0.vectors`

```
doc 0
  field f1
    term a
      freq 1
      offs 8-9
      positions 2
    term is
      freq 1
      offs 5-7
      positions 1
  ...
  field f2
    term another
      freq 1
      offs 0-7
      positions 0
```

```
doc 1
  field f1
    term a
      freq 1
      offs 16-17
      positions 4
    term doc
      freq 1
      offs 0-3
      positions 0
    term is
      freq 1
      offs 13-15
      positions 3
  ...
```

# Example data

- `_0.docvals`

```
field f2
  type FLOAT_64
0 0.0
1 0.0010
  docs 2
  end
field f3
  type VAR_INTS
0 0
1 1234567890
  docs 2
  end
END
```

# Current status

- PortableIndexExporter tool working
  - Usable as an archival tool
  - Exports complete index data
    - *Unlike XMLExporter in Luke*
  - No corresponding import tool yet ...
- Codec-based tools incomplete
  - ... because the **Codec API is incomplete**
- More work needed to reach the cross-compat goals
- Help is welcome! 😊

# Summary

- Portability of data is a challenge
  - **Binary back-compat difficult and fragile**
- Long-term archival is a challenge
- Solution: use simple data formats
- Portable index format:
  - **Uses plain text files**
  - **Extensible – back-compat**
  - ***Reducible* – forward-compat**
  - **Enables “pre-analyzed input” pipelines**
  - **PortableCodec**
    - *for now PortableIndexExporter tool*

# QA

- More details: LUCENE-3491  
*<http://issues.apache.org/jira/browse/LUCENE-3491>*
- Contact: [ab@lucidimagination.com](mailto:ab@lucidimagination.com)

## Thank you!