WHAT IS PROSKOMMA?

PROSKOMMA

a **Scripture Runtime Engine** that makes Scripture processing

- fast
- flexible
- simple
- memory-frugal

Key components:

- USFM and USX parsers
- a content model
- succinct storage in working memory
- a GraphQL API (with or without a server!)
- a SAX-like render model

WHAT IS PROSKOMMA?

PROSKOMMA

a project

- created by Mark Howe
- published on github and npmjs under an MIT licence
- financed initially by Unfolding Word and MVH Solutions

a codebase

- written in “vanilla” ES6 Javascript
- about 12k lines of code in the core
- about 2.2k unit tests in the core

a community

- part of Open Component Ecosystem (Discord)

THE CONTENT MODEL

PROSKOMMA

DocSet - collections of documents, (eg a Bible translation)
identified by configurable composite id (eg lang/abbr, org/lang/abbr…)

Document - (eg a book of the Bible)

Sequence - a flow of text

- the canonical content
- an introduction
- a heading
- a footnote…

Block – (eg a paragraph)

Item – what goes inside a block

THE CONTENT MODEL

PROSKOMMA

Items may be

Tokens – printable characters classified by Unicode class into
- word-like
- whitespace
- punctuation

Grafts – links to another sequence:
- at the block level (eg headings, introductions…)
- at the item level (eg footnotes, cross-references…)

Scopes – something that wraps content, corresponding to
- character and word-level markup
- milestones
- chapters, verses...

THE CONTENT MODEL

PROSKOMMA

The content model was originally designed for USFM, but also supports

Tables (with options to filter/sort by row, column, content…)

Trees (tested mainly with CLEAR syntax trees)

Key-Value lookup

SUCCINCT STORAGE

PROSKOMMA

The Curse of XML/JSON Bloat

- documents represented in working memory as trees
- ∴ lots of 64-bit pointers
- ∴ working memory typically 10-30x the size of the serialized document

Succinct vs compressed data

- compressed data typically needs to be uncompressed before use
- succinct data is less compact but can be used in its relatively compact state

Succinct data in Proskomma

- uses JS typed arrays
  - C-style memory blocks
  - byte-level control
  - around 300x faster than standard JS arrays

SUCCINCT STORAGE

PROSKOMMA

Succinct storage tricks

- variable-length integers
- bit-level headers
- optimised for linear search, eg counted strings, record lengths
- content encoded by variable-length enums

So what?

- load and work with multiple, complete translations and sources in a browser
- sub-second serialization load/save of complete translations in native format
- “fast-enough” search etc via block-level indexing

GraphQL API

PROSKOMMA

What is GraphQL?

- a query language developed by Facebook
- a standard implemented for most programming languages
- a solution to under-fetch and over-fetch

Isn’t GraphQL a server technology?

- typically yes, but the FB reference implementation includes no server code
- Proskomma provides a GraphQL interface via method calls
- Proskomma can also support production-ready server GraphQL via Apollo

GraphQL API

PROSKOMMA

Why use GraphQL in Proskomma?

- It provides a way out of the ‘One Right Data Format’ argument by offering
  - Scripture by paragraph
  - Scripture by chapter and verse
  - Scripture chunked by any combination of markup
  - Arbitrary chapter/verse spans
  - “just the text”
  - Tokenised text with in-scope markup
  - …
- It provides strong typing without Typescript
- The schema is self-documenting via the GraphQL endpoint
- A single query can return multiple types of content needed by the UI

STREAMING RENDERING

PROSKOMMA

Why streaming?

- Low memory footprint
- convenient for reports and “document-shaped” output

ProskommaRender (legacy implementation), used for

- Epub generation
- PDF generation (via PagedJS)

PerfRender, used for

- PERF generation from Proskomma
- USFM export
- Arbitrary transforms on Scripture content

STREAMING RENDERING

PROSKOMMA

SofriaRender, used for

- SOFRIA generation from Proskomma
- Rendering within apps (with “wrapped” chapters, verses, phrases…)

PERF/SOFRIA transforms can be combined into **pipelines**

These pipelines may be developed interactively using **Perfidy**