WHAT IS PROSKOMMA?

PROSKOMMA

A **Scripture Runtime Engine** that makes Scripture processing simple, fast, flexible and memory-frugal

Key components:

- a content model for USFM and beyond
- succinct storage in working memory
- USFM/USX import
- JSON representations of the content model including
  - PERF
  - SOFRIA
- 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
- SOFRIA development financed by Faith Comes By Hearing

a codebase

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

a community

- part of Open Component Ecosystem (Discord)

DEPLOYMENT OPTIONS

PROSKOMMA

- in a **Node** command-line script
-
- via “vanilla” **Node Express**
-
- via an **Apollo GraphQL** server
-
- in a **browser** (tested with React, NextJS and Svelte)
-
- in **Android** (tested with Ionic/Cordova and React Native)
-
- in an **Electron app**

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

PROSKOMMA

SUCCINCT STORAGE

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

DATA IMPORTATION

PROSKOMMA

Lexers

- for USFM (regex-based)
- for USX (SAX-based)

Parser/Tidier

- restructures and indexes content

Succinctifier

- Builds enums and succinct documents

Pathways also exist for tabular and tree data...

JSON REPRESENTATIONS

PROSKOMMA

PJMA – **P**roskomma **J**SON **M**odel **A**rchitecture

- Reflects Proskomma model without succinct optimizations
- Supports document, table, tree and key-value lookup content
- Schema variants for two major use cases:

PERF – **P**roskomma **E**ditor-**R**eady **F**ormat

- chapter/verses as empty milestones (so easy to move)
- Separate sequences linked by uid (so easy to update independently)

SOFRIA – **S**cripture **O**bjects **F**or **R**endering **I**n **A**pplications

- chapter/verses as spans within paragraphs (so easy to select content)
- sequences nested within a single object (so easy to render in, eg, HTML)

PROSKOMMA

JSON REPRESENTATIONS

PJMAS – **P**roskomma **J**SON **M**odel **A**rchitecture (**S**uccinct)

- corresponds very closely to Proskomma internals
- ideal for rapid loading and saving of Proskomma state
- one Proskomma docSet per PJMAS document (due to per-docSet enums)

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…)

Identity Transforms for PERF and SOFRIA allow XSLT-style “copy and change” functionality in JSON.

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

These pipelines may be developed interactively using the **Perfidy** application.

SUPPORT FOR EDITORS

PROSKOMMA

Content within Proskomma may be modified using **GraphQL mutations**.

Modifications to succinct data structures are much slower than writes because

- enums need to be maintained
- data is stored in sequential blocks rather than as a tree with pointers

It therefore makes sense to avoid fine-grain (eg per-keystroke) modifications.

Epitelete middleware provides an API for editing PERF via a UI with
schema validation of content
multiple levels of undo/redo
optional stripping/merging of markup not needed by the editor (eg alignment)
support for report generation (eg checks, searches…)

Epitelete-PERF-HTML roundtrips PERF to editor-friendly HTML

PROSKOMMA

RELATED PROJECTS

Epitelete

Middleware for PERF-based editors

Diegesis

A series of PoCs using Ionic (Cordova) and React

Proskomma-React-Hooks

Hooks to provide the most common Proskomma functionality “the React way”

NEXT STEPS

PROSKOMMA

- **Version 1.0 for ETEN summit** (November 2022)

- **Versions 1.X**

  - tighter PERF/SOFRIA schema
  - closer PERF/SOFRIA integration
  - optimisation of GraphQL endpoints for speed and memory usage
  - faster/more flexible editing options
  - pipelines go Turing complete (Project Prostheke)

- **Version 2.0**

  - reworked succinct format
  - formal spec for internals
  - implementations in multiple languages