Emdros demo
BibleTech 2013 Conference

Ulrik Sandborg-Petersen, PhD
Aalborg University, Denmark
Emergence Consult, Denmark
Agenda

• What is Emdros? (Overview, Purpose, Architecture, etc.)

• Demo of an Emdros application (a Bible Study application for the Danish market)

• Technical details of the applications' implementation
What is Emdros?

- **What it is in general:**
  - A text database engine for annotated text

- **What it is also:**
  - A Full Text Search engine
  - A powerful query engine for linguistics data (think original languages research queries)
  - A digital library backend, capable of doing most of the heavy database lifting involved in creating a digital library (such as a Bible study application)
  - And a whole lot more...
What is Emdros?

- Emdros is written in a highly portable subset of C++, with language bindings for C#, Python, Java, Ruby, and Perl.
- Emdros currently runs on:
  - Windows
  - Mac
  - iOS
  - Linux
  - Solaris
  - ... with a pure Java (Android) port under way ...
What is Emdros?

- Emdros is mature:
  - At least 17 years of R&D
  - Two PhD dissertations and one B.Sc./comp.sci. thesis to back it theoretically
  - First line of code written in 1999.
  - Core code has been finished since 2001.
  - Code has been fine-tuned and heavily optimized over many years of R&D
  - Development, maintenance, and optimization is an ongoing effort..
What is Emdros?

- Uses one of five backends for storage:
  - PostgreSQL (for the server)
  - MySQL (for the server)
  - SQLite 3 (for the desktop or iOS device)
  - SQLite 2 (for the desktop)
  - My own proprietary BPT engine (for everything)

- Is able to store a database in one file with zero configuration (BPT, SQLite 2, SQLite 3) for easy data management and deployment to users
What is Emdros?

- The primary advantages of Emdros include:
  - Speed. It scales from the iPod over the desktop to the server.
  - Versatility. It does most of the heavy lifting involved in storing, retrieving and searching textual data, as well as assembling the data as documents for display to the end-user.
  - Ease of use for the database creators (text geeks). The database model underlying Emdros allows text geeks to think about text in a natural way, that fits the worldview of text.
  - Ease of use for the software programmers. A thin API, made powerful through a query language.
On an iPhone 4

- 2.5 seconds to pull out and assemble an entire Old Testament in Danish (~1,300 pages; ~700,000 words) as HTML, with formatting markup.
- 69 milliseconds to pull out and assemble 1 chapter
- 17 milliseconds to pull out and assemble 1 verse
Emdros speed

• ~300 milliseconds to do the same dance for the entire Danish Old Testament

• In a horse-race between a fine, Danish purveyor of iOS Bible study apps, Emdros won the race by being 17 times faster than their solution at pulling out the entire OT, and about 4.5 times faster at pulling out a single verse.
Emdros ease of use

- The lead technical person of Prison Fellowship (Chuck Colson) once told me verbally that he had never in his career seen anything that used so little code to pull out such complicated stuff from a database as is possible by using Emdros.

- Hence, the programmer's job is made quite a bit quicker and therefore cheaper to do.
Emdros philosophy

- Based on pure math. Pure, simple, easy-to-understand math.

- The math permeates the software from bottom to top. That means, the same underlying principles, once grasped, will allow the programmer-user to understand every layer.

- The software stack is not based on complex interrelationships, but simple, clean, thin interfaces between each layer.

- The API is very simple. The reason it is so powerful is because the query language is powerful.

- The API is also consistent.
Emdros legalities

• I am the sole proprietor of every layer of the stack, except the Open Source databases at the bottom.

• Two of the Open Source databases (SQLite 2 and SQLite 3) are Public Domain.

• I am the sole proprietior of the BPT engine.
What is Emdros

App 1

Harvest layer (Retrieval)

App 2

MQL layer (query language)

App 3

EMdF layer (database model)

Storage layer (BPT, SQLite 2, SQLite 3, MySQL, PostgreSQL)
<table>
<thead>
<tr>
<th>Monad:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>w: 1</td>
<td>w: 2</td>
<td>w: 3</td>
<td>w: 4</td>
</tr>
<tr>
<td></td>
<td>surface: The</td>
<td>surface: door</td>
<td>surface: was</td>
<td>surface: blue.</td>
</tr>
<tr>
<td></td>
<td>psp: article</td>
<td>psp: noun</td>
<td>psp: verb</td>
<td>psp: adjective</td>
</tr>
<tr>
<td>phrase</td>
<td>p: 5</td>
<td></td>
<td>p: 6</td>
<td>p: 7</td>
</tr>
<tr>
<td></td>
<td>phr_type: NP</td>
<td></td>
<td>phr_type: VP</td>
<td>phr_type: AP</td>
</tr>
<tr>
<td>clause</td>
<td>c: 8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Demo
Statistics about the demo

• 23,585 lines of C++ code (wxWidgets based):
  – 4,502 lines are for the GUI parts.
  – 11,332 lines incorporate a custom HTML viewer widget
  – 3,882 lines are for generically handling Bible references, morphology codes, and other house-keeping.
  – 1,718 lines handle license keys.
  – Only 2,152 lines are needed for handling the database-side
How to pull this off?

- Use wxWidgets as a cross-platform GUI framework
- Rely on Emdros to do the heavy lifting:
  - Emdros's simple API and powerful query language and pure-math principles allow for consistency and simplicity in all database handling.
  - Emdros's Full Text Search library takes care of the FTS heavy lifting.
  - Emdros's RenderObjects and RenderXML libraries take care of the assembly of content display as XHTML.
Principles

- **Principle #1**: Store, with each database, a mechanism for displaying any part of the database as XHTML. The way to pull out data from the database and assemble it as XHTML thus follows the data.

- **Principle #2**: Store metadata about the database inside the database, thereby enabling application-specific handling of each database, as well as easy deployment.
RenderObjects and RenderXML

1. Emdros Database
2. Render Objects
3. XML or JSON or HTML or whatever
4. Further Processing
5. RenderXML
6. HTML or whatever
7. XML
The RenderObjects and RenderXML libraries rely on “stylesheets” encoded in JSON for knowing how to treat the data for retrieval and document assembly.

- JSON is a well-known, well-understood standard.
- JSON is moderately easy to understand, easy to read for humans, and easy to parse for machines. This mix makes it almost ideal for human-written content such as this.
JSON stylesheets

• The input to the RenderObjects library is:
  – An Emdros database connection
  – An indication of which parts of the database should be retrieved for assembly.

• The input to the RenderXML library is:
  – An XML document, probably (but not necessarily) produced by the RenderObjects library.
JSON stylesheets

- The RenderObjects and RenderXML library both produce a single document.
- The document can be of any nature, so long as it can be encoded by a string of bytes. This includes:
  - XML
  - (X)HTML
  - JSON
  - YAML
  - ... whatever ...
JSON stylesheets

- Each JSON stylesheet tells the RenderObjects library:
  - Which object types to retrieve (e.g., Word, Paragraph, Heading, Verse, Chapter, Book, etc.)
  - Which attributes/features should be retrieved from each object in each object type to be retrieved.
  - What to do and/or emit at the beginning and end of each object.
  - Which post-processing to perform on the document.
RenderObjects stylesheet example

"base" : {
  "object_types" : {

    "chapter" : {
      "docindexfeature" : "xmlnsindex",
      "get" : ["chapter"],
      "start" : "<chapterstart chapter='{ feature 0 }' />",
    },

    "verse" : {
      "docindexfeature" : "xmlnsindex",
      "get" : ["osisID"],
      "start" : "<versestart osisID='{ feature 0 }' />",
    }
  }
}
RenderObjects stylesheet example

"token" : {
   "docindexfeature" : "xmlindex",
   "get" : ["wholesurface", "parsingtag", "strongs"],
   "start" : "<t \n   surface='\{ featurenomangle 0 }'\n   parsingtag='\{ featurenomangle 1 }'\n   strongs='\{ featurenomangle 2 }' />",
},

"postprocess" : [
   { "prefix" : "<tisch>" },
   { "suffix" : "</tisch>" },
   { "prepend_XML_declaration" : true }
]
Principles of RenderObjects and RenderXML

- Be declarative: Let the stylesheet author say what he/she want to retrieve, and what to do with it.
- Be powerful: Support complex document-assembly processes through a powerful template language.
- Be simple: Don't require the use of advanced features if not necessary for the user.
- Be concise: Don't be verbose. It's a human who has to write this stuff (stylesheets).
- Be benign: What you don't know about the language won't hurt you.
Principles of RenderObjects and RenderXML

- Use design patterns: The template language follows certain design patterns across the various template tags, thus making the learning curve less steep.

- Have no impedance mismatch between data and display: Let the stylesheet structure and the structure of the data closely match, so that, if you understand one, you can understand the other, and vice versa.

- Support stylesheet reuse: The RenderXML library supports inheritance between stylesheets.

- Be embeddable with the data: The JSON stylesheet can (and probably should) follow the data which it is able to display, right in the database.
Questions?

ulrikp@emdros.org
http://emdros.org