SourceToHtml – A Sample Design Document

Jim Fawcett – MVCC Code-a-thon, 14 Nov 2020

This is a sample to help you draft your own design for the TextFinder application – see student handout for a description of that.

Project Specification:

1. Description:

SourceToHtml transforms source code to a web page, i.e., HTML decorated with CSS styles.

2. Program inputs:

Inputs are provided on the command line and take the form:

```
SourceToHtml /P [path to file] /s GrayStyle /n [true|false]
```

where:

a. /P defines one or more fully qualified file names
b. /s selects an existing style
c. /n indicates whether line numbers are prefixed to each line

3. Program outputs:

Output, when applied to `myfile.h` and `myfile.cpp`, are files `myfile.h.html` and `myfile.cpp.html`

Design Description:

1. Uses:

The program will generate outputs for communication between developers who may be remotely located. Typical uses:

a. Support for developers working on a large project that need to understand something about the code they will call or be called by.
b. Support for design review meetings.
c. Instructors use in their design and implementation courses.

Having web page displays allows a designer to think about another's code without downloading or forking. They will often do that, but code published on the web makes information transfer quicker and easier.

2. Program tasks:

The SourceToHtml program will need to execute the following tasks:

a. Accept arguments from the command line, parse them, and store for use during construction of the output.
b. Transcribe each source file, replacing markup characters with escape sequences, e.g., < transcribes to < and prepending transcribed code with <pre> and postpending </pre>. Eliminating markup avoids confusing browser page parsing and the use of pre tags instructs browsers to honor source whitespace layout.
c. Embed transcribed source in an HTML document with header and body.
d. Apply page header and footer regions using appropriate markup from a subdirectory.
e. Apply theme styles by linking to a local style sheet.

3. Program structure:

The diagram below shows a package structure for SourceToHtml processing. Packages consist of one or more files of the source code, e.g., Embedder.h and Embedder.cpp. These packages are derived from program tasks, above, as follows:

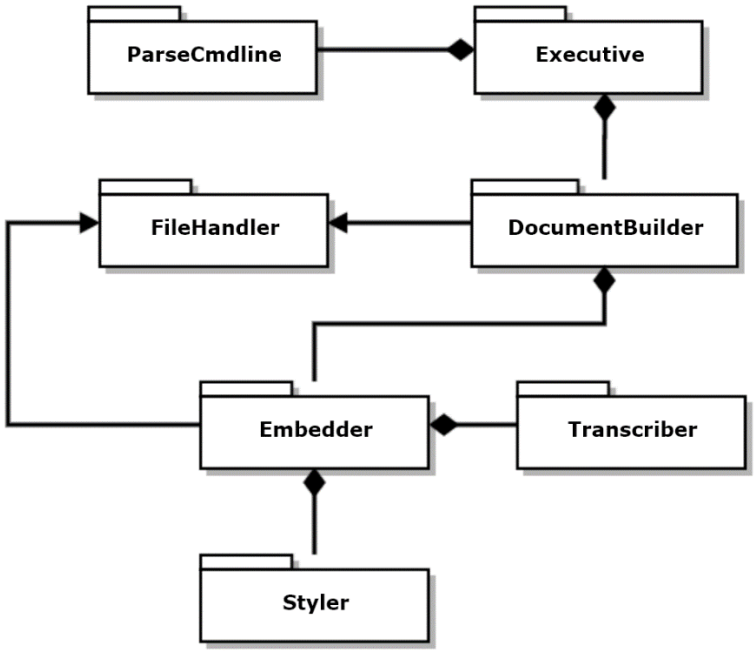

Figure 1. – SourceToHtml Package Diagram

a. **Executive** is responsible for collecting command line arguments, argument error handling, and passing each filespec, style, and lineno command to **DocumentBuilder.**
b. **DocumentBuilder** accepts those arguments and builds a document as text, using the services of **FileHandler**, **Embedder, Transcriber, and Styler**. It passes that to **FileHandler** for writing to a file with html extension.

c. **FileHandler** opens a file stream from the source file and passes a reference to the stream back to **DocumentBuilder.** It writes the document text to a file with html externsion. **FileHandler** also reads document header and footer markup.
d. **Transcriber** converts HTML markup characters in the source text into escape sequences to avoid confusing browser document parsing.
e. **Embedder** embeds the source in <pre> tags and wraps that with markup for header and footer, as provided by **FileHandler**. It prepends header markup and postpends footer markup, and finally embeds the result in a document with head and body sections and a stylesheet link in the head section.
f. **Styler** builds a style link, referring to a styles file and passes that back to embedder. For the current design that is simple enough to move into embedder itself. However, future versions will provide visibility controls for comments and function bodies to give users views at different levels of abstraction. That is much more complex, involving some parsing of the code text. For that reason, it has been factored into a separate part.

4. Processing:

Transcription is a fairly simple process, simply substituting escape sequences for the markup characters: <, >, &, ‘, “. For reasons of performance, the design should transcribe the input stream on its way from file to string. That avoids the need to insert character sequences into an existing string, which is an order NxM process, as each insertion would require moving all characters after the insertion points to make space for the escape sequences.

Embedding is a process of ordered assembly of the document:

a. Head markup, including title and style link
b. Header block, read from a file managed by **FileHandler**
c. Transcribed source with optional line numbers
d. Footer block, also read from file managed by **FileHandler**

The assembly can be executed efficiently by streaming each part into a destination file reference provided by **FileHandler**.

Should performance become an issue, for example publishing code for an entire large project, it would be fairly simple to execute individual page transformations in parallel. Essentially, main would send individual filespecs to a threadpool, where each thread invokes **DocumentBuilder**.

5. Error Handling

There are two primary sources of error for Src_To_HTML, parsing the command line and opening files. These need to be handled differently:

a. Command line parsing:
   Parsing errors should result in the display of a help message illustrating how to correctly supply arguments, then gracefully shut down.
b. File opening errors:
   These may be due to problems with accuracy of the filespec or file permissions. In case of an opening error, the program should emit a one-line error message and continue to the next file if there is one.

6. Testing:

Testing starts with the creation of several directories containing different sorts of files.

a. Directory with no files
b. Directory with files that are not source code, including binary files. The program is not expected to process these files correctly, but the test verifies that program operations continue.
c. Directory with several source files, perhaps for different programming languages
d. Directory with correct results of converting files in c. These can be initially generated by carefully examining results from test c and editing where necessary.

Good testing practice would create these directories with their contents from a script, so that every act of testing would start from the same environment, and so, be repeatable.

Execution of tests are implemented using scripts with different combinations of command line arguments.

7. Future Extensions:

Three extensions are proposed:

a. Create a Graphical User Interface for selecting files and providing options with checkboxes. This does not need to be integrated with the Src_To_HTML program. It can simply create an instance of this program with arguments supplied by the user. It does not need to be implemented in the same language used for Src_To_HTML. It simply needs to use language facilities to spawn the executable and provide it with command line arguments.
b. Parallel execution of page transformations. For languages that have available threadpools (provided by language libraries like Rust, or available from open source like those provided by https://jimfawcett.github.io/ThreadPool.html, this task is quite straight-forward.
c. Implement altering the display of function bodies and/or comments from display:block to display:none with CSS styling. This supports high level views of code and eliminates comments for developers who use the code a lot and don't need them. This is fairly complex processing, using JavaScript and CSS, and parsing of source code to find the appropriate blocks to show or hide.

The first and second would entail about a person week of implementation and testing for each by an experienced developer (including experience with threading). The third is more involved and could take three or four person-weeks to fully implement and test.

8. Summary and Conclusions:

The design, as discussed here, can be implemented with very little risk in about one person week, including testing and deployment. It should prove to be useful for the intended user groups, e.g., developers and instructors.