Jan Gundtofte-Bruun
jan@g-b.dk
2639 7565

WHAT did I do?

The RFA project was an intranet site presenting hardware spec sheets for many thousands of products, each spec sheet existing in a number of revisions. There was some highlighting of changes and additions between revisions.

My task was to implement highlighting for deletions as well, and also to implement “view modes” so that in addition to showing the spec sheets with highlighting, it should be possible to turn off highlighting, or show only the changed passages.

I ended up rewriting a lot of the XML parsing and all of the presentation code. Also, I implemented XML-aware revision comparison which was no mean feat.

WHY was it necessary?

Basically, it turned out that the existing highlighting was marked up by the editors manually, and quite poorly. Not all edits were marked up, and removed passages were simply deleted without any mark-up at all. In order to identify those passages, some form of comparison of sequent files was needed.

Also, the file format was not really XML but a misunderstood approximation. To make matters worse, the in-browser presentation was extremely slow, letting users stare at a blank browser page for a long time (up to two minutes for large files), while the page was being rendered by parsing the file line-by-line and constructing appropriate HTML along the way. This really needed to be addressed.

HOW did I do it?

I decided to effectively ignore the existing but incomplete change markup and instead perform comparisons between sequent revisions. This would yield the data necessary to exhaustively highlight the actual additions and changes as well as the desired deletions.

Furthermore, the files didn't start out as XML, they were pulled via FTP from a mainframe in an arcane file format and were converted to a kind of pseudo-XML. I set forth to improve the converter so it would generate clean, actual XML. This was no easy task because there were literally no specifications for the legacy file format, and the sole maintainer of the existing parser had died. An attempt to glean empirical rules from the source files and code proved futile, so I ended up black-boxing the converter and pipelining a new module to translate the pseudo-XML output into proper XML.

The hardest part was to write the code to identify sequent files and annotate them with revision marks. It's too long to explain here, but my algorithm, the mark-up, and much of my thought process are described in this post at StackOverflow.

Finally, I created XSLT and CSS templates so the system could present the XML files as pretty formatted HTML in no time at all.

Now, the various view modes are implemented with a small set of short CSS templates going through the same XSL transformation. Easy!

The result: With fewer overall lines of code, the current version offers proper and XML-aware diff, faster performance, and multiple view modes which can be extended very easily.

WHAT's so cool about that?

This was my first foray into pure Java, and I dare say I not only achieved the set goals but surpassed them, and handed back a project in considerably better shape than it had been given to me.

The points I feel most proud about are:

- Making sense of 26kloc of utterly undocumented code.
- Implementing a revision comparison function that is aware of XML structure and stores its findings as attributes to the relevant elements in the newer XML document.
- Replacing a slow, complicated HTML rendering system with an XSLT solution has made it more responsive, much simpler, and easily extensible. A win for the users, and for the maintainer.
- Improving the data format into proper XML. Data needs to be stored in (open) standard formats, period.