Review of API Docs Auto-generation Systems

Systems reviewed

- Natural Docs
- JsDoc Toolkit (supersedes JSDoc)
- jGrouseDoc
- ScriptDoc
- Markdown

General Comments

- Markdown is a general text-to-HTML documentation generator, based on wiki-markup-like principles. It is not specific to API documentation at all. It might be something to consider for Tutorials, though.
- ScriptDoc is a standardized format for code comments, but as far as I can tell, there is no tool available for parsing those comments and generating HTML docs.
- The remaining three (Natural Docs, JsDoc Toolkit and jGrouseDoc) are designed for API docs, and generally work by parsing the comments in the code. The remainder of my comments will focus on these three systems.

Criteria

Responses to Colin's suggested criteria:

Easy to maintain and update

- All three systems require comments to be added to the code for anything that is to be in the API docs.
- The comment format for Natural Docs is, well, natural, but very picky with respect to whitespace
- The comment format for JsDoc Toolkit and jGrouseDoc is very javadoc-like, so less natural, but very familiar for anyone who's worked with javadocs before.
- Because these systems are comment-based, anyone modifying a function signature will naturally be in the right place to update the associated documentation (but must remember to do so).

Current and up to date

- All three systems could be invoked as part of our hourly/nightly build process and deployed on the build server.
- At release time, a snapshot of the docs could be taken and published at a stable (i.e. non-trunk) url.

Support users who don't adopt the latest release right away

- None of the three systems have any form of built-in versioning of documentation.
- All three systems generate HTML output, which could easily be (manually) archived for previous revisions, if we wanted to do this.

Foster contribution to the documentation

- All three systems are based on comments in the code, and so all three are equally weak in this area:
    - The general community cannot easily fix or add to documentation.

Other Comments

Full Language Support

- jGrouseDoc does read and use the function signature that is immediately following a comment header - but if the comment header is not present, the function signature is ignored.
- JsDoc Toolkit will parse functions in a namespace if the namespace has an `@namespace` tag defining it. This might produce docs organized by component, if judiciously used. Functions on a `that` object will not be properly parsed unless the `that` object is declared to be a namespace, which produces odd-looking navigation, and even then, might not work.

The implications for this are:

- If we want API docs for something, we must write the proper comments for it. If it's not properly commented, it won't be in the docs.

Comment Structure

- JsDoc Toolkit and jGrouseDoc are very javadoc-like. The format is almost identical, so developers familiar with javadoc comments will have no trouble commenting their code for one of these systems.
- Natural Docs uses a much more natural looking format, which is much easier for humans to read, but it is actually rather picky: Whitespace is very important, for example, and thus the comments will "take up a lot of space" in the file.

Output

- Natural Docs produces output that is file-based. That is, the main navigation lists the files, and the docs for each file lists the functions in that file, in order of their appearance in the file.
- JsDoc Toolkit produces output by class, and in each function description, mentions which file it is found in.
- jGrouseDoc produces output that is organized by files, as well as identifying global functions.

Quirks

- JsDoc Toolkit gives numerous warnings of the form "WARNING: Trying to document applySkin as a member of undocumented symbol that." I tried ensuring that the `that` object had a comment, but that didn't seem to address the issue. No documentation was produced for the function, nor anything else in that file.

Namespacing

- All of these things seem to consider everything we do to be in the global namespace.
- JsDoc Toolkit has a `@namespace` tag, but it is not terribly useful. Multiple occurrences in different files are all percieved to be in the same file, and don't affect the description of the functions in that file - they're

General Impressions

I tried generating the docs for our code using all three of these systems, starting with no modifications to the comments, then experimenting with adding comments to some functions.

First thought: I suspect that our judicious care in namespacing, scoping, etc. confuses these systems.

- JsDoc Toolkit chose an odd selection of functions to document, and ignored the comments I deliberately added.