USENIX Association

Proceedings of the 17th Large Installation Systems Administration Conference

San Diego, CA, USA
October 26–31, 2003

Designing, Developing, and Implementing a Document Repository

Joshua Simon – Consultant
Liza Weissler – METI

ABSTRACT

Our company had grown large enough and complex enough to require a centralized repository for its documentation. Various internal groups produced all sorts of documentation; regions and districts produced documentation for and about clients, including proposals for work and the results of work performed. Often the documentation produced did not take advantage of previously-produced work.

The Repository project intended to centralize all documentation in a single ''one-stop shop'' for creating, updating, storing, and searching documents, to provide various information about every document so documentation authors and information gatherers could search for and use or reuse existing similar documentation as appropriate. It also was intended to be minimally intrusive for both the authors and the users of the documentation. This allowed the company to enforce a common look-and-feel for all documents within a certain type and for consistency in content as well. Other goals included the ability to update content in exactly one location and have that change propagated throughout all relevant documentation.

The user interface had to be simple enough for nontechnical people (managers, sales staff, administrative assistants, and so on) to be able to use it to add, edit, and search for documents meeting their variable criteria, and yet complex enough to be meaningful in terms of the results. Because of the high costs of solving this problem correctly, we decided to take a low-key internal approach to the project instead.

Introduction

Problem

While working as consultants, our former company had grown large enough and complex enough to require a centralized repository for its documentation. Various internal groups, such as Finance, Information Technology, Legal, Marketing, Recruiting, and Sales, produced documentation. Regions and districts produced documentation for and about clients, including proposals for work and the results of work performed. Often the documentation produced did not take advantage of previously-produced work. The Repository project intended to centralize all documentation in a single ''one-stop shop'' and provide various information about every document so documentation authors and information gatherers could search for and use or reuse existing similar documentation as appropriate.

There's an old adage about the shoemaker's children never having good shoes. Similarly, even though we were a company of system administrators, we often needed to design our own tools. Most everyone in the company recognized the need for such a repository but nobody had the cycles to design or implement one. Josh was moved into a corporate-side role to work on this project, given his extensive history of documentation and project management work, as well as the demonstrated ability to translate between technical and nontechnical staff. Josh asked Liza for her assistance on the database and Perl side, since she was a member of the tool development team and already familiar with the internal IT department policies and because she has a background in library sciences.

Because of the nature of the business and her own responsibilities, Liza could only work on the project during her off-hours. Similarly, we were constrained from spending any real money on hardware or software; even though most technical people and many nontechnical people wanted a repository, nothing was budgeted to pay for it.

Another constraint was the term *documentation repository* having different meanings for different people. The term could mean an electronic library like the physical one at Alexandria, or a vault containing copies of the physical documentation we'd produced. But defining *documentation* was itself a hard problem. Certainly any physical document we'd deliver to a customer counted – such as summaries, analyses, recommendations, project plans, and so on. Similarly, the pre-project work, such as statements of work, service level agreements, and other contracts count as documentation. But what about the online question-and-answer database? What about marketing materials and sales literature? What about the HR-specific forms (such as the IRS I-9 and W-4, and the annual reviews of your employees and management)? As we looked

further into the back office within the company we found that more and more document types could and did exist. Our repository had to allow for those document types. This led to our categorizing system.

While we could have just added disk space onto a centrally-located file server, we had many different people creating and editing documentation, many of whom used different operating systems and different tools for producing or updating the documentation. Furthermore, different people use different naming conventions and different organization or classification schemes, so finding something without a search engine of some kind would be exceedingly difficult.

Goals

The major goals of the project were to provide a single place on the company network for all documentation to be created, updated, stored, and searched, while being minimally intrusive for both the authors and the users of the documentation. This should have allowed the company to enforce a common look-and-feel for all documents within a certain type and for consistency in content as well. Minor goals included the ability to update content in exactly one location and have that change propagated throughout all relevant documentation.

Requirements

We spent a substantial amount of time identifying the requirements for such a documentation management package. These included:

- the ability for anyone to add documents to the repository
- the ability for document owners to edit the documents
- access control for who could read or make changes to documents; for example, only the specific employee, management, and Human Resources should have access to financial information
- strict restrictions on deleting documents from the repository; in general, we were going to forbid the delete function to all except specific ''librarians'' whose role included database content management
- keyword-based searching
- storing meta-data and a URL instead of the entire document
- no constraints on how people create, edit, or maintain the documents themselves; revision control is a solved problem we didn't want to deal with

We then began researching the vendors in the space and the costs involved. Because of the high costs of solving this problem correctly, we decided to take a low-key internal approach to the project instead.

The interface itself had to be simple enough for nontechnical people (managers, sales staff, administrative assistants, and so on) to be able to use it to add, edit, and search for documents meeting their variable criteria, and yet complex enough to be meaningful in terms of the results.

Being system and database administrators really made this project possible. As system administrators we'd had previous experience with talking intelligently to vendors' sales personnel. We're skilled at converting what business (usually nontechnical) people want into and out of technical terminology. It's typical, in our experience, for system administrators to fill programming needs when there are no other ''official'' programmer resources available for urgent project work. Having a database-savvy member of the team saved us a lot of unnecessary work as well.

Design & Development

Based on the requirements analysis and the reduced necessary functionality, we decided to track the following information:

Author: The authors of the document; we found that being able to search for documents by author was useful since people would remember ''Alan or Betty wrote . . .''

Title: The document title

Description: A short description about the document, which would show up on search results

Revision: The revision number or string, such as ''1.0'' or ''First Edition,'' to allow for tracking multiple copies of a document over time

Date: The date the document was published or released (which could theoretically be in the future)

URL: to the document, including the protocol specification

Permissions: The permissions and state information: whether the document has been approved for release or was a draft, whether changes can be made to the document, whether the author has checked it out for revisions, and whether public (meaning unauthenticated) users could access the document

Search flags: Company-internal codes for what major and ''Networking'' and ''Security''

Class: The document class, one of client/technical, internal documentation, marketing/sales literature, recruiting, or other; these five categories were a direct result of our analysis of defining a document

Subclass: The document subclass, a well-understood company-internal code to further classify the document, different for each class

Keywords: A list of keywords to be used in searches; for cross-application consistency we used the same keywords for the repository as we did for the question-and-answer database

A sample record, using this paper for content, would be the following:

Author: Simon, Joshua; Weissler, Liza
Title: Designing, Developing, and Implementing a Documentation Repository
Description: A LISA 2003 paper on how we designed, developed, and implemented a general-purpose documentation repository
Revision: 1.0
Date: October 26, 2003
URL: http://www.usenix.org/publications/library/proceedings/lisa03/tech/simon.html
Permissions: Approved, Public [structure of bits: approved 1, can-change 0, checkedout 0, public 1]
Search flags: null
Class: I (internal)
Subclass: D (documentation)
Keywords: doc, repo

We had certain design elements determined from a recent documentation styles analysis project that defined fonts, relative sizes of headers and text, use of color, and so on, which made some elements of the user interface design fairly straight-forward.

The top-level or home page contains a list of the nine major topics:

- **Document-specific:**
  - Adding a new document record to the Repository
  - Editing an existing document record in the Repository
  - Uploading a new document to the Repository
- **Searching:**
  - Browse the entire Repository for document records
  - Searching for a document
  - Jump to the Templates page for the latest document templates
- **Repository documentation:**
  - User's Guide and Administrator's Guide
  - On-line Help
  - Get statistics on documents in the Repository

Each subsidiary page would contain navigation links in the header (including a link back to the Repository home page, a link back to the intranet home page, and links to related functionality in other applications on the web), the page content (such as data entry form, results, or error message), the application revision number, and the standard company footer. The standard footer provided contact email, a link to the company's internal bug reporting application, and the copyright statement.

Development of the application took place over a two-month period (February 7 through April 5). The application was written in Perl (5.005) using the DBI and DBD::Oracle modules, with Oracle 8.1.6 for the back-end database. Development and quality assurance testing were on systems running Red Hat Linux 6.2, while the production system was running Solaris 2.6.

Staffing of the development effort was basically one person in her spare time, so it may be said that the development budget was essentially zero (which yields quite a good return on investment). Our company had a long history of infrastructure building as volunteer effort on the part of employees; this helped the company keep costs down, and gave employees the sense of having more of a stake in the company. Of course, it also had some influence on the distribution of yearly bonuses. More critical applications were the "day job" of a team of virtual developers, but as this application fell a little lower on the priority list, it remained a volunteer effort.

We don't have hard and fast data on the number of person-hours spent on this project. As noted in the "Acknowledgements" section, 58 members of the company worked on this project either directly (design, development, and testing) or indirectly (related development, development of modules we could adapt, etc.). Josh spent probably four months working on this project, including the detailed requirements analysis, and Liza spent about one month on it.

The application itself was fairly quick to code due to a number of existing Perl modules in use at the company to standardize database access, authentication, error reporting, and so forth. The programmer herself was one of the company's "virtual developers" and thus was ramped up on the company's application design philosophy and structure. (Had this effort been a real job and not a volunteer effort, it likely could have been developed in well under half the time.)

The documentation that goes along with the code – both a User's Guide for the general user (extensive online help) and an Administrator's Guide for the librarians – was written alongside the application. Having a standardized cascading stylesheet for both the application and its documentation helped maintain the common look-and-feel across both. The documentation development time is included in the 6-months of application design and development time.

Concurrent with the application and database development, we collected bibliographic data on existing company-wide web-based documentation so we could force-load the Repository with valid data. This was an interesting process in itself, as the data could be almost anywhere. A simple conversion script parsed the bibliographic data, performing error checking and loading the data into the database.

Our system administration skills were essential to the success of this project. We analyzed the problem, which (as with many projects in technology)

seemed to get bigger and more complicated the more we looked at it. We identified our requirements so we could determine what we wanted as well as what we didn't want. We used organizational skills to figure out how best to allocate our volunteer resources to best effect. We documented the application, both in the code and in standalone User's and Administrator's Guides. Our familiarity with database schema design, library sciences, scripting, and best practices in software development and testing as part of the release cycle was absolutely essential in producing a usable, functional application.

Testing

The testing process took place in three major steps: alpha testing, quality assurance testing, and beta testing. Our rationale for this was four-fold. First, we wanted the people who would become the content consistency police, or the librarians as they were called, to be intimately familiar with the processes involved. Second, we wanted to make sure, before releasing the application to the whole company (including both technical and nontechnical staff), that both technical and nontechnical people could use it. Third, the IT department had a dedicated test group (off-hours volunteers) who would do our QA testing for us. And fourth and finally, it's good software development practice to do three-stage testing (alpha, QA, and beta).

The company had three full-time technical writing staff. We used them as our alpha test team, gave them full access to the database, and asked them to test everything: adding, editing, and deleting documents. Since they were using a test database (which would be purged and reloaded with real data before production), we let them know deleting records was okay for the alpha test, though we asked them not to do so with ''real'' documents once we'd gone live. During the two-week alpha process we answered four email questions and reported a total of 26 bugs or feature requests, most of which were fixed before the QA test period began.

Immediately after the alpha period we went to a Quality Assurance (QA) period. QA was handled internally by a team of volunteers called "Bugzappers." The process required members of the team to say whether they'll participate in a release, and if so, to review the application specifications and documentation, test features, report bugs, make feature requests, confirm whether bugs and feature requests purported to have been handled actually were, and finally, give a yea or nay as to whether the application can be released. We had the participation of about half a dozen Bugzappers who picked nits with the application during the QA period. During the QA period our test team identified 13 new bugs, all of which were fixed before beta. At the end of the QA period, only six bugs or feature requests remained open.

To perform a wider test we released a beta version of the application and database to the documentation-focused experts within the company. This included the alpha and QA test teams, as well as the Publications committee, who oversaw and mentored all technical writing within the company; the Methodologies Development team, who designed, developed, and documented detailed methodologies; and the Repository group, who were responsible for any and all thoughts on documentation and tool repositories company-wide. During the week-long beta test the team identified 21 new bugs or feature requests, most of which were fixed before production. At the end of the beta test period, only 17 bugs and feature requests remained open.

Table 1 shows the bugs, feature enhancement requests, and user training issues during the testing process.

Type	During Dev & QA	During Beta	Entire Process
Bugs	30	6	36
Enhancement requests	7	7	14
User training issues	6	3	9
Total	**43**	**16**	**59**

Table 1: Bug status.

Production Implementation

Once the beta testing was complete, we put the application into production. Doing so was as simple as announcing to the company as a whole that the application existed and putting a link to it on the main intranet home page.

Within the first month in production, users identified some 15 additional bugs and feature enhancement requests, all of which were later implemented, such as tying the documentation more closely to the application; cleaning up the user interface; and providing automated email notifications to the librarians when records were added, edited, or deleted to ensure sanity in the database.

A month after the initial release (called 2.0 for historical reasons) we released a patched version (2.1) that fixed four of these issues. Additional patch and incremental hot-fix releases over the next four months fixed or implemented an additional four of 13 bugs or requests. Table 2 shows the time line of the testing and release cycles.

Measuring Success

It is difficult to measure the success of this project in a quantitative manner. Other than the number of documents, the number of users, and the

number of searches, there is no real quantitative measurement of success. These measurements are discussed, along with qualitative measurements, in the following sections.

Version	Date	Description
x2.0.0.1	Apr 5	Alpha test
x2.0.0.2	Apr 13	Quality Assurance
x2.0.0.3	Apr 23	Beta test
x2.0.0.4	Apr 27	Beta test, continued
v2.0	May 7	Initial production release
v2.1a	Jun 9	Rearranged top-level page, added Upload and Templates links, added version to footer
v2.1b	Sep 7	Corrected typos and formatting
v2.1c	Oct 4	Cosmetic style changes for application & documentation
v2.1d	Oct 15	Bug fix
v2.1e	Oct 19	Bug fix

Table 2: Incremental release time line.

Number of documents

The number of documents – or in reality, the number of unique document URLs – in the database grew over time. The initial bibliography collection phase, in fact, allowed us to begin production with nearly 800 documents. Table 3 and Figure 1 show the number of documents over time.

The continued growth of documents in the Repository after going into production, not all of which were added by the librarians as an extension of the bibliography collection process, showed us that the Repository was indeed useful. This met our expectations.

Date	Event	Documents
Feb 24	Original coding	3
Feb 28	Bibliography collection phase 1	346
Mar 15	Bibliography collection phase 2	567
Apr 15	Bibliography collection phase 3	752
May 7	Initial release	794
Jul 17	Revisions announced	897
Oct 23	One author left the company	1101
Dec 4	Other author left the company	1128
Dec 18	Insider source reported statistics	1130

Table 3: Documents over time.

Number of Users

The number of users also grew over time. This met our expectations, as we continually grew the number of possible users, as shown in Table 4.

Phase	Users
Development	2
Alpha test	5
Quality Assurance	11
Beta test	34
Production	> 250

Table 4: Number of users over time.

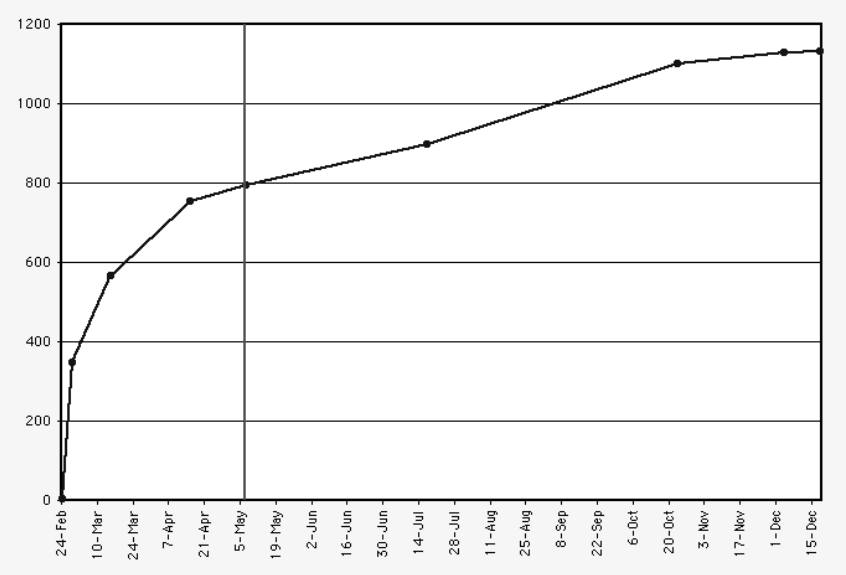

Figure 1: Documents in respository over time.

Between the start of the beta and the start of production, the company experienced several rounds of layoffs, reducing the maximum possible audience from over 400 to around 250. That maximum number continued to decrease during the production period down towards 150-200. This curtailed the growth pattern in usage and also reduced our ability to research and analyze the data further.

Number of Searches

We do not have data on the number of searches performed, as the httpd access_log files have not been made available. Both of the authors of this paper left the company through layoffs, and the company has not produced the old logs for this paper.

However, even were this information be made available it would be difficult if not impossible to draw any meaningful conclusions from it. Searches during the development and testing phases are meaningless, since they are intended to test both successful and unsuccessful searches. Searches made during production might be meaningful if we tracked success (defined as the search resulting in either at least one document or the document(s) the user wanted or needed), but we did not do so early enough to collect the data. Finally, even if we knew how many searches were performed (regardless of success), the large reduction in the user base makes most of the numbers meaningless.

Qualitative Measurements

It is also difficult to measure success qualitatively. Without access to the log files, and without a survey as to whether the users were satisfied with their use of the Repository, there is no real qualitative measurement that was performed. Feedback was generally positive, with bugs and enhancement requests submitted by several users; several members of the company, in various roles (such as technical, sales, marketing, and executive management), mailed to thank the development team for their efforts and praising the Repository.

Related Work

There were several intranet document management systems when we first started looking at vendors (over two years ago as of the date of first publication of this paper). Based in part on a document management seminar one of the technical writers attended, there were 37 software packages on our ‘‘long’’ list, including XML development packages. We realized before we could narrow down to a ‘‘short’’ list that we could not afford any of the packages, though the most likely candidates included Documentum, Enigma’s DynaText, IntraNet Solutions’ Intra.doc!, QUIQ’s Knowledge Network, and Vignette’s content management system.

We also briefly considered using wikis for this project. A wiki is a composition system-meets-discussion medium tool used for collaboration. While wikis are great for groupware or collaborative work on documentation development, we weren’t developing documentation in this project. We were developing a repository to store and track documentation explicitly developed elsewhere. A wiki was the wrong tool for what we wanted.

Future Work

Future work includes fixing bugs identified during the continuing use of the Repository software as well as adding the additional features omitted from the current release. In no particular order, these include:

- Separating the one big index.cgi script into smaller, more-manageable chunks, say to have a search.cgi instead of index.cgi?action=search. The original intent was to have much of the guts of what the various actions do are collected into subroutines in a separate application Perl module, keeping index.cgi relatively lean and mean, but as the application grows it makes sense to break it apart.
- Re-think the company and non-company author handling, or at least automate record updates when an employee leaves the company.
- Provide a record-level authorization permission structure.
- Provide and track type-dependent information (e.g., publisher for books and magazines, magazine title and issue/volume/number for articles and columns, etc.)
- Include command-line interfaces to create new documents & templates, update existing documents & templates, search for documents & templates, administer the database, and convert a document from one format to another.
- Provide a file system-like interface for browsing.
- Provide tools to convert between a consistent back-end format (such as XML) and the desired front-end formats (HTML, PDF, RTF, TXT, etc.); for example, xml2html and xml2pdf and pdf2xml and html2xml.
- Provide a tool to check out (and lock) a document for revision.
- Provide an interface to the public Internet web site (so non-authenticated personnel can search for PUBLIC documents).
- Provide an interface for people to suggest changes to documents they do not own.
- Provide and track document-level security so users cannot see document records (including the URLs) for documents the web may not let them access.
- Provide a way to ‘‘group’’ documents.
- Provide a way to ‘‘group’’ permissions.
- Provide a better mechanism for documents to include or supersede each other. Updating data in one place should update it everywhere.

- Update the document record automatically if the content of a certain URL changes.
- Generate and track document numbers.
- Provide a means for a document to be broken off from a reference so changes to that reference no longer are applied to the document.
- Provide a custom query interface (build-your-own SQL).
- Provide a way to save and load complex queries.
- Enforce documentation process flow (creation, modification, deletion, permissions, change control, state changes, and so on).

Conclusions

Our major goal of providing a single all-inclusive starting point for searches and a repository for uploading documents was met. It was minimally-intrusive for both authors and users. Our minor goal of updating content in one place and propagating that change to other documents was not implemented in this version of the software but remains a possible enhancement.

What did we learn during this project? We learned how important ease-of-use and a clean user interface are for applications, both for document authors (adding and editing records, uploading documents) and searchers. We learned that different people map information in different ways, and that any knowledge mapping or searching system has to be usable regardless of how we do so.

We learned how important our system administration skills were. Requirement analysis, organizational and project management skills, discussions with vendors, being able to translate between business and technical people, and best practices in programming, software development, and testing were all essential to successful deployment of the repository. Documenting the application for both technical and nontechnical users as well as for the administrators helped the application gain momentum within the company. Having a database-savvy member of the team saved us a lot of unnecessary work as well. We learned that having a project plan, with specific milestones and goals, is essential. And we learned that preventing project creep can be done, provided that the project manager is firm and the requirements are well-documented and well-understood.

We would strongly suggest to anyone interested in implementing a repository that you identify what you want and don't want well in advance. Do not allow yourself the luxury of adding "just one more cool feature" unless it truly is something you need. Consider both free and for-pay software that exists before writing your own if at all possible. If you're in an environment which already uses some tracking or coordinating system (like the Class and Subclass we used here), implement it as part of your database. Talk to your users early and often; if you want them to use the tool they need to understand what's in it for them, what benefits it can provide, and how easy it is to use.

Availability

The source code for the Repository script can be made available upon request. The version deployed in our former company uses many internal code stubs which we are reworking to be generic. At present it still makes several assumptions, including that there is an Oracle database back-end and the appropriate CPAN Perl modules are available. We want to make the script a bit more robust and intelligent to allow for different databases and authorization schemes before publishing the software under some form of public license.

Acknowledgements

The authors would like to thank the following people:

- Mike Stok for developing the original upload script (version 1.0 of the Repository).
- Katherine Ross, Jeff Schouten, Jordan Schwartz, and Emily Stemmerich for the initial bibliographic data collection that let us preload the Repository with nearly 800 data records.
- Bill Huff and Tim Peoples for developing the underlying Perl modules that normalized database authentication and access.
- Cliff Nadler and Ryan Skadberg for their assistance, advice, and implementation ideas.
- Lee Amatangelo, Todd Chapman, Brian Clark, Katrinka Dall, Mark Dawson, Doug Freyburger, Marc Furon, Charles Gagnon, Jason Heiss, Barbara Howard, Gerard Hynes, Brian Kirouac, Ron O'Neill, Joe Royer, Tapan Trivedi, and Jeff Tyler for confirming the integrity of the data before we rolled the application into production.
- Angela Gatto, Brian Worrall, and Julie Zacharias for their tremendous assistance in drafting the initial requirements specification and for their alpha-testing the Repository.
- Alvin Gunkel, Barbara Ingram, Peter Pak, Keith Patterson, Tom Whitley, and Michael Wilson for their assistance in the quality assurance process.
- Ed Bailey, Chris Barnash, Bryon Beilman, Dave Bianchi, Matt Coffey, Steve Cruz, Ralph Dahm, Randy Dees, Ryan DiDomizio, Jim Flanagan, Jeff Giuliano, Mark Jones, David Leonard, Jim Niemira, Jason Powell, Michael Rice, Rodney Rutherford, Joel Sadler, Andy Silva, Ed Taylor, Rob Worman, and David Young for their extensive beta-testing of the Repository.

Furthermore, thanks to the dozens of people within the company who wrote documentation, uploaded it into the Repository, ensured the accuracy of the Repository records, and collected bibliographic

details of others' works to ensure completeness and accuracy of the data in the Repository. Your help brought us to over 1,100 valid document records in under six months of production use.

Finally, thanks to Alva Couch, Æleen Frisch, Tom Limoncelli, and Adam Moskowitz for their assistance in reviewing this paper. Your comments helped us tremendously, even when we didn't agree with them.

Biographies

Josh Simon has 12 years of experience in UNIX system administration, project management, and technical writing. He has a long history of contributions to SAGE, including serving on the SAGE Executive Committee, being the Desk Editor of SAGEwire, chairing the SAGE Online Services Committee, and serving on five LISA program committees. He's written and coordinated summary writeups for several USENIX Annual Technical Conferences and LISA conferences in the past for publication in *;login:*. His non-technical interests include cooking, reading mysteries and science fiction, and plotting to take over the world. Reach him electronically at jss@clock.org .

Liza Weissler has 17 years of experience in UNIX system administration, Oracle database administration, and application development. She worked for the RAND Corporation (Santa Monica, CA) and Collective Technologies before giving up her Southern California native status and moving to southeastern Arizona. She is now happily ensconced in the foothills of the Huachuca mountains, is employed by Management and Engineering Technologies International Inc (METI; www.meticorp. com), and works as a contractor to the US Army's Network Enterprise Technology Command (NETCOM) at Fort Huachuca, Arizona in between vacations. Contact her electronically at liza.weissler@us.army.mil .

Appendix A: Database Schema

Overview

This is the sql used to create the ''repo'' database in Oracle. Basically the tables are:

- **documents** – The main table
- **states** – Where the public/blessed/locked/frozen bits are defined
- **codes** – Internal codes
- **class, class_type** – Define class/subclass
- **authors** – Table to store info on folks no longer with the company

Of the others: long_data is used to store notes for documents. auth_stat helps out with the author sorting and unique author statistics. doc_history is the audit trail.

There are some foreign key constraints defined for the documents table, namely that the state and class/subclass must be defined in the states and class tables. There's no constraint on codes since it can be null.

The initial sequence creation requires:

```
drop sequence doc_id_seq;
drop sequence cl_id_seq;
drop sequence ld_id_seq;
drop sequence dh_id_seq;
drop synonym members;
```

```
drop table documents;
drop table states;
drop table codes;
drop table class;
drop table class_type;
drop table doc_history;
drop table long_data;
drop table authors;
drop table auth_stat;
```

documents **Table**

```
create table documents (
   doc_id          number not null,
   doc_in_auth     varchar2(256),
   doc_non_in_auth varchar2(256),
   doc_sortkey     varchar2(60),
   doc_title       varchar2(256) not null,
   doc_description varchar2(256),
   doc_keywords    varchar2(512),
   doc_revision    varchar2(10),
   doc_published   date,
   doc_url         varchar2(256),
   doc_st_id       number not null,
   doc_public      number(1),
   doc_cd_id       number,
   doc_cl_id       number not null,
   doc_notes       number,
   doc_lastupdate  date,
   doc_updater     number,
   primary key (doc_id)
);
```

states **Table**

```
create table states (
   st_id      number,
   st_desc    varchar2(20),
   primary key (st_id)
);
```

codes **Table**

```
create table codes (
   cd_id      number,
   cd_code    char,
   cd_desc    varchar2(30),
   primary key (cd_id)
);
```

class **Table**

```
create table class (
   cl_id            number,
   cl_class         char,
   cl_subclass      char,
   cl_subname       varchar2(30),
   primary key (cl_id)
);
```

class_type **Table**

```
create table class_type (
   ct_class         char,
   ct_name          varchar2(30),
   primary key (ct_class)
);
```

doc_history Table

```
create table doc_history (
   dh_id            number not null,
   dh_doc_id        number not null,
   dh_st_id         number,
   dh_type          char,
   dh_timestamp     date not null,
   primary key (dh_id)
);
```

long_data **Table**

```
create table long_data (
   ld_id      number not null,
   ld_data    long,
   primary key (ld_id)
);
```

authors **and** auth_stat **Tables**

```
create table authors (
   au_login  varchar2(10) not null,
   au_lname  varchar2(30),
   au_fname  varchar2(30),
   primary key (au_login)
);
```

```
create table auth_stat (
   as_name    varchar2(40) not null,
   as_doc_in  number not null,
   as_isin    number,
   primary key (as_name)
);
```

Synonyms, Foreign Keys, and Sequences

```
create synonym members for resources.members;
alter table documents
 add (foreign key (doc_st_id) references states(st_id) on delete cascade,
  foreign key (doc_cl_id) references class(cl_id) on delete cascade,
  foreign key (doc_notes) references long_data(ld_id) on delete cascade,
  foreign key (doc_updater) references resources.members(mb_em_id)
 );
alter table class
 add (foreign key (cl_class) references class_type(ct_class) on delete cascade
 );
```

```
create sequence doc_id_seq;
grant select on doc_id_seq to repo_role;
create sequence cl_id_seq;
grant select on cl_id_seq to repo_role;
create sequence ld_id_seq;
grant select on ld_id_seq to repo_role;
create sequence dh_id_seq;
grant select on dh_id_seq to repo_role;
```

Triggers

```
CREATE or REPLACE trigger doc_history
BEFORE insert or update or delete on repo.documents
FOR EACH ROW
```

```
BEGIN
```

```
IF DELETING THEN
INSERT INTO repo.doc_history (
       dh_id, dh_doc_id, dh_st_id, dh_type, dh_timestamp
)
VALUES (
       dh_id_seq.nextval, :old.doc_id, :old.doc_st_id, 'D', SYSDATE
);
END IF;

IF UPDATING THEN
INSERT INTO repo.doc_history (
       dh_id, dh_doc_id, dh_st_id, dh_type, dh_timestamp
)
VALUES (
       dh_id_seq.nextval, :new.doc_id, :new.doc_st_id, 'U', SYSDATE
);
END IF;

IF INSERTING THEN
INSERT INTO repo.doc_history (
       dh_id, dh_doc_id, dh_st_id, dh_type, dh_timestamp
)
VALUES (
       dh_id_seq.nextval, :new.doc_id, :new.doc_st_id, 'I', SYSDATE
);
END IF;

END doc_history;
```