



Musings

[Access control](#)
[Adaptive technologies](#)
[Catalogs of the future](#)
[Communication](#)
[Computer literacy](#)
[Day in the life of Mr. D.](#)
[Design elements](#)
[Distance education](#)
[Ebooks](#)
[Emerging technologies](#)
[Gift cultures](#)
[Indexing](#)
[Information systems](#)
[Librarianship](#)
[Marketing](#)
[Mr. Serials Process](#)
[My library in your library](#)
[Proactive services](#)
[Resource sharing](#)
[Search strategies](#)
[Strategic planning](#)
[Systems administration](#)
[Systems librarianship](#)
[Tools of the trade](#)
[Unique collections](#)
[Usability](#)
[WWW and libraries](#)
[Virtual libraries](#)

Creating "Smart" HTML pages with PHP

Or, how to create websites that know about themselves

This text describes a process for creating HTML pages using a database application, an indexer, and PHP -- a cross-platform, open source scripting language especially designed for the Web -- all for the purposes of presenting value-added content.

Contents

1. [Narrative](#)
2. [Code](#)
3. [Links](#)

Narrative

Here's the problem. You want to create HTML pages for your website. Not only do you want your Web pages to say something meaningful, but you also want them to portray a similar graphic identity as well as be syntactically correct. Through Internet spiders/search engines like AltaVista or Google, you want people to identify your site as being relevant. Once a person has identified a page as being particularly useful, you want to make it easy for the person to "find more pages like this one" so you can keep them on your site. This document describes one approach this problem through the combined use of a database application and an HTML indexer tied together with PHP.

HTML pages, especially if sets of them are to have a similar look and feel, can be broken up into parts. For example, there is almost always a footer containing information about when the page was created and who created it. There is almost always some sort of header where much of the page's graphic identity is encoded. Furthermore, each of these pages reside on a particular host in a particular part of a file system. Since many of these things are similar within sets of HTML pages, these things lend themselves very well to fields in tables of relational databases.

An HTML indexer can be used against your sets of pages providing a bit of searchability. This is especially useful if your website is large and not easily browsable. Some HTML indexers can extract the content of HTML META tags and provide access to them through field searching mechanisms. For example, suppose you had the following markup in your HTML:

```
<META NAME="creator" CONTENT="Morgan, Eric Lease">
<META NAME="title" CONTENT="Gift Cultures">
<META NAME="abstract" CONTENT="This short essay examines more closely
the concept of a gift culture and how it may or may not be related to
librarianship.">
<META NAME="date" CONTENT="2001-01-01">
<META NAME="subject" CONTENT="gift cultures; librarianship;">
```

If such markup existed, then some indexers will allow queries such as "subject = 'gift cultures'" and output more relevant search results than a free text query for just plain "gift cultures". Unfortunately, the search syntax from indexer to indexer is slightly different and users rarely take advantage of advanced syntax anyway. Ah, but you have a computer, and maybe you can make the entire process easier. In fact, you can using MySQL, SWISH, and PHP. Here's how.

First, index your existing content (with SWISH). Search it via the GET method and notice how the resulting URLs have a similar structure: prefix, encoded search term(s), suffix.

Second, create a database (in MySQL) accounting for the similar content between sets of HTML pages. Tables will include things like headers, footers, hosts, content, indexer information (prefix and suffix) as well as META data elements such as

authors and subjects.

Third, write a (PHP) script allowing you to do database input as well as output in the form of HTML. PHP is particularly adept at processing HTML forms (for data entry) because form input elements "automagically" become variables in your scripts. PHP knows how to do I/O against your MySQL databases because the MySQL libraries are compiled in by default. HTML forms are easy to create in PHP through the use of the include function. PHP can interact with local and remote file systems by specifying files as URLs or through an FTP interface. Finally, PHP comes with a bevy of string manipulation functions including XML routines.

Once you have done lots of input into your database you will want to finally generate your HTML. This is not too hard. Just join the tables in your database, munge together your HTML, and save the result to the local (or a remote) file system. The trick is to make your pages "smart" by including the subject terms in your document and marking them up with the prefix/suffix of your indexer/search engine, specifically, marking them up as field searches against the META tags.

This approach has many advantages. First, since your data is saved in a relational database, you can make changes in one location and have those changes ripple throughout your content. Second, if the system is designed correctly to begin with, then your resulting HTML should be well-structured. Third, by marking up subject terms with canned searches it will be easy for readers to find similar pages without having to guess about the syntax of queries or hoping for the best when it comes to the indexer relevance ranking algorithm.

Finally, the concepts outlined above can be implemented with any number of technologies, but the MySQL/SWISH/PHP combination is hard to beat. The price is right. The support is there, and they work as advertised.

Code

The following code snippet is from routine in a PHP application called Home Page Maker. The purpose of the routine is to create and save a "smart" HTML page as described in this text. Home Page Maker is available for download. See the Links section as the end of the document.

```
# save; the heart of the matter
elseif ($arg == 'save') {

    if (! sizeof($ids)) {

        # display form
        include "./pages-save-id.inc";

    }

    if (! $username || ! $password) {

        # get username and password
        include './pages-save-username.inc';

    }

    else {

        # process each id
        for ($i = 0; $i < sizeof($ids); $i++) {

            # find this page
            $sql = "SELECT p.*, h.*, d.*, f.*, a.*, e.*
                    FROM pages as p, hosts as h,
                    headers as d, footers as f,
                    authors as a, engines as e
                    WHERE page_id    = '$ids[$i]'
                    AND h.host_id   = p.host_id
                    AND d.header_id = p.header_id
                    AND f.footer_id = p.footer_id
                    AND a.author_id = p.author_id
                    AND e.engine_id = p.engine_id";
```

```

$r = mysql_fetch_array(mysql_db_query (DATABASE, $sql));
checkResults();

# find all subject associated with this page
$sql = "SELECT s.subject
        FROM subjects as s, i4Subjects as i,
        pages as p
        WHERE p.page_id = '$ids[$i]'
        AND p.page_id = i.page_id
        AND i.subject_id = s.subject_id
        ORDER BY s.subject";
$all_subjects = mysql_db_query(DATABASE, $sql);
checkResults();

# initialize subjects
$subjectSearch = '';
$subjectHeadings = '';

# process every subject heading
while ($s = mysql_fetch_array($all_subjects)) {

    # append the term to a list of subject headings
    $subjectHeadings .= $s["subject"] . ' ';

    # initialize the canned query
    $query = $s["subject"];

    # append the term to the canned searches
    $subjectSearch .= '<a href=' . $r["prefix"] .
        rawurlencode("$query") . $r["suffix"] . '>' .
        $s["subject"] . '</a> ';
}

# initialize HTML and build URL
$html = $r["content"];
$url = 'http://' . $r["address"] . $r["path"] .
    "/" . $r["filename"];

# process each macro
$html = str_replace ('##HEADER##', $r["header"], $html);
$html = str_replace ('##FOOTER##', $r["footer"], $html);
$html = str_replace ('##TITLE##', $r["title"], $html);
$html = str_replace ('##AUTHOR##', $r["author"], $html);
$html = str_replace ('##ABSTRACT##', $r["abstract"], $html);
$html = str_replace ('##SUBJECTS##', $subjectHeadings, $html);
$html = str_replace ('##AUTHORURL##', $r["url"], $html);
$html = str_replace ('##AUTHOREMAIL##', $r["email"], $html);
$html = str_replace ('##DATECREATED##', $r["date_created"], $html);
$html = str_replace ('##DATEEDITED##', strftime('%Y-%m-%d'), $html);
$html = str_replace ('##SUBJECTSEARCH##', $subjectSearch, $html);
$html = str_replace ('##URL##', $url, $html);

# open, write, and close the temporary file
$fp = fopen (TEMP . $r["filename"], "w");
fwrite ($fp, $html);
fclose($fp);

# PUT it via FTP
$s = ftp_connect ($r["address"]);
ftp_login ($s, $username, $password);
ftp_chdir ($s, $r["root"] . $r["path"]);
ftp_put ($s, $r["filename"], TEMP .
    $r["filename"], FTP_ASCII);
ftp_quit ($s);

# clean up
unlink (TEMP . $r["filename"]);

# create a list of completed urls
$done .= "<li><a href=$url>$url</a>";

}

# done
echo "Done. See: <ol>$done</ol>";
}

```

}

Links

1. MySQL - <http://www.mysql.com/>
2. SWISH - <http://sunsite.berkeley.edu/swish/>
3. PHP - <http://www.php.net/>
4. Home Page Maker - <http://www.infomotions.com/musings/smart-pages/home-page-maker.tar.gz>

Author: [Morgan, Eric Lease](#) (eric_morgan@infomotions.com)
Date created: 2001-04-08
Date updated: 2001-04-08
Subject(s): [HTML \(hypertext markup language\)](#); [indexing](#); [PHP](#);
URL: <http://www.infomotions.com/musings/smart-pages/index.shtml>