caltechLIB:2001.003

PURR – The Persistent URL Resource Resolver

Ed Sponsler
October 9, 2001
Caltech Library System

CONTENTS

Introduction

URLs have a precarious functional lifespan. They resolve the intended target (web page, PDF, audio stream, and so on) only so long as the target is available, otherwise they'll point to a dead end. Unfortunately, because companies fold, management changes, URL hostnames change, pages move, services are upgraded and reorganized, content becomes obsolete, and a host of other reasons, reliable, persistent URLs are the exception to the norm.

The goal towards URL persistence challenges us with three distinct opportunities to failure: scrapping the service as a whole, changing the hostname portion of the URL or changing the remainder of the URL responsible for navigating to the precise target on the host. Thoughtful management decisions and commitments largely overcome the first two obstacles. A commitment by the institution for preserving a service in perpetuity contributes greatly to service persistence and using an alias hostname frees the service to move from one host to another, and therefore preserve the hostname portion of the URL.

The remaining portion of the URL (following the http://hostname.whatever/ part) provides navigation to the specific target resource, typically by following a file system's directory path to the intended target file. However, whenever the file name or it's location in the file system's directory structure changes, so does the URL.

A typical URL: http://hostname.whatever/example/purr/index.html

This nomenclature is commonly used since web servers are programmed to find (and deliver) the target file by simply parsing the file's path and name from the URL. This is convenient for many web applications but it is a crutch to URL persistence.

A solution is found in assigning a unique identifier to a document and to use that identifier in the URL in place of the directory path/filename.

Example: http://hostname.whatever/PURR:unique.001a

It is then necessary to modify the way in which the web server uses the URL to find the document.

We implemented PURR to provide persistent URLs to documents in our eprint archives. (See http://library.caltech.edu/digital). The unique identifier comprises the archive name followed by a colon, the year and a series number. This identifier is printed on the document header. Thus, any of our documents may be accessed by giving the Resolver the identifier in the form of a URL.

For example the document your reading (number caltechLIB:2001.003) is located here, in the caltechLIB eprint archive:

http://resolver.library.caltech.edu/caltechLIB:2001.003

PURR Implementation

Index of Figures

The CLS Environment

The Caltech Library System (CLS) implements PURR to resolve documents contained in its Digital Collections of eprint archives. The archive software, EPrints.org, stores document metadata in a MySQL database. A script synchronizes information from the EPrints database with the PURR database. Described below is not only the implementation of PURR but also the synchronization routine (synch_purr.pl, Figure 2). You will have to adapt this routine to fit your particular environment.

The development environment:

- Linux
- MySQL
- Apache
- Perl
- mod_perl
- DBI

The core component of PURR is the Resolver.pm perl script which runs as an Apache module. You will need to compile mod_perl into Apache for Resolver.pm to work as published here. mod_perl provides access to the Apache API via perl. For more information, see Stein (1999). MySQL runs the database backend. DBI is a database connectivity module for perl freely available from CPAN. Linux (a unix variant) is used as the operating system because, like everything else in this environment, it is free and flexible.

Apache Configuration

Apache is available here: http://httpd.apache.org
mod_perl is available here: http://perl.apache.org

The following is a basic procedure to compile mod_perl into Apache. It is essentially what we do at CLS. Expand both packages in the same directory, such as /usr/src. Then enter the mod_perl directory and do this:

```
$ perl Makefile.PL \
APACHE_SRC=/usr/src/apache_source_directory \
DO_HTTPD=1 \
USE_APACI=1 \
PREP_HTTPD=1 \
EVERYTHING=1

$ make; make install
```

Then enter the apache directory and do this (at minimum):

```
$ ./configure \
--prefix=/usr/local/apache_installation_directory \
--activate-module=src/modules/perl/libperl.a

$make; make install
```

Create a file called startup.pl, containing the following. Place startup.pl in your apache/conf folder:

```
#!/usr/local/bin/perl
```

```
BEGIN {
   use Apache();
   use lib Apache->server_root_relative('lib/perl');
}
use Apache::Registry();
use Apache::Constants();
use CGI qw(-compile :all);
use CGI::Carp();

1;
```

Add these lines to your /apache/conf/httpd.conf file:

```
PerlRequire conf/startup.pl
PerlFreshRestart On

SetHandler perl-script
PerlHandler Apache::Resolver
```

Place the Resolver.pm script (see Figure 3) into apache/lib/perl/Apache.

Restart apache.

Database Configuration

The main table

Figure 1 shows the 'main' table fields and properties. There is a primary key (pkey), the unique identifier (id) the URL to the document (url) and a miscellaneous notes field. Although it isn't used (yet), the title is stored to provide some human recognizable label to the record.

```
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| pkey  | int(11)      |      | PRI | 0       | auto_increment |
| id    | varchar(255) |      | UNI |         |                |
| url   | varchar(255) |      |     |         |                |
| notes | text         | YES  |     |         |                |
+-------+--------------+------+-----+---------+----------------+
```

Figure 1 -- Database Structure

Here is a SQL script to produce this table. Login with the root mysql account, open the mysql database and:

```
mysql> create table main (pkey int not null primary key auto_increment,
    -> id varchar(255) not null, url varchar(255) not null, notes text,
    -> unique index index_id (id))
    -> go
```

Resolver.pm queries this table to retrieve the document's URL based on the document's Unique Identifier (supplied in the URL sent to Apache).

Access control

You will need to supply the appropriate access controls. The following is a basic security setup. Let us assume that the username is 'resolver', the password is 'password' and access to the database is occurring on the local machine 'localhost'. Then, you will need to first make an entry into the user table (with no specific controls other than to allow 'resolver' to connect from 'localhost'). Login with the root mysql account, open the mysql database and:

```
mysql> INSERT INTO user (host, user, password)
    -> VALUES ('localhost', 'resolver', password('password'))
    -> go
```

Next provide specific access controls on the PURR database from the localhost. If the database is named 'purr_eerl' then execute this SQL statement:

```
mysql> insert into db (host, db, user, select_priv, insert_priv,
    -> update_priv, delete_priv) values ('localhost', 'purr_eerl',
    -> 'resolver', 'y', 'y', 'y', 'y')
    -> go
```

After making any changes to the access tables, reload the database:

```
$ mysqladmin -u root -p reload
```

Populating the PURR Database

As mentioned previously, the Library uses PURR to resolve documents in its Digital Collections archives run with EPrints.org software. So the following information get quite specific for that environment.

One way or another, you will need to put data into the PURR database. The following script, synch_purr.pl extracts data from an EPrint archive database and imports it for immediate use into the PURR system described in this report.

A special note: each EPrint archive must have its own PURR database. The reason will become clear as you read on. An obvious consequence of this is to be sure to run slightly modified version of synch_purr, one for each EPrint archive, perhaps on a nightly cron schedule.

Some notes on the script

The script assumes that the EPrints database is on the same host / port as the PURR database, and that access to read the EPrints data is granted to user 'resolver'.

The script must insert three pieces of information into the PURR database.

- The Report Number
- The Destination URL
- The Title (optional)

The Report Number stored in the EPrints database does not include the archive name. That is, if the Unique Identifier of a document was 'PURR:unique.001a' then the value stored in EPrints is 'unique.001a'. However, since there is a separate PURR database for each archive, this isn't a problem. You will see in Resolver.pm how these pieces come together.

The URL which our resolver returns is to the main EPrint document 'service' page.

An example: http://caltecheerl.library.caltech.edu/documents/disk0/00/00/09/27/index.html

This URL is not stored anywhere, but may be reconstructed using available data. The base URL is entered as a variable near the top of the script. The string '/documents/disk0/' is always the same and so is hard coded. The numbers following are derived from the internally used Eprint ID, a field in the EPrint database ('eprintid'). In the URL example above, the eprintid is 00000927. And of course 'index.html' is always the same and hard coded.

The title is stored as an EPrint field.

Kobes, et. al., (2001) treat the subject of perl DBI programming in Chapter 6 of their excellent book, "Professional Perl Development".

```
#!/usr/local/bin/perl -w
# synch_purr.pl
# Ed Sponsler, August 17, 2001
# This script exports data from an EPrints.org archive and imports
# necessary data into a PURR database.

use strict;
use warnings;
use DBI;

our ($dsn_export, $dbh_export, $sth_export, $purr_db,
    $dc_db, $script_user, $purr_server, $purr_port, $db_user, $db_pass,
    $url_base);

$purr_db = 'purr_eerl'; # PURR database name (unique for each EPrints db)
$dc_db = 'eerl'; # Digital Collection EPrint archive database name
$db_user = 'resolver'; # DC and PURR database username
$db_pass = 'password'; # DC and PURR database user password
$url_base = 'caltecheerl.library.caltech.edu'; # base URL of EPrint archive

$dsn_export = "DBI:mysql:database=$purr_db";

$dbh_export = DBI->connect($dsn_export, $db_user, $db_pass);
$sth_export = $dbh_export->prepare("DELETE from main");
$sth_export->execute;

my ($dsn_import, $dbh_import, $sql_import, $sth_import);
our ($a, $b, $c, $d, $url, $title, $qtitle, $reportno, $eprintid);

$dsn_import = "DBI:mysql:database=$dc_db";
$dbh_import = DBI->connect($dsn_import, $db_user, $db_pass,
   {RaiseError => 1});
$sql_import = "SELECT title, reportno, eprintid from archive";
$sth_import = $dbh_import->prepare($sql_import);
$sth_import->execute;
$sth_import->bind_columns(\($title, $reportno, $eprintid));
while ($sth_import->fetch)
{
   $a = substr($eprintid,0,2);
   $b = substr($eprintid,2,2);
   $c = substr($eprintid,4,2);
   $d = substr($eprintid,6,2);

   $url = "http://$url_base/documents/disk0/$a/$b/$c/$d/index.html";
   $qtitle = $dbh_export->quote("$title");
   $sth_export = $dbh_export->prepare("INSERT into main SET id='$reportno', url='$url',
notes=$qtitle");
   $sth_export->execute;
}

$sth_export->finish;
$dbh_export->disconnect;
$sth_import->finish;
$dbh_import->disconnect;
```

Figure 2 -- synch_purr.pl

The Resolver.pm Script

Resolver.pm is called by Apache at the appropriate time during the processing of a request. Apache passes the 'request object' to this script and immediately runs the 'handler' subroutine. The first thing that 'handler' does is copy the 'request object' into $r. Many Apache subroutines are now available through $r. For more details, see Stein, et. al, (1999).

The convention CLS uses for constructing Unique Identifiers is:

archive_name:report_number

The discussion on synch_purr pointed out that only the report_number is stored in the EPrints database, and so, this is the only number stored in the PURR database. However, there is a separate PURR database for each EPrint archive. Since the archive_name is included in the URL, Apache is able to figure out which PURR database in which to lookup the URL based on the report_number given.

The hash %db is used in the get_url subroutine to translate the archive name included in the URL into the appropriate PURR database. So given a URL:

http://resolver.library.caltech.edu/caltechEERL:1988.023

It can split the caltechEERL:1988.023 into two parts, caltechEERL and 1988.023. The first is used to determine which PURR database to look up, and use the second is used to lookup which record within that database contains the URL to return to the requestor.

In this way, a single Resolver.pm script may be used for an indefinite number of PURR / EPrint databases, just by adding additional pairs in %db.

The basic error handling is to redirect the user to our main Digital Collections page if no Unique Identifier is given to resolver.caltech.edu and to produce an error page with email link if the Unique Identifier is garbage or non-existent.

```
package Apache::Resolver;
use Apache::Constants qw(REDIRECT);
use DBI;

our %db; # This hash will translate archive names into the PURR db name

%db = (
   caltechCSTR => 'purr_cstr',
   caltechEERL => 'purr_eerl',
   cav2001 => 'purr_cav',
);

our ($db_user, $db_pass);
$db_user = 'resolver'; # database username
$db_pass = 'password'; # database user password

sub handler {
   my ($r, $uri, $id, $url);
   $r = shift;
   $uri = $r->uri;
   if ($uri eq '/') {
      $r->header_out(Location => 'http://library.caltech.edu/digital');
      return REDIRECT;
   }
   $id = substr($uri,1);
   $url = get_url($id);
   if ($url) {
      $r->header_out(Location => $url);
      return REDIRECT;
   }
   else
   {
      $r->content_type('text/html');
      $r->send_http_header;
      $r->print(<<_HTML_);
<HTML>
<HEAD>
<TITLE>Error: Not Found</TITLE>
</HEAD>
<BODY>
<H1>Error</H1>
<P>
The Unique ID: <b>$id</b> cannot be found.<br>
Try again, or <a href="mailto:web\@library.caltech.edu">contact us</a>.
</P>
</BODY>
</HTML>
_HTML_
   return OK;
}
}

sub get_url {
   my ($id, $dsn, $dbh, $sql, $sth, $url, $rep_id, $reportno);
   $id = shift;
   ($rep_id, $reportno) = split (':', $id, 2);
   unless ($db{$rep_id}) {return 0};
   $dsn = "DBI:mysql:$db{$rep_id}";
   $dbh = DBI->connect($dsn,$db_user,$db_pass);
   unless ($dbh){die "Can't open database: $DBI::errstr";}
   $sql = "SELECT url from main WHERE id='$reportno'";
   $sth = $dbh->prepare($sql);
   $sth->execute;
   $sth->bind_col(1,\$url);
   $sth->fetch;
   $sth->finish;
   $dbh->disconnect;
   return $url;
}
1;
```

Figure 3 -- Resolver.pm

A Couple Useful Books

Kobes, Randy; Wainwright, Peter & Gundavaram, Shishir (2001) "Professional Perl Development". Wrox Press, Inc. [Esspecially Chapter 6, "Databases"]

Stein, Lincoln & MacEacherm (1999) "Writing Apache Modules with Perl and C: The Apache API and mod_perl". Oreilly & Associates, Inc.