

Everything Engine

Jani Patokallio

22.3.2002

The Everything Web System

*“Another Dumb Perl-MySQL
Web Content-Management System”*

More specifically, it uses a unified object model rendered in a database and XML to create a set of modular tools for moving around blocks of text and HTML.

Competitors:

- Slash, Scoop, ...
- WebDynamo, Escio Portal ...

Basic Everything Concepts

“Within Everything, everything is a node”

Any keyword is a *node*

A node can have multiple *writeups* by users

Nodes link to other nodes:

- *hardlinks* are specified by the author
- *softlinks* are determined by popularity

Everything is Not a Blog

Blogs are in a linear article/followup format
Everything is a free-form database

Blog content is linked into threads (parent/reply)
Everything content links to other nodes

Blog content is created by blog maintainers
Everything content is created by users

Everything is Not a Wiki

Wikis have one article per keyword
Everything can have many

Wiki articles do not have authors
Everything articles do

Wiki articles are all equal
Everything articles are moderated through
voting, editing, and deletion

Major Everything Engine Sites

- Everything²
- Perl Monks
- Java Junkies
- SourceForge Clustering Foundry
- AnimeFu

History: Everything1

Original developers:

- Nate Oostendorp
- Rob “CmdrTaco” Malda

Hired:

- Ryan “dem bones” Postma
- Darrick Brown

Slashdot sold to Andover

Incorporated as Blockstackers Intergalactic (BSI)

History: Everything²

Everything¹ had limited usability

- Max two writeups per node
- Max 512 chars per writeup
- Few community features...

Everything² born November 13, 1999

At peak 7 hired people working

Net boom fizzled, all employees fired
Continues with volunteer labor

Everything² Statistics

~\$500 in donations/month
45,000 distinct users
930,000 nodes in database
5,000,000 impressions/month

Everything Engine \neq Everything²

Everything Engine

- The underlying software
- Aims for stability
- Still in pre-1.0

Everything²

- The forerunner and testing grounds
- Runs a very tweaked version of EE
- Has a (partly) separate dev team

Development and Licensing

Development

- “Quiet bazaar”
- Code development on Sourceforge
- Developer discussions on EveryDevel

License

- Artistic License (as in Perl)
- non-viral Open Source

Installation

Well documented, but painfully complex

Requirements:

- Linux (pref. Debian, but Red Hat also supported)
- Perl and *lots* of modules
- Apache and mod_perl
- MySQL
- “nodeballs”

Documentation

Comprehensive
Maintained almost entirely online

“The Everything Bible”

- An 8-chapter guide to setting up and configuring EE

Everything Documentation Index

- Editorially maintained list of tips, tricks, features, bugs, questions, answers...

Structure

bin/	Installer script
docs/	Pointer to EveryDevel
Everything/	Core code (Perl)
nodeballs/	Content (XML & more)
scripts/	Helper scripts for installer
tables/	SQL code for installer
web/	Apache config for installer

Code

Clear division between “public” API for Everything operators and “private” code for developers

Everything: 4 kLOC core Perl (with SQL)

- API clear and well commented

Nodeballs: 15 kLOC XML (with HTML and Perl)

- Not intended to be modified
- Hellishly complex, no comments at all!

Everything::NodeBase

```
sub getNodeByName {
    my ($this, $node, $TYPE) = @_ ;
    my $NODE, $cursor ;

    $this->getRef($TYPE) ;

    $NODE = $this->{cache}->getCachedNodeByName($node, $$TYPE{title}) ;
    return $NODE if(defined $NODE) ;

    $cursor = $this->getDatabaseHandle()->prepare(
        "select * from node where title=? && " .
        "type_nodetype=" . $$TYPE{node_id}) ;
    $NODE = $cursor->fetchrow_hashref() ;
    $cursor->finish() ;

    # OK, we have the hash from the 'node' table. Now we need to
    # construct the rest of the node.
    $this->constructNode($NODE) ;
    return $NODE ;
}
```

ecore/htmlcode/titlebar.xml

```
<NODE export_version="0.5" nodetype="htmlcode" title="titlebar">
  <field name="author_user" type="noderef" type_nodetype="user,nodetype">
  <field name="authoraccess" type="literal_value">iiii</field>
  <field name="code" type="literal_value">my ($delimiter) = @_;
$delimiter ||= " | ";
my $str, @links;
push @links, linkNode ($HTMLVARS{default_node}, "Log $$USER{title} ou
push @links, linkNode ($USER, "Edit $$USER{title}'s settings", { 'dis
my $EN = getNode('epicenter nodes', 'nodegroup');
foreach (@{ $$EN{group} }) {
  my $N = getNode $_;
  push @links, linkNode($N) if $N->hasAccess($USER, 'r');
}
$str = join "$delimiter\n", @links;</field>
  <field name="dynamicauthor_permission" type="literal_value">-1</fie
  ...
  <field name="title" type="literal_value">titlebar</field>
  <field name="type_nodetype" type="noderef" type_nodetype="nodetype,
</NODE>
```

ecore/htmlcode/insertJavascript.xml

```
<NODE export_version="0.5" nodetype="htmlcode" title="insertJavascript"
  <field name="author_user" type="noderef" type_nodetype="user,nodety
  <field name="authoraccess" type="literal_value">iiii</field>
  <field name="code" type="literal_value">return "" unless(int(keys %
my $str = "&lt;script language='javascript'>\n";
foreach my $key (keys %INCJS)
{
  my $js = $DB->getNode($key, 'javascript');
# We need to check to see if the javascript is "dynamic". Dynamic
# javascript has embedded perl code in it that we need to parse.
# This way we can have perl dynamically generate javascript based
# on the current node, user, etc. WOO HOO!
  $str .= ($$js{dynamic} ? parseCode('code', $js) : $$js{code});
  $str .= "\n";
}
$str .= "&lt;/script>";
$str;</field>
  <field name="dynamicauthor_permission" type="literal_value">-1</fie
  ...
</NODE>
```

Future

- Complete v1.0
- Mobile E²
- E² subscriptions
- Commercial use?
- ...