Drupal.org - Internal messaging

Gerhard Killesreiter

Contents

1 Motivation

It has been observed that users at drupal.org are using a shorthand, such as "@Dries", to address each other and make a particular user aware of a point they want to make. While it is mostly clear who is addressed, it is not neccessarily clear that the addressee will actually take notice of the message. The likelihood of the addressee noticing the message is only significant if the addressee is already monitoring the post in question.

We will outline a method to

1. increase the likelihood of the addressee noticing messages in already monitored posts,
2. enable participants to alert a user to posts he doesn't observe yet,
3. and do all of the above across all subdomains of drupal.org.

In addition, we'll allow users to use the hashtag notation, e.g. "#infrastructure", "#DA".

2 Implementation

2.1 Filter

The first step is a filter that filters all new input on each site on *.drupal.org. This can not be a filter in the usual Drupal sense, since these filters are not context-aware. Also, our filter needs to act on the final HTML that is shown to the user. The reason for this is that the filter configuration on each subsite can be different and we do not want to keep track of this, but we do want give subsites the change to use meaningful filters.

The filter will thus be invoked by hook_nodeapi "insert" and hook_comment "insert" and call the theme function for nodes or comments to get the correct HTML.

What the filter does is

1. go through the HTML, search for @ and #, and generate a snippet containing some context before an after,
2. send the snippet to storage.

This approach has a slight security issue: The stored and filtered content will no longer be regarded as user-input, since it is filtered. As such we have to display it unfiltered when we finally display it. This means that all sites need to have up–to–date filter settings or it will be easily possible to extend illegetimate access from one subsite to all others.

There is also an issue regarding what a username is. While it is clearly defined in the users table, the actual users may refer to each other by abbreviations thereof, e.g. people refer to me as "killes" but that is not my username. It is however my IRC handle. More on this in 2.3 and 2.4.

2.2 Storage

To store the preprocessed snippet we'll use a non-relational database such as MongoDB.

The stored information should contain

- username, local uid, and link to bakery master profile of the user who created the snippet
- a timestamp
- name and URL of the subsite the snippet originates from
- the actual snippet
- the URL that the node or comment can be found on

- ...

One DB will hold the information for all subsites.

2.3 Processing

First we need to transfer the snippet found in 2.1 from the MongoDB database to a MySQL one. While we do that we'll also do some processing.

This can be done by a drush command running in some Drupal instance.

The schema of the MySQL table should be optimised to allow for efficient queries.

The difficult part is to connect the snippets with the account of the user that is addressed. As already pointed out in 2.1, this relationship is not neccessarily clear.

Instead of trying to do some sort of pattern matching, we take the approach to allow users to subscribe to certain @-tags and hashtags.

In their dashboard, they'll be able to configure blocks for these. One block can be for one or several @-tags and hashtags.

For any text following @ or # up to a stop character (space, colon, ...) we'll assume that it is a term. There are usernames which contain spaces, but we'll assume that nobody will actually use them for our purpose. E.g. Moshe Weitzman's username is "moshe weitzman", but people tend to use "@moshe" when referring to him (561 comments on drupal.org use "@moshe*", 47 "@moshe weitzman*", the former including the latter), e.g. http://drupal.org/node/14936#comment-313082.

For each term found in this way, there is a term created in a dedicated vocabulary, if it doesn't already exist. At this point we'll lose the distinction between @-tags and #-tags. This is done deliberately to keep things simple.

The term is referenced in the table that holds the actual snippet.

Before the snippet can be saved there, it needs to be cleaned up: any relative URLs need to be prefixed with the subdomain where the snippet came from and broken HTML needs to be removed.

2.4 Display

As mentioned in 2.3 we want to allow users to choose which terms they track. The general message listing will contain the snippet and a link to the node or comment where it originated as well as a the username of the user who wrote it.

A user can add new terms to existing messaging blocks or create new blocks.

One obvious issue is privacy, there is essentially none. Any user can choose to watch the "Dries" term. This is not really an issue since all content on drupal.org is public anyway.

2.5 Cleanup

With about half a million user accounts, 700 new nodes per day, and over 2500 comments per day alone on the main site, a lot of snippets can be generated.

Snippets will be deleted as soon as nobody is watching the corresponding terms anymore and they have reached a certain age. Terms without snippets can be deleted anytime.

3 Possible extensions

3.1 Direct messaging

This would allow direct creation of snippets without going creating a comment or node anywhere. It would run on a subsite that could have a very reduced look, i.e. only a field to add new snippets. This would provide a similar user experience to a microblogging service.

3.2 External availability

It should be easily possible to make each feed available through methods for external reading, ie Mail, XMPP or RSS.

3.3 Other ideas

I am interested in getting input from the community on possible extra features.