Newscoop 3 Cookbook

The open content management system for professional journalists

TEMPLATE DESIGN REFERENCE

SOURCEFABRIC

Newscoop
SOURCEFABRIC

NEWSCOOP 3 COOKBOOK

Updated for version 3.5.2

SOURCEFABRIC
www.sourcefabric.org

NEWSCOOP 3 COOKBOOK
THE OPEN CONTENT MANAGEMENT SYSTEM FOR PROFESSIONAL JOURNALISTS

PUBLISHED: April 2011

Updated for Newscoop 3.5.2

This manual has been edited and reworked in a collaborative effort led by FLOSS Manuals. Thanks to all contributors!

COVER DESIGN: Till Sperrle, ITF Grafikdesign

DOCUMENT CREATION: FLOSS Manuals

ISBN: 978-3-9814137-1-7

NEWSCOOP IS DEVELOPED AND MAINTAINED BY SOURCEFABRIC.
COMMENTS AND QUESTIONS CAN BE SENT TO:
Sourcefabric o.p.s.
Salvátorská 10
110 00 Praha 1
Czech Republic
contact@sourcefabric.org
www.sourcefabric.org

PUBLISHED BY: Sourcefabric Publishing · Micz Flor · Prinzessinnenstraße 20 · 10969 Berlin

TABLE OF CONTENTS

INTRODUCTION

1 Ready to cook? 2
2 Newscoop sites to get you inspired 4
3 About this book 8
4 Who we are 9
5 Resources 11

PLANNING YOUR PUBLICATION

6 Turning your idea into a publication 13
7 Planning your workflow 16
8 Subscriptions and revenue 18
9 Structuring your content 23

FROM DESIGN TO TEMPLATES

10 The power of $gimme 26
11 Best practices for designing your content 30
12 Preparing designs for templates 34
13 Organising template files 37
14 Tips and tricks for the template novice 40
15 Installing template packs 43

MAKING YOUR TEMPLATES

16 Publication 46
17 Issues 50
18 Sections 52
19 Articles 54
20 Article comments 59
21 Sites in multiple languages 65
22 Search templates 68
23 Making a 404 page template 72

ADVANCED TEMPLATING

24 Topics, switches, keywords to structure content 75
25 Managing static pages 79
26 List of popular articles (most read) 81
27 List of latest comments 83
28 Using maps 85
29 Managing multiple authors and articles 89
30 Subscriptions, registration and content access 90
31 Browser detection and robots (search engines) 97
32 Mobile device detection and templates 101
33 Tag clouds using topics 103
34 Dynamic page layouts 106
35 Working with images 107
36 Pagination of lists and long articles 111
37 Audio player for attached MP3 files 114

38 RSS, sitemap, KML and XML 117
39 Search engine optimisation (SEO) 125
40 Newscoop and HTML5 130
41 Troubleshooting templates 132

TEMPLATES USING JQUERY, PHP AND THIRD PARTY SERVICES

42 Using jQuery and PHP in templates 139
43 Pulldown menu with Superfish using jQuery 142
44 Tabs with jQuery 146
45 Ticker with jQuery 148
46 Image galleries with jQuery 149
47 Calendars with jQuery 153
48 Using polls with AJAX 157
49 Widgets: YouTube, Vimeo, Twitter, Flickr & Co. 159
50 Profile pictures: Gravatar, Facebook, Twitter 164
51 Combining Newscoop and SoundCloud 166
52 Working with Facebook 169
53 Working with advertising 172
54 Ways to create archives 174

TEMPLATE REFERENCE

55 Language elements and conventions 181
56 Basic syntax 183

TEMPLATE REFERENCE - OBJECTS

57 Introduction to Newscoop objects 189
58 Article object and attachment, comment, location 190
59 Author and author biography 197
60 Browser 199
61 Image 203
62 Issue 204
63 Language 205
64 Publication 206
65 Section 207
66 Subscription 208
67 Subtitle (subheads in long articles) 209
68 Template 210
69 Topic 211
70 URL 212
71 User 214
72 Default objects (list of all available objects) 215

TEMPLATE REFERENCE - CONTROLLING OBJECTS

73 Set objects 217
74 Unset objects 223
75 Local - temporary variable environment 226

TEMPLATE REFERENCE - LISTS

76 List articles and article related content 228

77 List article topics and subtopics 241
78 List of issues and sections 244
79 List subtitles (pagination of long articles) 248
80 List images 250
81 List languages 253
82 List search results 255

TEMPLATE REFERENCE - FORMS

83 Search form 258
84 Login and registration form 260
85 Subscription form 265
86 Comment form 267
87 Action check on form submission 269
88 General form elements and functions 272

TEMPLATE REFERENCE - MAPS AND GEOLOCATION

89 Map 275
90 Displaying a map 276
91 Location multimedia 278
92 Geolocation search 279

TEMPLATE REFERENCE - MODIFIERS

93 Date and e-mail formatting 281
94 File size formatting 283
95 Truncating UTF8 strings 284
96 URL display and modifying 285

APPENDIX

97 Credits and license 290

INTRODUCTION

1. READY TO COOK?
2. NEWSCOOP SITES TO GET YOU INSPIRED
3. ABOUT THIS BOOK
4. WHO WE ARE
5. RESOURCES

1. READY TO COOK?

Have your tools ready. Photo by Brittany Silverstein (CC-BY).

Welcome to the Newscoop 3 Cookbook! Newscoop is an open content management system for professional journalists. Designed by newsroom experts, it powers professional newspapers and magazines worldwide. It meets the demands of modern journalism with intuitive interfaces, flexible publishing, multilinguality, geolocation tools, social media integration, and powerful templates compatible with HTML5 and mobile devices. You can find out more about Newscoop at http://newscoop.sourcefabric.org

Like every good cookbook, this book has lots of good recipes, ideas and tips for creating online publications with good taste, and which taste good too. And, like every good cook, there are a few utensils and ingredients you should have in your kitchen and ready to go. This manual will provide you with enough advice and code examples to get a high-quality site up and running with Newscoop.

The Newscoop 3 Cookbook will walk you through all the stages of creating an online publication. We start at the planning stage, because in our experience at Sourcefabric, careful planning helps avoid future obstacles. Then we walk through the design process, into the stage of converting the design into templates. The following chapters guide you from the essential need-to-know, to the advanced, and finally more exotic ways of using Newscoop for your publication.

You can get going quickly with the template packs from the Sourcefabric website. It's possible to adjust these pre-made template packs to your needs. In the long term, spending time to understand the possibilities of Newscoop will help you to create a unique publication that stands out from the crowd. It also allows time to reflect on the organic processes unique to each news organization, and therefore to each Newscoop deployment.

WHERE TO GO FOR INFORMATION, HELP AND SERVICES

We have more documentation on Newscoop. There is a manual aimed at journalists and editors, a wiki based reference guide, and more. You can access these resources at:
http://manuals.sourcefabric.org

There are support and developer forums provided by Sourcefabric. Meet the Newscoop teams, ask for help and offer support if you know the answers. These forums can also be followed via e-mail.
http://forum.sourcefabric.org/

You can subscribe to our newsletter and read the latest blog posts at:
http://www.sourcefabric.org/newsletter

And see the development milestones, report bugs and request features at:
http://dev.sourcefabric.org

Our development wiki with the latest documentation can be found here:
http://wiki.sourcefabric.org

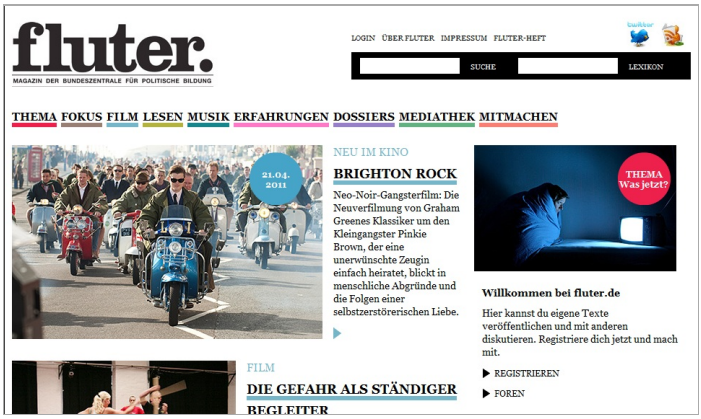
If you need help or want Sourcefabric to do the job for you, get in contact with Services at:
http://services.sourcefabric.org

Sourcefabric provides guaranteed paid support and services for Newscoop, including custom template development, consulting, project management and implementation.

2. NEWSCOOP SITES TO GET YOU INSPIRED

Before we get right into Newscoop templates, first we can take a look at what others have achieved. Newscoop's community is diverse and international, and the sites it powers are also quite diverse in their coverage and specialization. Some are daily newspaper sites, some are online portals, some are radio stations, and some are superpowered blogs.

FLUTER.DE



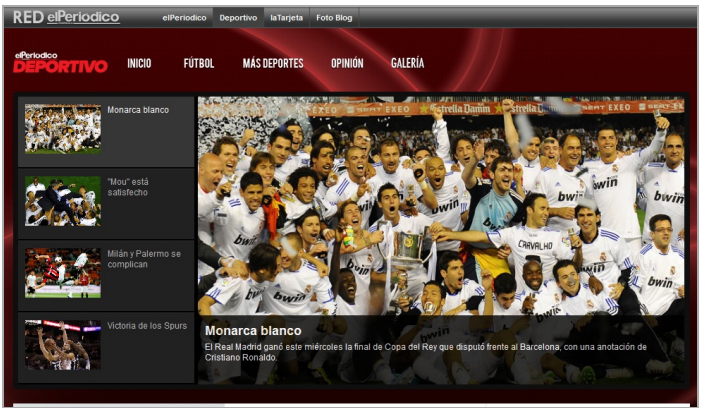
A youth culture portal based in Berlin, http://www.fluter.de has been an active member of the Newscoop community almost since its beginning. Fluter's installation makes effective use of jQuery (notably Superfish pulldown menus - described later in this cookbook). Fluter's 'Mediathek' is also notable - it's an online archive of video and audio that uses Newscoop's media archive and enables site visitors to make their own video playlists.

WEST AFRICA DEMOCRACY RADIO

West Africa Democracy Radio http://www.wadr.org is a news and talk radio network, with content broadcast on FM, satellite, and online. It has more than thirty affiliate stations covering the region. WADR's website makes extensive use of Newscoop's multilinguality, Soundcloud plugin, news ticker, topics, and maps - all of which are documented in this Cookbook.

EL PERIODICO

Guatemala's acclaimed investigative newspaper has two notable sites running on Newscoop - its main site at http://www.elperiodico.com.gt and a sports site at http://deportes.elperiodico.com.gt.

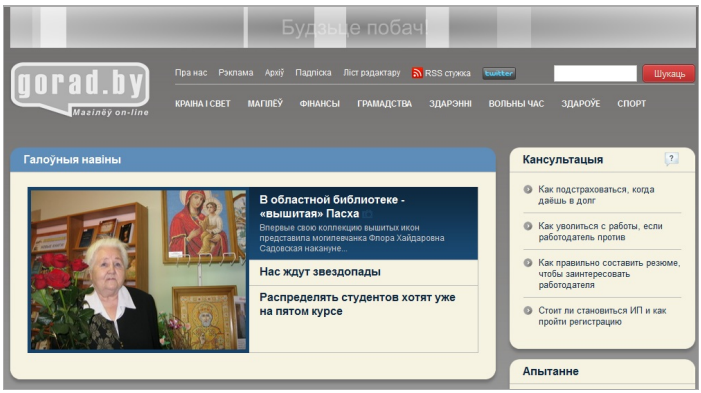


Both sites make extensive use of jQuery; the news site enables readers to customize their home pages, for example. ElPeriodico also provides access to an important archive which serves all articles since the site launched in 2004 (check the chapter *Ways to create archives* in this book).

EL FARO

Central America's first online news site http://www.elfaro.net is updated continuously, every day. El Faro's "Radionoticias" service uses article attachments and a Flash audio player to serve a large amount of audio (see the Cookbook chapter *Audio player for attached files*). They also make extensive use of photojournalism, based on Newscoop's media archive (see the chapter *Working with images*).

GORAD.BY



Minsk's http://www.gorad.by is an online newspaper which posts about twenty items per day. The Newscoop-powered site uses jQuery slideshows extensively (see the chapter here on *Image galleries with jQuery*), calendar-based archives (explained in the chapter *Calendars with jQuery*), and Facebook integration (described in the chapter *Working with Facebook*).

KOSMOBLOG

Jakub Gornicki is a leading blogger in Poland, who moved his blog to Newscoop at the beginning of 2011 when he started working with Sourcefabric. Using a new sample template set based on Lucian Marin's "The Journalist" theme adapted for Newscoop, Kosmoblog makes use of Newscoop's multilingual features (he blogs in Polish and English), search engine optimization techniques like keywords in the URL and sitemap (described in the chapter *Search Engine Optimization*) and Disqus comments (also described later).

You can find a wide-ranging list of sites using Newscoop on the Sourcefabric website: http://sourcefabric.org/en/products/newscoop_whosusing/

3. ABOUT THIS BOOK

Updated to Newscoop 3.5.2

The Newscoop 3 Cookbook is a work in progress and open to your input. Dedicated to Open Source software development, Sourcefabric believes that the software and its documentation are open to the community. The more contributors add to the products, the more errors get caught and fixed, the more features are implemented. We also believe that software improves from good documentation, and documentation improves with the wide use of software.

If you experience the Newscoop 3 Cookbook in its printed form, it might be difficult to see how this book is open to your contribution. If you have come across the Newscoop 3 Cookbook online at FLOSS Manuals, you might get a better idea about how you can participate in improving this documentation. Register, log in and add your knowledge, use cases, your code snippets, comments and best practices to make this a better publication. Working with the online book production platform Booki, the content of this book is continuously evolving. Chances are, the latest PDF, ePub or the online version on the site http://www.flossmanuals.net contains more details than the printed copy.

"All work and no play..." - the Newscoop Cookbook team at work.

Sourcefabric started this publication with a one-week Book Sprint in Schloss Neuhausen, Germany, two hours north of Berlin. Authors for the Newscoop 3 Cookbook include (in reverse alphabetical order): Holman Romero, Ljuba Ranković, Lucian Marin, Micz Flor, Alexei Danilchuk and Douglas Arellanes. Our facilitator, chef, sprintmaster, captain and all-around make-it-happen person was Adam Hyde of FLOSS Manuals. Thanks also go to Katerina Michailidis for her logistical support and to Daniel James for copy and style editing.

If you contribute to this book, you may also want to add yourself as an author here...

4. WHO WE ARE

The Newscoop community comes from a diverse and international background of journalists, editors, developers, implementers, trainers, designers, documentation writers and others who care about Free Software and Free Media. The community finds its focus in Sourcefabric http://www.sourcefabric.org, the non-profit foundation which maintains the Newscoop project, as well as other tools and activities related to its mission of providing media organisations with the open source software, tools and support to produce quality journalism.

A last minute photo shoot from the annual "Sourcecamp" community gathering, held in Berlin in 2010, before the participants - who came from 12 countries - got into their trains, planes and automobiles back home.

Newscoop was first released under the name of "Campsite" in 2000. After substantial rewriting, it was renamed to "Newscoop" in January, 2011. Newscoop and Campsite have always been free and open source (always under the GNU General Public License) and will remain so.

About Sourcefabric

Sourcefabric o.p.s. is a not-for-profit organisation based in Prague, Czech Republic, with branches in Berlin, Germany and Toronto, Canada, and satellite offices in Warsaw, Minsk, Cluj, and Guatemala City.

The Sourcefabric partners Sava Tatić, Micz Flor and Douglas Arellanes were among the key people behind an initiative led by the Media Development Loan Fund http://www.mdlf.org to support open source solutions for independent media organizations in emerging democracies. The Media Development Loan Fund's Center for Advanced Media, Prague, or CAMP, consolidated a lot of work that had been happening since 1999 to develop user-centric, financially-viable, multilingual platforms for journalists. Sourcefabric is the continuation of that endeavour, but with greatly expanded aims and a new organisational structure.

We are concerned with the 'fabric' of these things - their structures, relations and interactions. At the core of our experience and vision is a commitment to enable quality journalism. We do this through open source software and services that address these structures with solutions that are flexible, strong and interwoven; just like fabric.

5. RESOURCES

In this Cookbook we often refer to the Newscoop template pack called "The Journal". If you are interested in sniffing out some other sneaky tricks in this (or other) template packs, you can go to the Sourcefabric website and download template packs from http://www.sourcefabric.org/en/products/newscoop_templates/

TOOLS

Before you begin a Newscoop implementation, getting these things together really should be your first steps.

- Working Newscoop installation
- Text editor, preferably one with PHP markup
- Image editor for preparing designs and images (GIMP, Photoshop, etc.)
- Site specification (see the next chapter, *Turning your idea into a publication*)

The steps to successfully install Newscoop can be found in the "Newscoop for Journalists and Editors" manual at http://manuals.sourcefabric.org

While Newscoop includes a template editor which works perfectly well, you'll probably prefer to work locally on your computer. Good text editors include Gedit on Ubuntu, Kate on Kubuntu, Notepad++ or even Dreamweaver on Windows, and of course command line editors like Emacs, vi, pico and nano.

PLANNING YOUR PUBLICATION

6. TURNING YOUR IDEA INTO A PUBLICATION
7. PLANNING YOUR WORKFLOW
8. SUBSCRIPTIONS AND REVENUE
9. STRUCTURING YOUR CONTENT

6. TURNING YOUR IDEA INTO A PUBLICATION

While it's a natural impulse to want to jump right into the more hands-on chapters of the Newscoop Cookbook, you'll save yourself a lot of time and trouble if you approach your Newscoop implementation project in the way we recommend here. These recommendations represent consensus among our community on best practices for Newscoop implementations.

One very important initial step is to understand the human element to the technology you're about to implement - who is involved in the project? You can benefit a lot from understanding who fulfils the following:

- Who makes decisions? Who makes decisions on the site project? Who makes decisions on coverage, articles and resource allocation?
- Who is the technical person? Who's the person in charge of the operation of the site, e-mail accounts and the like?
- Who has which role in the current workflow? How do things get done now, and who does it?
- What is the expected new workflow? How will things work for the staff after the new site project is finished?
- What are the current and expected visitation patterns for readers of your publication?

Being thorough at this stage helps to avoid making incorrect decisions or getting your decisions overturned at a later stage. In the worst case, you could end up with a dysfunctional design, and quite likely a dysfunctional project.

Draft a site specification that goes into as much detail as possible for the implementation, but remains flexible enough in the event of inevitable changes. This site specification, essentially the project's blueprint, should especially take into consideration the following:

WHEN IS THIS PROJECT CONSIDERED A SUCCESS?

What are the criteria for considering the project a success? For example, is the goal to increase page views or decrease the bounce rate? Is the goal to increase readership among a certain demographic or country?

Does the project do enough to make sure those goals are met?

Are the solutions you're proposing appropriate for the task at hand? If increasing reader time on site is a goal, does the site display enough "additional stories" links?

DO YOU KNOW ENOUGH ABOUT THE ORGANISATION?

Make sure you get as much information as possible on the following:

What do you know about your audience? Has market research been conducted? If so, what does it say? If your publication is using the Google Analytics tool, what does it say? What are the audience's needs, interests and technical capabilities? Needs and interests can also be things that the team can provide, but your users' technical capabilities (especially as delivered by Google Analytics) can help you determine the optimal design and level of interactivity for the project, affecting everything from font size to browser compatibility.

What is the functionality required? As much as you can, list and describe the functionality required in the project. Brief descriptions should suffice, but if you're thorough enough, you'll minimize the inevitable "Uh-oh, we forgot about <$fill_in_the_blank>" moments.

Which functionality is top priority? Which things are "nice-to-have," as opposed to "must-have?" When the success criteria are clear, the answers to these questions should get a lot clearer too. Limited budgets also work pretty well to force clearer prioritization.

Sitemap and features

Get or produce a site map of the existing content structure. What content does the site have, and how is it arranged? How often is the site currently updated? Is there content being created that is going underutilized?

For example, the current site may be a blog publishing 10 items per day, and because it's a blog, items are published in a reverse chronological order with newest articles first. But major stories are getting buried by newer, less-important entries. This would be something to address with the new site structure.

Discuss the proposed new site structure. Does the new site involve multiple languages? Issues and sections, or topics? How often will the site be updated? Will you create a new issue every day/week/month? How many sections will the new site have, and what are their names?

Subscriptions or no subscriptions?

If you plan to use subscriptions on your site, find out as much as you can about the subscription and paywall mechanisms involved. What are the revenue expectations for the new subscription-based site? What are those expectations based on (market research or a snowball's chance in hell)? Are there markets for the content that have not previously been considered? For example, many countries have significant diasporas interested in getting news from home, and they're often willing to pay for quality news. For more on this, see the chapter about contemplating *Subscriptions and revenue*.

Workflow

What is the content workflow? Who on the staff is responsible for which content? You'll find a lot more on this in the chapter on *Planning your workflow*.

Mashups, social media and other third-party services

What third party services are involved? List the intended integration with third-party services (if any). Social media, video, audio, comments and other widgets would count here. How complicated will the envisioned integration be?

This Cookbook includes examples and code snippets for Soundcloud, Disqus, YouTube, Vimeo, Flickr, Twitter, Facebook, Gravatar and others.

Custom development to set you apart

What additional custom development is required (if any)? Are you going to do any of this custom development yourself, or will there be another contractor to handle it? If your project requires custom development, it's very important to clearly specify not only the functionality required, but also the ways it will extend or interact with Newscoop's APIs. Also, it's helpful to work closely with the Newscoop core developers, so that they can provide advice and feedback, as well as including any plugins or add-ons in future Newscoop releases.

FINAL STEPS AND TIME TABLE

Once the draft of the specification is ready, have all team members go over it and comment, and then when everybody's OK with it, finalize it. If you're in an organization with one or more big bosses, get them to sign off on it. Then your team can make binding time/work estimates for the project.

While such a specification might seem like a lot of work (it usually is), it will give you a much better idea of the overall scope of the project, and will also help you to make a more realistic estimate of the time and labor involved.

7. PLANNING YOUR WORKFLOW

Newscoop users Nestan Tsetskhladze and Eteri Turadze of Netgazeti.ge in Batumi, Georgia. Photo by Douglas Arellanes (CC-BY)

Who is doing what? And when? What does it require and what depends on it? Who can do it? Who has to do it? And who signs off what's being done? All those questions can be grouped together into one concept: Workflow.

"Workflow" is one of those terms that gets discussed a lot among newsroom technology types, but workflow means many things to many people.

"A workflow consists of a sequence of connected steps. It is a depiction of a sequence of operations, declared as work of a person, a group of persons, an organization of staff, or one or more simple or complex mechanisms" is how Wikipedia describes it.

In Newscoop, workflow refers to the steps that must be taken in order to accomplish a certain task, usually related to publishing content on your site. Inside Newscoop there are three main milestones hardwired into the system which have proven to capture the essence of the article publishing process:

- New - the article has been generated, either by your staff or a citizen journalist
- Submitted - the article has been fact checked, images are added, the sub editor approves
- Published - either manually or automatically, the article goes live

You can fine tune this process in Newscoop with custom switches for Article Types. For more detailed technical information, please see the chapter on *Topics, switches and keywords to structure content*, as well as the chapter in the manual Newscoop 3 for Journalists and Editors titled *Publishing Articles*.

But publishing articles is only a small part of the overall workflow of running a site, in fact: Newscoop has built-in publishing automation to do the job for you, once you have lined up the articles. There are a lot more aspects that you will need to consider at the beginning of your site planning, such as:

What are the steps the editors will have to take in order to place an article in the top position on the site? The top position might be determined by most recently published, by having the editor click and drag items in the section, by assigning a topic to an article with a name of "top story", or maybe by attaching a custom switch. All of those approaches are valid and possible, but it's up to you (and your point of contact at the organization) to figure out what's best for their purposes.

Does the proposed workflow correspond to current staff levels? Does the functionality you're planning for the new site require a much larger staff? In other words, who's going to take and prepare all the photos for the very cool JQuery slideshow widget you want to implement? (What, you're a radio station and don't have too many photos in your archive?) Who's going to be in charge of monitoring site comments? Who's in charge of the overnight shift on the site? (Wait, you mean there's going to be an overnight shift?!?)

What are the issues that could lead to staff resistance to the project? While you may think that your slideshow widget is cool, the staff may say "this stupid slideshow widget means my workload is doubled!" The more you can anticipate issues with staffing and work with your point of contact to address them, the better off your project will be.

Who are the staff members capable of taking on the new roles your project introduces? For example, who are the avid social media advocates on the staff, and can they be brought in to take on the publication's social media tasks?

Where are the project's time and work savings for the staff, or will the project mean that everyone will have to stay late every day? Explaining time savings or additional burdens accurately and clearly will make you a lot of friends both on the staff and with the publisher.

Is your proposed workflow too complicated for non-technical staff? For example, when you have a dynamic page layout based on custom switches, are the steps clearly communicated to the staff, and are they clear on how to do it?

What are the ways your proposed workflow can fail? What can you do to simplify things without giving up basic functionality? Often, a proposed design and its accompanying workflow is too complicated for the staff to execute on a regular basis. If the staff can't do it, you'll need to make sure your design and approach doesn't fail.

KEEP AN OPEN MIND

Editors on tight deadlines love things that are simple to use. Well, as simple as possible, anyway. Such solutions never work on live sites!

Instead, try to solve undesired situations by fine-tuning your templates. Try to put yourself in the position of an editor, and predict the possible mistakes they'll make when they have to live with your work on a daily basis. Stay flexible and implement changes soon after the team agrees on them. A website is like a living organism; you'll have to look after it.

8. SUBSCRIPTIONS AND REVENUE

Newscoop provides the means to manage access to content with subscriptions. A subscription means that parts of your publication are only visible to users who are registered and logged in. By default, subscriptions are disabled - and planning your publication is completely unaffected by this chapter. Just proceed and make your publication. If you change your mind, come back to this chapter later and activate the Newscoop subscription system without changing your publication structure or content. If, on the other hand, you came to Newscoop because it provides an in-built subscription system, by all means, read on.

Because of the ongoing difficulties news organizations face in bringing in revenue, many have turned to subscriptions as a way of making money for their content. There's been a lot of heated debate about subscriptions or "paywalls" around content, but it's important to note that subscription management has been part of the code base in Newscoop since 1999, and has represented a significant source of revenue for many media organizations.

It's also important to point out that subscriptions do not necessarily have to be paid. You may decide that your publication will require that visitors share certain demographic information in return for site access, or you may decide that certain areas of the site are for members only. Or perhaps membership is sponsored by an advertiser, who provides free site access in return for the opportunity to promote their product to subscribers.

Newscoop flexibly supports subscription models. This allows you to switch from free to paid services and back again. Users can register themselves on your publication. The payments made for subscriptions need to be handled by your staff and applied to user profiles. Newscoop currently does not support fully automated transactions, where, for example, a user signs up for a subscription, pays by credit card and then immediately has site access.

ACCESS CONTROL FOR CONTENT AND FEATURES

Newscoop provides a fine-grained way to manage content in relation to subscribers and non-subscribers. You can set content to be subscriber-only:

- For all content in a publication
- For all content in a specific section
- For all content in a specific language
- On an article-by-article basis
- For an IP address range (useful when selling corporate subscriptions or to libraries)

If a reader ventures into an area on your site where they do not have access, you can manage in your templates what they will get to see - or what they will be able to do (like writing comments). Newscoop is very flexible in changing this access on the template level. In another chapter of this Cookbook, you will find the nitty-gritty of templating for subscriber pages. Also, below you will find details of how to administer subscriptions in the administration interface.

When preparing your publication's site map, it's a very good idea to map out which parts of the site - if any - will be visible to subscribers only. It's also important to try to figure out how much of each article you want to display to non-subscribers. According to these decisions you might also plan the fields in an article - there could be a part of the article which is only available to subscribers, like the phone number in classifieds, while everything else is available to all readers.

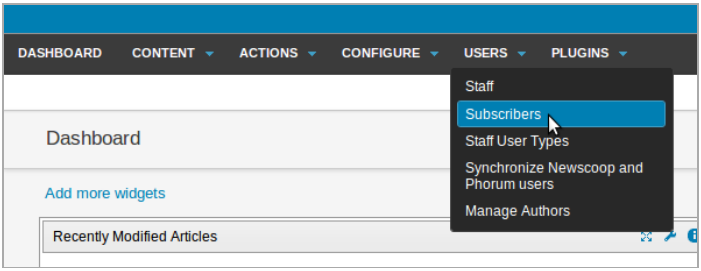
For our purposes as site implementers, here are some things to consider:

- Is the entire publication for subscribers only?
- Which sections are for subscribers only?
- Which pages are free to non-subscribers? The front page? The section page?
- How much of the article is free to non-subscribers?
- What will your page "sorry, this page is for subscribers only" say, and how will it look?
- What information will you require from your subscribers?

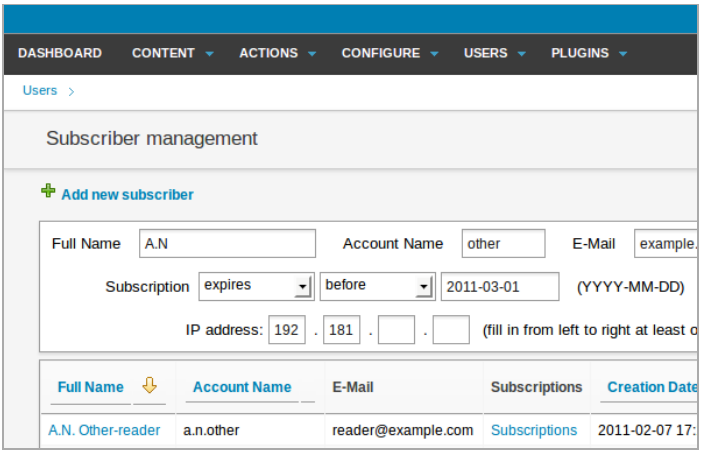
Once you do work out your site's subscription strategy, you should read the chapter about *Subscriptions, registration and content access* in this Cookbook for specific templating approaches.

MANAGING SUBSCRIPTIONS IN THE ADMIN INTERFACE

The list of subscribers is available in the Newscoop administration interface by clicking on Users in the main navigation menu, then Subscribers:



The subscription management interface includes a search tool which enables you to list readers according to several criteria, including subscription expiry date:



The duration of paid or trial subscriptions is determined by the default time periods set when you created the publication. (You can change these settings for your publication by clicking the Configure icon in the Publication list). During a trial period, the reader has a chance to make a payment, and consequently to have their subscription extended by an administrator.

The publication's administrator is automatically notified by e-mail of all new subscriptions. Subscribers also receive automatically generated e-mail reminders that their subscription will expire in two weeks. The text of the subscription expiry message is created in a Newscoop template. The default system template reads as follows:

```
"Dear <reader>,

This is an automatically generated e-mail message. Your <subs_type>
subscription (started on <start_date>) to publication <publication> will
expire on <date> (in <number> days).

Please enter the site http://<pub_site> to update subscription."
```

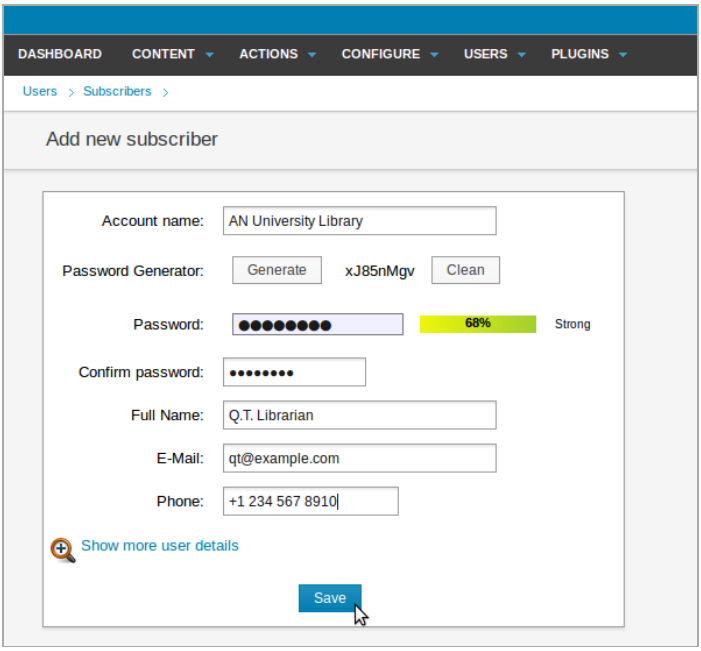
This file is located in the system_templates folder and is named _subscription_notifier.tpl. It can and should be customized to best suit your needs and style.

IP address access

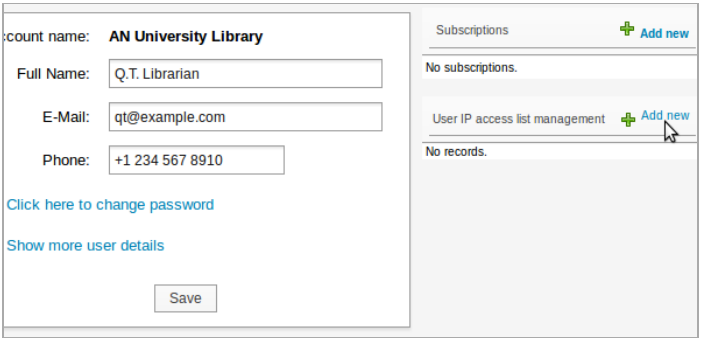
Publishers often sell corporate or institutional subscriptions, and Newscoop has support for these out of the box. Newscoop's IP based access feature can be used to automatically log in subscribers that have IP (Internet Protocol) addresses in a known range. This means that groups of readers with computers within that IP range do not need account names or passwords to access your subscription-only content.

To create an IP Address subscription, go to the subscriber management page in the administration interface, in the drop-down menu under Users -> Subscribers.

First, create a subscriber account by clicking on the 'Add new subscriber' link, next to the green plus sign icon. Enter an account name, a password, and a full name with an e-mail address and phone number for a contact at the subscribing group or institution. You do not need to inform individual readers using IP address access of the existence of this account name and password; it's only for the subscriber contact.

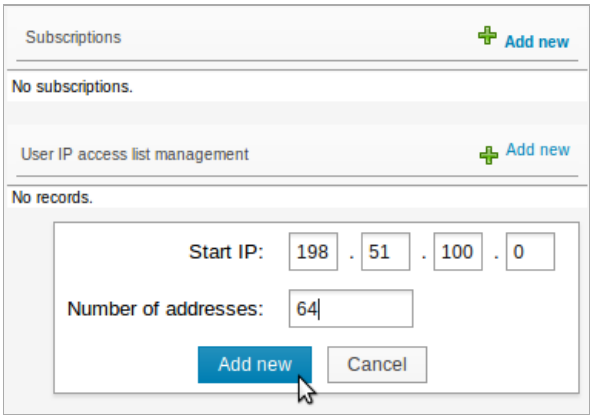


Once the new subscription account has been created, click on the **Add new** link in the User IP access list management table on the right hand side of the page.



The form allows the input of a range of consecutive IP addresses. If necessary, more ranges can be added later. The start IP address is the lowest address in the range, while the number of addresses sets the size of the range, including any address ending in zero. For example, if a university library had the range of IP addresses 198.51.100.0 to 198.51.100.63, then you would enter:

- start IP address: 198.51.100.0
- number of addresses: 64



It is not necessary to add anything to the Newscoop templates for IP based subscriptions to work.

It is also possible to search for subscribers based on known IP addresses. In the search box on the Subscriber management page, enter a number in at least one box in the field labelled IP address, starting on the left side. Click the **Search** button, and the page will display all subscribers that have known IP addresses which match that number (or numbers).

ADVERTISING AND NEWSCOOP

The other side of the revenue equation for most online publishers is advertising. For this purpose, we recommend the robust and open source OpenX system http://www.openx.org

Ad insert codes - whether they are served from your own OpenX server, a banner ad display network or a service like Google AdWords - can be included in any template or sub-template, so you have complete flexibility in both placement and size.

You can also build your own templates to manage your banners. All three options are explored in greater detail in the chapter *Working with advertising*.

9. STRUCTURING YOUR CONTENT

Newscoop's built-in logic for organizing content follows a logical and hierarchical structure derived from print publications. The following structure is built in by default:

- Publication, which consists of chronologically ordered issues
- Issue, which consists of sections
- Sections, which store articles
- Articles, which can be of different Types
- Article Types, which you can build specifically for your publication

You can overrule this structure and change this hierarchy to fit your publication. For example, you could use issues as containers for storing the site's substructure (without chronological importance, for example). Or you could have only one issue with lots of sections. Some situations are best handled with multiple publications, while others might work best with an emphasis on Topics, not Sections.

How you organize your content usually depends on:

- The nature of the content being published
- The workflow inside your team
- The amount of content to be published
- The rate new content is added

EXAMPLE: MAGAZINE

Let's imagine the website of an existing weekly print magazine. In this Cookbook, we often use the template pack "The Journal" which reflects magazine structure. "The Journal" covers topics like politics, business, science and technology, health, entertainment and sports. In such a situation, it's logical to use the built-in publication > issue > section > article structure in a straightforward way.

In Newscoop, you don't need to create a new issue from scratch every time - just select 'Add new issue' and then 'Use the structure of the previous issue' - you'll get an empty copy of the previous issue with all of its sections, settings and assigned templates.

"The Journal" can be seen at http://journal.templates.sourcefabric.org/

And another example, "Newsport", can be seen at
http://newsport.templates.sourcefabric.org/

EXAMPLE: RADIO STATION SITE

Newscoop can be used to power a radio station website, where content is bilingual, and where the 'publication' only has one issue. Each radio show has a section, and individual articles can be added into this section when a new episode of the show has been broadcast. Breaking news items are presented on the front page.

Newscoop's Soundcloud plugin can be used to upload individual audio clips and attach them to news articles. Recordings of entire shows can be uploaded and displayed on show articles. In addition, Sourcefabric's Airtime broadcast software can be used to automatically record live shows and upload them to Soundcloud.

An example of this approach can be seen at West Africa Democracy Radio's site at http://www.wadr.org.

EXAMPLE: BLOG

Newscoop can be used as a blogging platform, which is especially suitable for group blogs or blogs with multiple authors and sections. For this purpose you might choose a structure where issues are months, or you could have yearly issues. The website front-end doesn't have to reflect the issue structure, as Newscoop can generate the listing of articles regardless of where they are in the structure; for example, ordering them by publication date.

Inside the blog, you can structure your content with Newscoop Topics. Topics are flexible and could either be used like tags, or in a hierarchical structure like categories or a taxonomy. Find out more about topics later in this manual.

An example of how Newscoop can be used as a high-end multi-lingual blogging platform can be seen in "The Journalist" template from Sourcefabric. The online demo is at http://journalist.templates.sourcefabric.org/

Any other structure logic can be developed and used with Newscoop. It's really a matter of good planning, and analysis of the factors discussed at the beginning of this chapter.

FROM DESIGN TO TEMPLATES

10. THE POWER OF $GIMME
11. BEST PRACTICES FOR DESIGNING YOUR CONTENT
12. PREPARING DESIGNS FOR TEMPLATES
13. ORGANISING TEMPLATE FILES
14. TIPS AND TRICKS FOR THE TEMPLATE NOVICE
15. INSTALLING TEMPLATE PACKS

10. THE POWER OF $GIMME

Now that you're venturing into the territory of Newscoop, $gimme serves as your map and compass. $gimme is your reliable friend, your eager spy and the keeper of all knowledge. $gimme will give you information and answer your questions. "Give me the name of the article", "Give me the number of the issue", are requests $gimme will follow without even a shrug. Obviously, "give me" is how $gimme got its name. And $gimme speaks a simple language, along the lines of "gimme publication name" or "gimme user email."

$GIMME AND THE NEWSCOOP TEMPLATE LANGUAGE

We like to refer to Newscoop's template language as a programming language for news. $gimme lets you list and arrange articles, display text and multimedia, prepare content for third-party services and include external content. It allows editors total design freedom and doesn't push them into a single way of presenting their stories.

EASY TO USE, EASY TO READ

The philosophy behind the Newscoop template language (and $gimme) has been to make it as intuitive and easy to use as possible. Here's how you would use $gimme to display the name (or headline) of an article, like "Newscoop 3 Cookbook now in shops":

```
{{ $gimme->article->name }}
```

Here's how you would use $gimme to display a section's description, such as "Film Reviews from the Venice Film Festival"

```
{{ $gimme->section->description }}
```

Display the date when an issue was published:

```
{{ $gimme->issue->publish_date }}
```

Display the file name of an article attachment:

```
{{ $gimme->article->attachment->file_name }}
```

Display the caption of an image, like "Work started at Berlin's new airport":

```
{{ $gimme->image->caption }}
```

Display the map provider you're using for your site's base maps:

```
{{ $gimme->map->provider }}
```

Here are some more examples along the same lines:

```
1. {{ $gimme->article->keywords }}
2. {{ $gimme->article->comment->subject }}
3. {{ $gimme->publication->default_language }}
4. {{ $gimme->author->name }}
5. {{ $gimme->author->biography->text }}
6. {{ $gimme->image->photographer }}
7. {{ $gimme->user->name }}
```

```
8. {{ $gimme->user->phone }}
```

And there's more. In the following chapters you'll see $gimme in action, providing all kinds of information added by editors and journalists.

SEPARATION OF PRESENTATION AND CONTENT

Web designers use the concept of "separation of presentation and content" every day when developing in HTML and CSS. In the HTML you can emphasize parts of the content with the **em** element, so whatever is wrapped inside tags is "what" needs to be emphasized. "How" the emphasis is being displayed can be set in the CSS file, assigning font family, size, style, color and so on.

Develop your design without touching any code

The Newscoop template engine gives you maximum flexibility to develop templates in HTML, CSS and JavaScript any way you like. You don't need to think in terms of blocks or widgets, and you can use sample text like "Lorem ipsum" when fine-tuning your layout. Turning your design into a template for Newscoop means to replace the sample text with $gimme. Here is a simple example. Here's the HTML we want to serve:

```
1. <div>Media and Journalism</div>
2. <h2>Newscoop Cookbook released</h2>
3. Anything web developers need to know about the CMS for professional journ
   alists in one comprehensive desktop reference
```

Separating content from presentation in the Newscoop template engine, we call in $gimme to give us what we need:

```
1. <div>{{ $gimme->section->name }}</div>
2. <h2>{{ $gimme->article->name }}</h2>
3. {{ $gimme->article->subtitle }}
```

Note that $gimme->article->name is wrapped in two curly brackets on either side. There's also a white space between the brackets on either side - while it's not mandatory, it's a good practice to make the template code more readable - but there are no whitespaces in $gimme->article->name. The template engine operates on the simple premise that it finds two opening and closing brackets, takes out anything in between and tries to make sense of it, replacing it with content from the database.

Once the logic is in place, the design can be changed any way you want:

```
1. <div class="section">{{ $gimme->section->name }}</div>
2. <h2 class="title">{{ $gimme->article->name }}</h2>
3. <span>{{ $gimme->article->subtitle }}</span>
```

Or you can add even more material to control the design with CSS later:

```
1. <div class="wrapper">
2.     <h2 class="title section{{ $gimme->section->number }}">
3.        {{ $gimme->article->name }}</h2>
4.     <span>{{ $gimme->article->subtitle }}</span>
5.     <div class="section">{{ $gimme->section->name }}</div>
6. </div>
```

Note how in the <h2> tag, a class is being added here that will change its value depending on the section the article is coming from. The interface designer can develop the look and feel of the site without having to go back to the programmer. Instead, all that needs to be changed is the template.

WATCH $GIMME AT WORK

On its own, $gimme can pull information out of the content database and place it inside your template. When you set it loose inside Newscoop templates, it advances to become the project leader of your publication; beyond simple retrievals and replacements, $gimme is then in charge of making design decisions. Checking information with $gimme can create the most complex designs in very few lines of code.

To put $gimme into action, you'll use functions from the Newscoop template language. These functions are developed with the news organisation in mind, delivering publication content with simple commands. The functions available are being expanded continuously alongside new features in Newscoop, and an expanding range of features for already existing functionality. A complete Newscoop template design reference is included at the end of this Cookbook.

Now let's take a peek into an actual code snippet using $gimme. In this case you'll see a Newscoop function alongside $gimme. The function is called list_article_authors. Everything inside this function (it has a closing tag like in HTML) is repeated as many times as there are authors who have worked on this article.

The following example - in plain English - would be like asking $gimme: "Could you please list all the authors that worked on this article?"

Lets look at the code step by step and keep telling $gimme what to do.

```
1. <ul>
2. {{ list_article_authors }}
3.     <li style="clear: left;">
```

For each author, we want the last name and then the first name separated by a comma.

```
<strong>{{ $gimme->author->last_name }}</strong>,
{{ $gimme->author->first_name }}<br />
```

Also show their image.

```
<img src="{{ $gimme->author->picture->imageurl }}" align="left" width="66px" />
```

Very important, show their full name alongside their work field (author, photographer, etc.), followed by their e-mail and their biography.

```
1. {{ $gimme->author->name }} ({{ $gimme->author->type }}):
2. <a href="mailto:{{ $gimme->author->email }}">
3.    {{ $gimme->author->email }}</a>
4. {{ $gimme->author->biography->text }}
```

When you are done, close this list. Thanks.

```
1.     </li>
2. {{ /list_article_authors }}
3. </ul>
```

You can see how $gimme makes it easy to pull up information in your templates. There is a quite a lot happening in twelve lines of code. We are delivering a list of authors alongside images, biography and e-mail. This will look great in the header of the article.

After this very straightforward example, we'll leap ahead into using maps inside articles (covered in detail later in this Cookbook). The following code will list the name and geo-location information for an article. Note that only "enabled" locations are listed, because the journalist might still be working on the map in the administration interface. Once a location is set to be seen by the public, it shows up using this code:

```
1. Location(s):
2. {{ list_article_locations }}
3.     {{ if $gimme->location->enabled }}
4.         <p>
5.         {{ $gimme->location->name }}<br />
6.         {{ $gimme->location->longitude }}, {{ $gimme->location->latitude
   }}
7.         </p>
8.     {{ /if }}
9. {{ /list_article_locations }}
```

We bet you've already started loving the power of $gimme. Besides $gimme, the Newscoop template language includes a complete set of functions and several other powerful capabilities that are condensed in the Newscoop Template Reference.

THERE'S MORE...

The Newscoop template language is an extension to a popular and powerful template engine called Smarty, which itself is written in PHP. Smarty is popular among PHP users and developers, and comes with a rich set of functionality you can use in your templates. You will see some examples in this cookbook and find a full reference online at http://www.smarty.net

With every new Newscoop version the template language and $gimme become more powerful. We keep developing and integrating new features, and extending the current ones, but we do also care about backward compatibility, which means we are not going to break your current website.

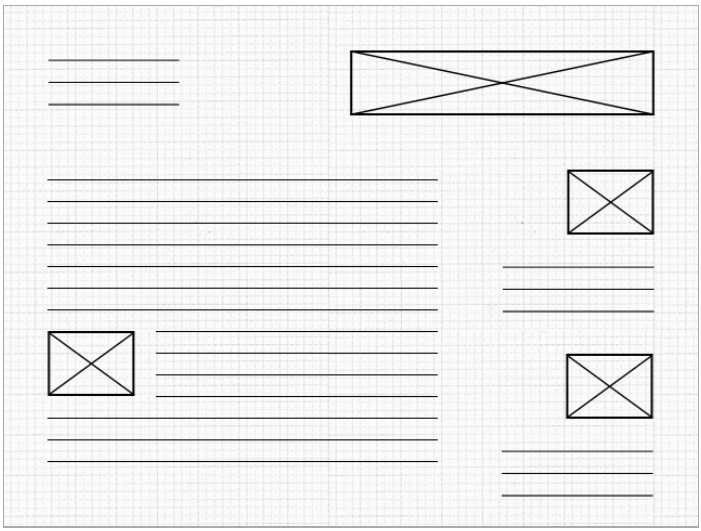
Once you understand the power of $gimme, you will start getting a sense of the freedom it offers you when building websites. Enjoy!

11. BEST PRACTICES FOR DESIGNING YOUR CONTENT

Publications can have multiple types of content: editorial copy, interviews, graphics, photos, audio and video recordings, and other digital assets. When designing pages for these types of content, you should keep focussed on the vision and style you want to convey. The importance of good design cannot be valued highly enough, because it creates the readers' emotional response to your publication. This has been proven on numerous occasions where the redesign process dramatically changed a site's traffic, even though the content remained the same.

It will be helpful to keep the potential of the Newscoop template engine in mind when creating your site's design. As a designer, this allows you to not think in boxes, but to think outside the box.

In this chapter, we want to lay down some general guidelines helping you to deliver the best structure possible for your design. This can be seen as a number of steps that a template designer should follow, like defining the grid of the template, and designing sections and pages based on that grid.



The diagram above shows you how to structure the design and also provide a good experience for your readers. Please notice the white space between sections; it can provide room to breathe, and also make clear distinctions between sections.

THE TEMPLATE GRID AND THE GOLDEN RATIO

Typical desktop and laptop computer displays have a resolution of at least 1024x768 pixels. 960 pixels is very close to the minimum limit of monitor resolution and is also divisible by 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 32, 40, 48, 60, 64, 80, 96, 120, 160, 192, 240, 320 and 480. This makes it a highly flexible base number to work with.

With Newscoop and with most news sites we recommend using a template that is 960 pixels wide. Fluid templates can provide more flexibility, but they defy the purpose of having a grid. A 960-pixel-wide grid (http://960.gs/) can help you divide your design structure into Newscoop's template elements like header, article area, sidebar and footer. This grid can also provide enough space for photographs and videos, things most media outlets use in great quantities.

An important part of defining the template grid is using the golden ratio (http://en.wikipedia.org/wiki/Golden_ratio). The value of the golden ratio is 1.618, but approximations can also be used, such as a ratio of 5:3. The golden ratio is most often found in nature. Nautilus shells are a perfect example; the spirals get smaller and smaller in the same proportion to each other.

Using the golden ratio can provide your readers with a clear perspective over your content. To give you a good example of how the golden ratio can be used for template design, you can set the content area to 600 pixels wide and the sidebar to 360 pixels wide, so that the ratio between the content area and sidebar is pretty close to 1.6 (600 divided by 360).

Aesthetics can be measured and more importantly can be constructed, even built from the ground up. Following a few guidelines can help you make sure you're on the right path to having a publication that not only is compositionally balanced, but also aesthetically pleasing, so that people can enjoy reading each article or part of the site. Designing a template grid doesn't have to be exact math, it's more of an experimentation of ratios and white space along with text and images.

ARTICLE DESIGN

When putting your design into HTML, you should follow web standards and the common uses of HTML tags. This is important to make your content machine-readable, which will enable search engines to understand your content and rank it higher.

The use of headings (H1, H2, etc.) for titles and subtitles, DT for dialogs, CITE for citations, BLOCKQUOTE for quotations can be seen as limited, but they can provide a very clear definition of the elements that should be designed and then styled in CSS. Usually H1 is kept for the site title, which is positioned in the header area. Remaining headings are used for article titles and subtitles of the other parts of a story.

A good way to design the article content would be using only the inline elements mentioned above and also paragraphs for the text, keeping block tags like DIV only for the presentation layer. You'll find more information on the use of tags and their importance in the chapter titled *Search Engine Optimisation (SEO)*.

The template grid defines what goes inside an article and what goes outside of it. The same conventions established for the site's design can also be used for article design; don't try to reinvent the wheel. The ratios, subdivisions and modularizations should be the same across the entire site.

COMMENTS DESIGN

Comments are not only the feedback received for a news story, but also the conversation of the community behind your site. When you design a comments structure you should take into account that people are not only communicating their ideas about the article, but also communicating with each other. Threaded comments reflect this dialogue very well. Adding profile pictures so that each reader has his/her own picture shown next to the comment is also a good idea.

When designing comments and comment forms, you do not have to follow and modify existing HTML structures. Because of Newscoop's advanced templating system, comment submission forms can be designed and styled completely using HTML and CSS.

There's a particular chapter in this manual titled *Profile pictures: Gravatar, Facebook, Twitter* where you can find more details about how users can set up profile pictures and display their pictures along with their comments.

DESIGNING THE OTHER PARTS OF A NEWSCOOP SITE

Newscoop's header area is the part where the masthead and the main navigation take place. The masthead is your publication's logo or title. It should be the first thing your reader sees. It must be legible, tell people about your publication's intent and convey what they can expect. It's important to remember that while you may have looked at your own publication thousands of times, your readers may have not.

Archive design must have a clear structure, so the reader can quickly identify the article they are looking for. Usually it is shown in reverse chronological order, grouped by years, months or weeks. In Newscoop you can also use calendars to design your archives. More about this can be found in the chapter *Calendars with jQuery*.

A well-designed search results page can provide readers with a pleasant experience, so they won't be afraid to look for articles published in the past. This manual also has a chapter titled *Search Templates*, where you can find more information on getting the most out of search results pages. Don't forget that your search results page should display not only the article's name, but also small pieces of supporting information about the articles (developers like to call this "metadata"), like the date it was published, the author's name and the topics assigned to the article.

There's more information about the structure of other parts of Newscoop sites in the chapter *Understanding Templates, HTML & CSS*.

TYPOGRAPHY

Newscoop allows total control over font usage; you can even embed font faces using CSS3. Actually, one of the important parts of the design is not the choice of a single font, but all of the site's typography as a whole.

A principal part of typography is setting the line height. As mentioned before, line height is a good place to make use of the golden ratio. A value of 1.6 em should work just fine, but the recommended value is 1.5 em for the kinds of content mentioned in the beginning of this chapter.

An example of good typography for web design is to set the font size to 13 pixels and the line height to 21 pixels. The ratio between line height and font size is very close to the golden ratio.

Using too many font faces might have major impact on the professionalism a publication conveys. Displaying fewer font faces might be less confusing for the reader. Consider using one font face for titles and a different one for article content.

```
<link href="http://fonts.googleapis.com/css?family=Droid+Sans" rel="stylesheet"
type="text/css" />
```

The above line of code is an example of how to use Google Web Fonts for embedding new font faces in your design. There are several services that can provide this functionality; some are paid services like TypeKit, and some are available for free like Google Web Fonts. These services take care of providing the right font format for each browser. Some browsers are using standard font formats like OTF or WOFF (Web Open Font Format), while many mobile browsers use technologies like SVG for displaying font faces.

12. PREPARING DESIGNS FOR TEMPLATES

One way to design for Newscoop templates involves slicing a static mock-up of your publication into multiple images. A graphics application such as GIMP or Adobe Photoshop can be used for this task. The resulting images can then be re-created using HTML and CSS. The main advantage of slicing up and re-creating the design is that it reduces the bandwidth load for your site, which means increases in your readership will scale better, and your readers will get the pages faster. For instance, design elements such as horizontal rules and colored backgrounds can be styled using CSS, and therefore do not need to be served as image files.

Let's look at the sample Template Package "The Journal".

Front page of the sample Template Package "The Journal".

Before you begin the slicing process, you need to identify the main structural elements of the design, so that you know how to arrange and structure your HTML files. Dividing the design should follow the grid established in the previous chapter. The main divisions should end up being tags. A is a block containing images and text that can be positioned using CSS.

An important thing to remember when creating HTML files is that you should use IDs for tags that are present only once in your site pages (e.g. navigation, sidebar, footer, etc.). You should use classes for tags that are repeated more than once on the same page (e.g. links, pages, etc.)

DIVIDING THE FRONT PAGE TEMPLATE

The diagram below shows how the design for "The Journal" front page template is divided:

Your HTML files will end up being .tpl files, which are Newscoop template files. For "The Journal", the front page template **front.tpl** is a skeleton which includes and positions the following sub-templates:

- **_html-head.tpl** This is not visible in the diagram above, but it is an important template used on all site pages. It defines the 'head' tag section of HTML pages.
- **top.tpl** Includes all templates with prefix 'top', and creates remaining elements of the header section of the page (date, RSS link, logo). This template is repeated on all site pages.

- **top-login.tpl** Template used to generate the login form and register link. If a reader is logged in, it makes links to the reader's account page and to logout.
- **top-highlights.tpl** Lists the three newest articles with the switch 'highlight' switched on.
- **top-recent-entries.tpl** Lists the three last published news articles.
- **top-search-box.tpl** Generates the search form.
- **top-nav.tpl** Used for site-wide navigation.
- **front-top-story.tpl** Displays just the newest article with the switch 'On front page is on'.
- **front-tabs.tpl** Makes tabs with most-read and most-commented articles, and then lists the actual articles.
- **front-three-categories.tpl** Lists three articles (regardless of section) which have the switch 'On section page is on'.
- **front-also-categories.tpl** Makes a horizontally-scrollable list of 12 articles that don't have any special switch on.
- **front-bottom-about.tpl** Template with hard-coded 'About this site' content.
- **front-bottom-meta.tpl** Makes hard-coded links to important pages that are used for Newscoop administration, such as the administrator login.
- **front-bottom-poll.tpl** Displays the latest defined poll.
- **footer.tpl** This is the part of every page that contains links to 'about' pages and legal information.

DIVIDING THE ARTICLE AND SECTION TEMPLATES

The diagram below shows how article and section page designs can be divided into sub-templates. The **top.tpl** and **footer.tpl** sub-templates are re-used from the front page design, but in between these are different sub-templates. Some of these sub-templates may have been created for just one specific purpose, while others will be re-used again and again.

13. ORGANISING TEMPLATE FILES

Every template package available for download from Sourcefabric has three main templates:

- Front page
- Section page
- Article page

These are called - no surprise here - front.tpl, section.tpl and article.tpl.

You could run a publication with these three templates and nothing else. But to make the most of the template engine's power, you should divide up your templates further, creating sub-templates which can be easily reused and edited.

There are some parts of pages which are identical, regardless of the section or the article. This is usually the case for the header, navigation and footer, for example. But it could also be an image stream from flickr, a Twitter feed, a 'Like' button from Facebook, or a similar icon for sharing content from a third party provider.

You don't have to duplicate this part of the template. In fact, it's even better to hand off these functions to sub-templates. Then when you apply a change to the sub-template, it will show up across your entire site.

You can take out sub-templates, place them in a subfolder and call them with an include, like this:

```
{{ include file="set_thejournal/_tpl/article-comments.tpl" }}
```

As you can see in this example, the path to the template which is being included needs to be named. Planning your templates' folder structure carefully will save you time later, especially when you run more than one publication on Newscoop and have several different designs in action.

We encourage you to follow the folder structure described below. It's not set in stone; you can structure your templates and folders however you prefer, but the Newscoop community has agreed on this structure after years of working with templates. We consider it to be best practice. Also, the template packs Sourcefabric provides all feature this structure. If you use the same logic it will be easier to use examples or copy sub-templates from one template package to another. Read on to find out what we recommend when it comes to naming templates and placing them in your file system.

TEMPLATE NAMING

Besides the main templates (front.tpl, section.tpl, article.tpl) you will probably have many sub-templates to run your publication. Depending on whether they're common across multiple pages, or unique to a page, you may follow these rules. But you don't have to, because Newscoop doesn't have reserved names for specific templates; it's really up to you to create your environment the way you like.

1. _html-head.tpl is the template we provide for all links, meta tags and information globally needed for your publication, like jQuery, webfonts, CSS files and universal JavaScript.

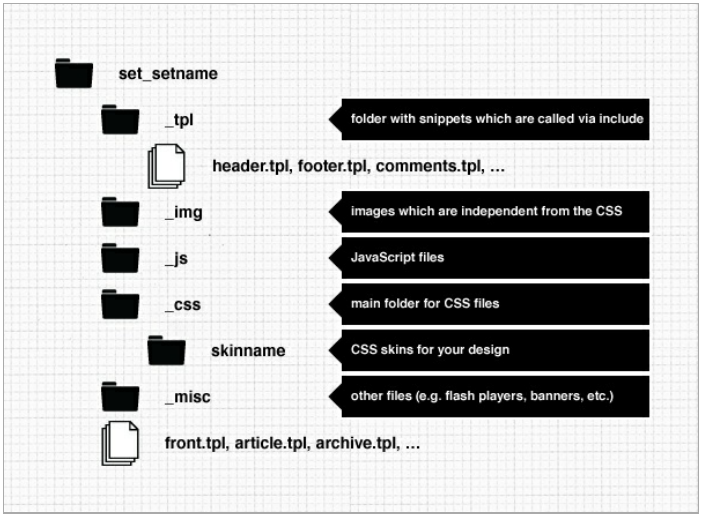
2. Unique code for sections or articles can be called from _html-head.tpl using {{ if $gimme->template->name == "path/to/article.tpl" }} or with other IF conditions like switches, topics or keywords. Such additional includes could be slideshows or special CSS files. For quick development, having one template (for example _html-head.tpl) saves you lot of time when you have to introduce some changes, and keeps things consistent. You can always chop it up later.
3. Includes that are used on more/all pages can have short names, like header.tpl, main-nav.tpl, footer.tpl
4. Included templates that are specific for a single page (or a few pages from the same context) can have a prefix, like article-sidebar.tpl or front-top-story.tpl. This way you will have templates for article page grouped together in the template file listing. The same goes for archive, front page, search and so on.
5. Special templates that are not directly used to output content on the web page but to define context (for example the _html-head.tpl which defines the HEAD section of web pages, or some RSS or sitemap generators) can have the '_' (underscore) sign as the first character; this way, all such templates are grouped together on the top of the template listing page.

FOLDER STRUCTURE

Your templates will always be inside the filesystem path:

```
/path/to/documentroot/templates/
```

These files are accessible through the template editor of the Newscoop administration interface. Inside this folder you can add sub-folders. We recommend the following structure:



Here's some practical advice on how to make your life easier with a clean and functional folder structure:

1. templates/**set_name/**
   Replace "name" with the name of your template package. For example, we are using "set_thejournal" for the template package "The Journal". The main templates are placed inside this folder, like front.tpl, section.tpl, article.tpl or universally used templates like search.tpl, archive.tpl, gallery.tpl etc. The include path is:
   {{ include file="set_thejournal/gallery.tpl" }}
2. templates/**set_name/_tpl/**
   All your sub-templates are collected inside this sub-folder. In our experience it isn't necessary to divide this into further sub-folders. Instead you should use the name of sub-templates for further clustering, like article-comments.tpl, article-author.tpl, etc. The include path is:
   {{ include file="set_thejournal/_tpl/article-comments.tpl" }}
3. templates/**set_name/_img/**
   If you need images on the file system for your publication, place them here. The source path is:
   <img src="http://{{ $gimme->publication->site }}/templates/set_name/_img/logo.png" />
4. templates/**set_name/_js/**
   Place all the JavaScript files you need in this folder. They are included like this in the header:
   <script type="text/javascript" src="http://{{ $gimme->publication->site }}/templates/set_thejournal/_js/tabs.js"></script>
5. templates/**set_name/_css/**
   You can place universal CSS files into this folder, things like grid, or font sizes. Alternatively you can use a subfolder (see next point). These CSS files are included like this:
   <link href="http://{{ $gimme->publication->site }}/templates/set_thejournal/_css/default.css" rel="stylesheet" type="text/css" />
6. templates/**set_name/_css/skinname/**
   For the CSS files, we decided to introduce a subfolder, which is best being described as a skin. For example, the CSS in this folder could change the colour of your site, in which case you might have folders like "red", "blue" and "morning mist" in here. You call these files with:
   <link href="http://{{ $gimme->publication->site }}/templates/set_thejournal/_css/green/green-skin.css" rel="stylesheet" type="text/css" />
7. templates/**set_name/_misc/**
   Use this folder for additional code, like a Flash Player which is being embedded in your templates. The URL path is:
   http://{{ $gimme->publication->site }}/templates/set_thejournal/_misc/
   If you need the folder path it would be:
   /path_to_document_root/templates/set_thejournal/_misc/

If you check out our template packages you will see that beyond this strict structure, we actually get a bit looser and let our hair down. For example, you'll find image folders inside the CSS folders, providing background images. A good rule of thumb would be keep your ship in order for as long as you can. Then get flexible and sail with the wind.

14. TIPS AND TRICKS FOR THE TEMPLATE NOVICE

This chapter is a collection of snippets offering a quick overview of the inclusion of files, modification of strings, working with date and time, and managing line breaks. The following chapters will bring these snippets to life. For more modifications, you can consult the http://www.smarty.net documentation.

INCLUDING FILES

Templates and sub-templates are included using the template path starting inside the *templates* folder:

```
{{ include file="set_thejournal/_tpl/article-comments.tpl" }}
```

DATE AND TIME

For full date and time formatting options, see the template reference at the end of this manual.

Print **current time** (e.g. format: 25 April 2011, 16:20:45)

```
{{$smarty.now|camp_date_format:"%e %M %Y, %H:%i:%S"}}
```

Print **date and time of publication** of an article

```
{{ $gimme->article->publish_date|camp_date_format:"%e %M %Y, %H:%i:%S" }}
```

MODIFYING STRINGS

Stripping tags from WYSIWYG content

```
{{ $gimme->article->intro|strip_tags }}
```

Truncate string to specific length

```
{{ $gimme->article->intro|truncate:70 }}
```

Escaping HTML and stripping tags (e.g. for meta description)

```
{{ $gimme->article->intro|strip_tags|escape:'html' }}
```

Upper and **lower** case

```
{{ $gimme->article->name|upper }}
{{ $gimme->article->name|lower }}
```

Replacing and chopping strings

Replacing parts of the string (example: whitespace into underscore)

```
{{ $gimme->article->name|replace:' ':'_' }}
```

You can also use a **regular expression** with regex_replace

```
{{ $gimme->browser->moz_data.2|regex_replace:"/\./":"-" }}
```

|regex_replace:"/\./":"-" replaces all dots with dashes. This can be useful in JavaScript elements. The following line uses regex_replace to replace fancy quotes like » « ” and so on with "

```
{{$gimme->article->name|regex_replace:'/&(.*?)quo;/':'&quot;'}}
```

Trim a string (chop off whitespace at the beginning and end)

```
{{ $gimme->article->seo_title|trim }}
```

COUNTING AND STATISTICS

Counting characters, words, sentences and paragraphs:

```
Headline "{{ $gimme->article->name }}" has
{{ $gimme->article->name|count_characters }} characters and
{{ $gimme->article->name|count_words }} words.
The full article has
{{ $gimme->article->full_text|count_sentences }} sentences and
{{ $gimme->article->full_text|count_paragraphs }} paragraphs.
```

USEFUL CONDITIONS

Checking **if variable is empty**

```
{{ if $gimme->article->seo_title|strip_tags|trim !== "" }}
```

Using the if condition with these string modifiers makes sure that no HTML tags or whitespace is in the field.

AVOIDING LINE BREAKS

Anything within {{strip}} {{/strip}} tags is stripped of extra spaces or carriage returns at the beginnings and ends of the lines before they are displayed. Let's say you had a publication where the value of the seo_title field often contained leading spaces, sometimes editors even get HTML tags into the field:

```
<strong>Newscoop</strong> Rocks!
```

The following example will output a single line starting and ending with quotes, but without leading spaces, like "STARTNewscoop Rocks!END". Any HTML code will be stripped out.

```
"{{ strip }}START
{{ if $teststring|strip_tags|trim !== "" }}
  {{ $teststring|strip_tags|trim }}
{{ /if }}END
{{ /strip }}"
```

If you want to keep whitespace between elements, use a workaround with:

```
 {{ textformat wrap=200 }}
```

This will turn everything into a single line with a space between each element, and applying a line break every 200 characters. Adjust the value of 200 to meet your needs. The example below will list various values in one line inside the body tag:

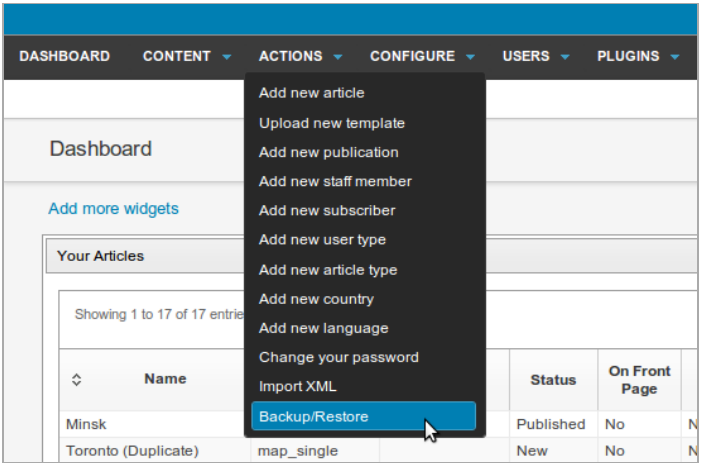
```
<body class="{{ textformat wrap=200 }}
{{ $gimme->browser->browser_working }}
{{ $gimme->browser->ua_type }}
{{ $gimme->browser }}
{{ /textformat }}" >
```

15. INSTALLING TEMPLATE PACKS

If you want to see a final result of the process of chopping up a design, the easiest way is to download one of the template packs for Newscoop from Sourcefabric. Each template pack contains a complete sample publication with all the templates that publication requires. You can then change the sample publication to meet your needs. On the Sourcefabric website you can find packs providing all elements of a publication: database structure, sample content and the actual templates:

`http://www.sourcefabric.org/en/products/newscoop_templates/`

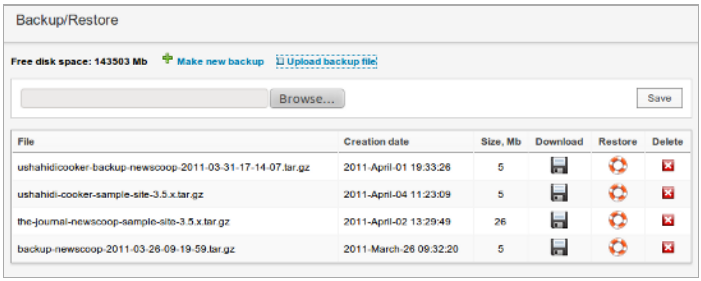
These template packs are provided in the form of Newscoop backup files.



Warning: if you already have one or more publications on your Newscoop server, uploading a template pack will erase your entire database and files from all your publications, and replace them with sample content! Do not attempt this procedure on a production Newscoop server unless you have backed it up first!

Using the Backup/Restore feature of Newscoop, you can install a template pack after downloading it from the Sourcefabric website to your local computer's desktop. Then log in to the Newscoop administration interface, and select Backup/Restore from the Actions menu.

On the Backup/Restore page, click the **Upload backup file** link, and click the **Browse** button to select the template pack from your local computer. Then click the **Save** button to upload the file. After upload, click the Restore icon for the template pack, on the right side of the list of the available backup files. This orange and white icon looks like a life preserver, of the kind you might find on a boat.



The restore process starts with an empty pop-up window, connecting to the server. Do not close this window, otherwise the restore process will be interrupted. The process takes some time (it can be minutes, depending on the size of the publication and speed of the server). It will finish with an alert window where you have to click "Resend".

After the restore process has finished, there is only one user in the system, the Administrator, with the login and password: admin

To finish the process, log in using admin as login and password. Then go to Content > Publications and select "Configure" in the new publication. Now you have to add the URL of your publication as a site alias. The template pack should now be visible at the URL of your publication.

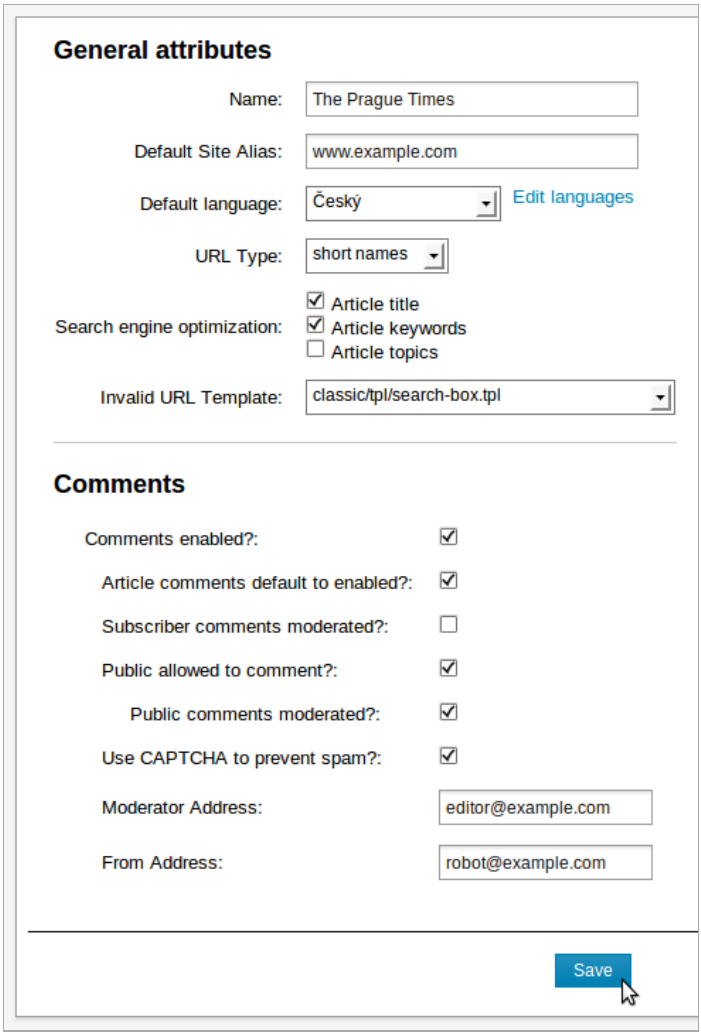
MAKING YOUR TEMPLATES

16. PUBLICATION
17. ISSUES
18. SECTIONS
19. ARTICLES
20. ARTICLE COMMENTS
21. SITES IN MULTIPLE LANGUAGES
22. SEARCH TEMPLATES
23. MAKING A 404 PAGE TEMPLATE

16. PUBLICATION

Newscoop content is organized in a hierarchical structure which conforms to the tradition of newspapers and magazines: publications, issues, sections and articles. Each publication is made up of issues, each issue is in turn made up of sections, which are comprised of articles.

In this chapter we'll look at how publication settings can be set in the admin interface and then accessed by $gimme->publication and its variants.



Configure Publication view for general attributes and comments in the administration interface

You can set a range of parameters in the configuration when creating or editing a publication in the administration interface. This chapter will explain only those parameters related to the $gimme->publication object through the template language.

Each publication has an attributes menu (accessible in the administration interface under Content->Publication and then by clicking on the wrench-and-screwdriver icon for configure).

There are three sections in the publication attributes menu for each publication:

- **General attributes** and **Comments** on the left side of the menu
- Subscription defaults on the right side of the menu

GENERAL ATTRIBUTES

The fields in the general attributes section present in $gimme->publication are listed below:

Name of the publication (e.g. "The Journal")

```
{{ $gimme->publication->name }}
```

Default Site Alias is the name of the web server on which your publication will be hosted (e.g. www.example.com). Newscoop enables multiple publications to be hosted on the same web server, provided that a unique alias has been set up for each publication by your system administrator. If you try to access the alias URL before this setting is made, you will see an error message indicating that the alias was not yet assigned to a publication.

```
{{ $gimme->publication->site }}
```

Here is an example using $gimme->publication->site taken from The Journal's _tpl/_html-head.tpl:

```
1. <script type="text/javascript">
2. var tb_pathToImage = 'http://{{ $gimme->publication->site }}/templates/set_thejournal/_img/loadingAnimation.gif';
3. </script>
4. <script type="text/javascript" src="http://{{ $gimme->publication->site }}/templates/set_thejournal/_js/tabs.js"></script>
5. <script type="text/javascript" src="http://{{ $gimme->publication->site }}/templates/set_thejournal/_js/scripts.js"></script>
6. <script type="text/javascript" src="http://{{ $gimme->publication->site }}/templates/set_thejournal/_js/easing.js"></script>
7. <link rel="index" title="The Journal" href="http://{{ $gimme->publication->site }}">
```

$gimme->publication->default_language returns information related to the publication's default language. Here are a few examples:

```
1. <ul>
2.     <li>name: {{ $gimme->publication->default_language->name }}
3.         is the language name</li>
4.     <li>number: {{ $gimme->publication->default_language->number }}
5.         is language identifier in the Newscoop database (integer value)</li>
6.     <li>english_name: {{ $gimme->publication->default_language->english_name }}
7.         is language name in English</li>
8.     <li>code: {{ $gimme->publication->default_language->code }}
9.         is language international code</li>
10.    <li>defined: {{ if $gimme->publication->default_language->defined }}SET{{ else }}NOT SET{{ /if }}
```

```
11.          is boolean value (true/false) - true if the language was set in t
     he current environment; false otherwise</li>
12. </ul>
```

A full reference to all properties of the template objects is given at the end of this book. The examples covered here might not cover all possibilities.

COMMENTS

The fields in the Comments section in the screenshot above are accessible through $gimme->publication as follows:

Public allowed to comment? Check this box if non-subscribers will be allowed to make comments on articles

```
{{ if $gimme->publication->public_comments }}YES{{ /if }}
```

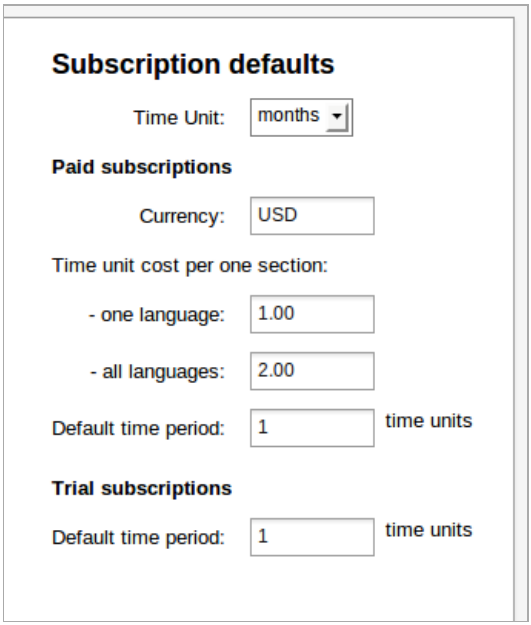
Public comments moderated? If you check this box, non-subscriber comments will be hidden from readers until they have been reviewed by a staff member

```
{{ if $gimme->publication->captcha_enabled }}CAPTCHA{{ /if }}
```

You can see some of these examples in action in the advanced section of this Cookbook.

SUBSCRIPTION DEFAULTS

The rest of the fields are related to subscriptions, which you can adjust later if you wish. First, you have to select a time unit for your subscriptions; which could be days, months, weeks or years.



Configure Subscription default view in the administration interface

You can set two types of subscriptions: paid and trial. Paid subscriptions have the following properties:

- Currency: the subscription currency. Even if a subscription request is sent and the currency setting is changed before payment, the information about the subscriber's payment due is correctly recorded
- Time unit cost per one section:
    - for one language: the price for access to a particular section in a single language
    - for all languages: the price for access to a section in all available languages
- Default time period: the usual duration of the paid subscription. This value is used when a reader subscribes through the website. The period for a particular subscription can be modified from the administration interface

Here is some sample code for subscriptions:

```
1.  <h3>Subscriptions</h3>
2.  <ul>
3.      <li>Trial subscription: {{ $gimme->publication->subscription_trial_ti
    me }} {{ $gimme->publication->subscription_time_unit }}</li>
4.      <li>Paid subscription: {{ $gimme->publication->subscription_paid_time
     }} {{ $gimme->publication->subscription_time_unit }}</li>
5.  </ul>
6.  <h4>Subscription costs:</h4>
7.  <ul>
8.      <li>{{ $gimme->publication->subscription_currency }} {{ $gimme->publi
    cation->subscription_unit_cost }} (access one language)</li>
9.      <li>{{ $gimme->publication->subscription_currency }} {{ $gimme->publi
    cation->subscription_unit_cost_all_lang }} (access all languages)</li>
10. </ul>
```

Controlling access based on subscriptions is described in the chapter titled "Subscriptions, Registration and Content Access."

17. ISSUES

If you've worked on a periodical publication (meaning that it's regularly published on a daily, weekly, monthly or quarterly schedule), you're familiar with the concept of issues. If you come from a blogging background, you're probably used to an unstructured process of publishing your articles. Often blogging systems will create arbitrary issues in your archive by clustering articles, for example in months.

In this chapter we'll look at how Newscoop works with issues, and how each issue's characteristics such as name, publish date and URL can be accessed in the template language.

In Newscoop, each issue can have a separate template assigned to it, meaning that you can have a different layout for different seasons, events or even different layouts for 1-, 2- or 3-column headlines (especially useful for breaking news coverage or slow news days). You can also translate individual issues. And, from the administration interface, you set automated publishing for an entire issue.

From the Newscoop administration interface for The Journal, you can access the issue details menu by going to Content->The Journal and then clicking on the wrench-and-screwdriver Configure icon.

Here is an example template that returns a number of attributes for a given issue:

```
1. <h3>Issue No.{{ $gimme->issue->number }}: {{ $gimme->issue->name }}</h3>
2. <ul>
3.     <li>Published: {{ $gimme->issue->publish_date|camp_date_format:"%e. %
   M %Y" }}</li>
4.     <li>Using template: {{ $gimme->issue->template->name }}</li>
5.     <li>Inside publication: {{ $gimme->issue->publication->name }}</li>
6.     <li>URL name: .../{{ $gimme->issue->url_name }}/</li>
7.     <li>Latest issue? {{ if $gimme->issue->is_current }}yes{{ else }}no{{
    /if }}</li>
8. </ul>
```

For "The Journal," this displays something like this:

Issue No.13: January 2011

- Published: 1. January 2011
- Using template: set_thejournal/front.tpl
- Inside publication: The Journal
- URL name: .../jan2011/
- Latest issue? yes

A full reference of properties for the object $gimme->issue can be found at the end of this manual.

Using 'set' and 'unset' to change values for an issue

In the templates you can also change the values for a given issue. In the following example this is done to include static pages from the first issue (example taken from _tpl/footer.tpl of "The Journal"):

```
1. {{ set_issue number="1" }}
2. {{ set_section number="5" }}
```

```
3. {{ list_articles }}
4.     <li class="page_item"><a href="{{ uri options="article" }}" title="{{
   $gimme->article->name }}">{{ $gimme->article->name }}</a></li>
5. {{ /list_articles }}
```

You can set the issue name and the issue number. You can reverse the issue to the default value (the value for the default runtime environment of the displayed page):

```
{{ set_default_issue }}
```

You can also set the issue to the last published issue:

```
{{ set_current_issue }}
```

If you want to drop the information you have set, you can use:

```
{{ unset_issue }}
```

Unsetting an issue does not lose the issue value forever. It can be set with the above commands to default, current or an assigned value.

18. SECTIONS

Sections are distinct clusters of articles. You will have seen sections in most print publications: Culture, Business, Sports, International Affairs, and so on. Usually these sections are visible in the publication's navigation; below you can see an example for the sample publication "The Journal". In this case, the section POLITICS is active, either because the reader is looking at the content of this section, or is reading an article inside the section:

Sections can be translated, and sections also have a description which can be added in the administration interface. The Newscoop developers are currently working on access levels for editors and journalists along section lines, meaning that you can assign your staff to sections and they are only allowed to work inside that section.

Here you can see the $gimme->section object in action:

```
1. <h3>Section No.{{ $gimme->section->number }}: {{ $gimme->section->name }}
   </h3>
2. {{ $gimme->section->description }}
3. <ul>
4.     <li>Published: {{ $gimme->section->publish_date|camp_date_format:"%e.
    %M %Y" }}</li>
5.     <li>Using template: {{ $gimme->section->template->name }}</li>
6.     <li>Inside: {{ $gimme->section->publication->name }}/{{ $gimme->secti
   on->issue->name }}</li>
7.     <li>URL name: .../{{ $gimme->section->url_name }}/</li>
8. </ul>
```

This displays something along these lines, depending on where you are, in "The Journal":

> **Section No.40: Health**
>
> All the things that do you good.
>
> - Published: 20. April 2011
> - Using template: set_thejournal/section.tpl
> - Inside: The Journal/January 2011
> - URL name: .../health/

Note: A section's publishing date is the same as the publishing date of the issue.

USING $GIMME->SECTION IN THE NAVIGATION TEMPLATE

In your navigation, you can use $gimme->section to check which element of the navigation needs to be active, because this is the section the reader is in (example taken from_tpl/top-nav.tpl of "The Journal"):

```
1. {{ list_sections }}
2.     <li class="cat-item{{ if $gimme->section->number ==
3.     $gimme->default_section->number }} current_page_item{{ /if }}">
4.         <a href="{{ uri options="section" }}" title="View all posts filed
 under {{ $gimme->section->name }}">{{ $gimme->section->name }}</a>
5.     </li>
6. {{ /list_sections }}
```

{{ if $gimme->section->number == $gimme->default_section->number }} checks if the section number provided by the list function is identical with the one of the page. If this is the case, an additional class is added. This class is then used in the CSS to style this element.

LISTING SECTIONS

In the above example, list_sections was already introduced. The list commands (for issues, sections, articles, languages, comments and others) can be tuned in many ways. A full reference can be found at the end of this manual.

Lists are usually not so elaborate when used on sections. You'll find more interesting examples in the chapter on articles. For sections, ordering is probably the most commonly used feature. Here are some examples; ordering by number (ascending):

```
1. {{ list_sections order="bynumber asc" }}
2.     {{ $gimme->section->number }}. {{ $gimme->section->name }}
3. {{ /list_sections }}
```

..or ordering by number (descending):

```
1. {{ list_sections order="bynumber desc" }}
2.     {{ $gimme->section->number }}. {{ $gimme->section->name }}
3. {{ /list_sections }}
```

This returns the section numbers in ascending or descending order, and also lists their names.

SETTING AND UNSETTING SECTION

You can also change the section's values in the templates. In the following example this is done to access the static pages in section number 5 of the first issue (the example is taken from _tpl/footer.tpl of "The Journal" sample publication):

```
1. {{ set_issue number="1" }}
2. {{ set_section number="5" }}
3. {{ list_articles }}
4.     <li class="page_item"><a href="{{ uri options="article" }}" title="{{
 $gimme->article->name }}">{{ $gimme->article->name }}</a></li>
5. {{ /list_articles }}
```

You can set the section name and the section number. You can reverse the section to the default value (the value for the default runtime environment of the displayed page):

```
{{ set_default_section }}
```

If you want to drop the information, you can use:

```
{{ unset_section }}
```

Unsetting a section does not lose the section information forever, it can be set with the above commands back to default, or an assigned value.

19. ARTICLES

In this chapter you will learn how to display elements of an article, as well as building article lists. Among all templates, **article.tpl** is generally the one where most template functions are called, and many sub-templates are included. No surprise, since a lot of things usually show up in an article page:

- title
- authors
- publish date
- intro
- full text
- attachments
- comments
- map
- the list of other articles from the same section
- ...?

Each article has an **Article Type**, which has a list of fields you set up to reflect the content (like "intro", "twitter", "full_text", "seo_title" and so on). Some of the elements in the list above refer to fields defined in the Article Type, like title, intro or full text. Others are objects related to an article, like attachments, maps or comments. There can also be references to objects independent from the article, which are linked to the article in the Article Edit screen of the administration interface, like authors.

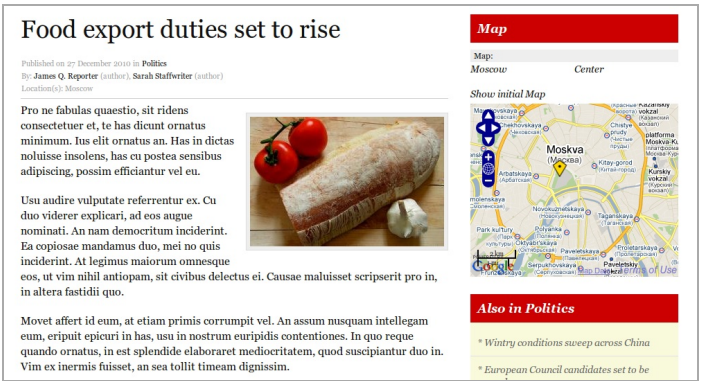
You are advised to use the {{ include }} feature of the template engine to manage all possible shapes and forms an article can take. Chop your article template into sub-templates and call them in when and where you need them. As we already described in a previous chapter on cutting a HTML page into templates, it is much easier to divide a template into pieces and use includes.

You can take a look at the example article.tpl in the template pack "The Journal".

Best practice is to separate the main article content from the other auxiliary parts. That's why we have different template includes like:

```
{{ include file="set_thejournal/_tpl/article-cont.tpl" }}
{{ include file="set_thejournal/_tpl/article-comments.tpl" }}
{{ include file="set_thejournal/_tpl/article-map.tpl" }}
{{ include file="set_thejournal/_tpl/sidebar-related.tpl" }}
```

Let's take a look at what is going on inside the _tpl/article-cont.tpl file:



Design delivered by template _tpl/article-cont.tpl

The article title is the most important thing in the Article. It should use either the <h1> or <h2> heading tag (depending on how you decided to mark up your publication name; see more on these issues in the chapter on Search Engine Optimisation):

```
<h2>{{ $gimme->article->name }}</h2>
```

You can refer to the publish date and the section where the article came from. Article authors, photographers and other contributors are being mentioned next.

```
1. Published on {{ $gimme->article->publish_date|camp_date_format:"%e %M %Y"
   }}
2. in <a href="{{ uri options="section" }}">{{ $gimme->section->name }}</a>
3. <br />
4. By: {{ list_article_authors }}
5.       {{ $gimme->author->name }} ({{ $gimme->author->type|lower }})
6.        {{ if !$gimme->current_list->at_end }}, {{ /if }}
7.     {{ /list_article_authors }}
```

Do you map and display locations? Great! Here is how to display them on the page:

```
1. Location(s): {{ list_article_locations }}
2.              {{ if $gimme->location->enabled }}{{ $gimme->location->name
   }}
3.               {{ if $gimme->current_list->at_end }}{{ else }}, {{ /if }}
4.              {{ /if }}
5.             {{ /list_article_locations }}
```

Now let's start the story. In the following code snippet we will display the introduction from the article, using the Article Field **intro**. We are also checking if the image with the number 1 is present and present it accordingly. The last line lists the rest of the story, stored in the Article Field **full_text**.

```
1. {{if $gimme->article->has_image(1)}}<img src="{{url options="image 1"}}"
   alt="{{$gimme->article->image1->description}}" />{{/if}}
2. <div class="intro">{{ $gimme->article->intro }}</div>
3. <div class="full_text">{{ $gimme->article->full_text }}</div>
```

Not that difficult, is it? If you are still with us, go an extra round and check if the content is actually available to the reader, with $gimme->article->content_accessible. If the value returned is TRUE, either the article is available to everybody or the reader is logged in and has the right to access the article. You can find more on subscription management in the section of this Cookbook on *Advanced templating*.

```
2.   {{if $gimme->article->has_image(1)}}<img src="{{url options="image 1"}}
" alt="{{$gimme->article->image1->description}}" />{{/if}}
3.   <div class="intro">{{ $gimme->article->intro }}</div>
4.   <div class="full_text">{{ $gimme->article->full_text }}</div>
5. {{ else }}
6.   <p>This article is accessible only to registered and logged in users!</
p>
7. {{ /if }}
```

As you can see, the **article-cont.tpl** sub-template is mostly about the article content itself. The rest are auxiliary, but very important parts. They are also included in the **article.tpl** template.

ARTICLE COMMENTS

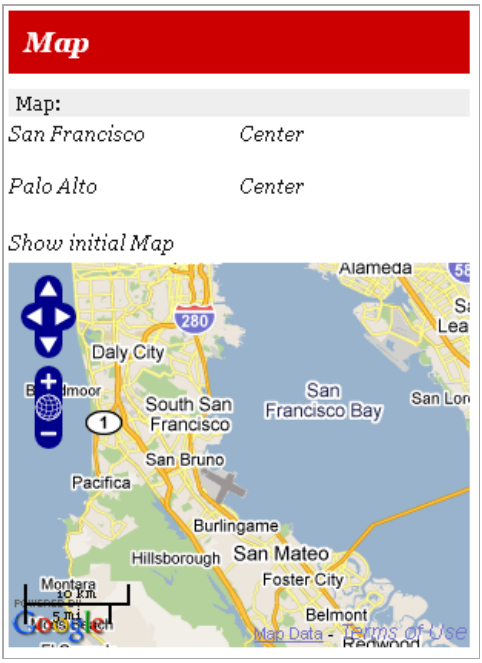
Comments are explained in more detail in the following chapter. We believe that they are an essential way to communicate with your audience - and for your audience to communicate among themselves. In "The Journal" template pack, comments look like this:

Screenshot from the comments on "The Journal" template package.

ARTICLE MAP

To round up the article page, here is the line that displays a map with locations that have been set in the Article Edit page by your journalists. There is more about maps and geolocation in a later chapter.

```
1. <div class="widget block"><h3>Map</h3>
2. {{ map show_locations_list="true" show_reset_link="Show initial Map" widt
h="300" height="250" }}
3. </div>
```



Map embedded in article. Locations are handled in the Article Edit screen.

ARTICLE ATTACHMENTS

Journalists can attach files to an article. We also include them in the article.tpl as a separate sub-template **article-attachments.tpl**. The code snippet below checks if attachments are present, and displays a list if there are.

```
{{if $gimme->article->has_attachments}}
 <ul>
  {{list_article_attachments}}
   <li>
    <a href="{{uri options="articleattachment"}}">
    {{$gimme->attachment->file_name}}</a>
    [{{ $gimme->attachment->extension }}|{{$gimme->attachment->size_kb}}Kb]
    <br />
    {{$gimme->attachment->description}}
   </li>
  {{/list_article_attachments}}
 </ul>
{{/if}}
```

LISTING ARTICLES

You can create lists of articles, which is usually done inside a section overview of the content. But it can also be interesting to use this on the article page for "further reading" or "related articles".

Listing articles with **list_articles** is the most powerful and mostly used statement in Newscoop. You can check the Template Reference at the end of this Cookbook for all function options. Here are just a few examples to give you a taste:

List the last 10 articles:

```
1. {{ list_articles length="10" order="byPublishDate desc" ignore_issue="tru
   e" ignore_section="true"}}
2. {{* code goes here *}}
3. {{ /list_articles }}
```

Show last published article, of type 'article', from section number 100, regardless of issue:

```
1. {{list_articles length="1" constraints="type is article section is 100" i
   gnore_issue="true" order="bypublishdate desc"}}
2. {{* code goes here *}}
3. {{/list_articles}}
```

List the last article from section numbers 100 to 140:

```
1. {{ list_sections constraints="number greater_equal 100 number smaller_equ
   al 140" }}
2.   {{ list_articles length="1" ignore_issue="true" order="byPublishDate de
   sc" }}
3. {{* code goes here *}}
4.   {{/list_articles}}
5. {{/list_sections}}
```

List 10 more articles from the same section, ordered by publish date, excluding the one already defined:

```
1. {{list_articles length="10" constraints="number not `$gimme->default_arti
   cle->number`" ignore_issue="true" order="bypublishdate desc"}}
2. {{* code goes here *}}
3. {{/list_articles}}
```

20. ARTICLE COMMENTS

In this chapter you will learn how to display comments and the comment form, and how to use CAPTCHA or ReCAPTCHA spam protection. Comments are the place for your readers to give their feedback on an article. Comments also reflect the conversation of the community of your publication. Your readers are not only communicating their ideas about the article, but also communicating with each other.

Enabling and disabling comments is set for each publication in the administration interface. Here you can enable or disable comments for publications, for article types and for individual articles. If you switch off comments at the publication level, no comments can be added at all. The next level is article type: if you disable comments here, the option will not appear in the Article Edit screen for articles of this type. If both of these options are enabled, editors can switch commenting off for an individual article.

LISTING THE MOST COMMENTED ARTICLES

Before we dive into article comments, here's a little nugget showing how you can list articles by the number of comments they have, in descending order:

```
1. {{ list_articles order="bycomments desc" }}
2.   <p>
3.     {{ $gimme->article->name }},
4.     comments: {{ $gimme->article->comment_count }}
5.   </p>
6. {{ /list_articles }}
```

This **list_articles** function will list the articles inside the current section, by default. To list all articles from an issue use:

```
{{ list_articles ignore_section="true" order="bycomments desc" }}
```

For the entire publication, use:

```
{{ list_articles ignore_section="true" ignore_issue="true" order="bycomments des
c" }}
```

LISTING ARTICLE COMMENTS

The following code has been taken from the template pack "The Journal", designed by Woo Themes and available on the Sourcefabric website. If comments are available, you can list them like this:

```
 1. {{ list_article_comments columns="2" order="bydate desc"}}
 2. {{ if $gimme->current_list->at_beginning }}
 3.   <h4>{{ $gimme->article->comment_count }} Response(s) to "{{ $gimme->art
    icle->name }}"</h4>
 4.   <ol class="commentlist">
 5. {{ /if }}
 6.    <li class="comment {{ if $gimme->current_list->column == "1" }}odd{{ e
    lse }}even{{ /if }}">
 7.       <div class="comment-head cl">
 8.          <div class="user-meta">
 9.              <strong class="name">{{ $gimme->comment->nickname }}</strong
    >
10.              {{ $gimme->comment->submit_date|camp_date_format:"%e.%m.%Y a
    t %H:%i" }}
```

```
11.             </div>
12.         </div>
13.         <div class="comment-entry">
14.             <p>{{ $gimme->comment->content }}</p>
15.         </div>
16.     </li>
17. {{ if $gimme->current_list->at_end }}
18.   </ol>
19. {{ /if }}
20. {{ /list_article_comments }}
```

list_article_comments lists the comments. **columns="2"** is used to add "odd" and "even" classes in the list. **order="bydate desc"** assures that the newest comment appears on the top of the list. The other values in this example are pretty much self-explanatory.

More properties of the article comment can be printed, like e-mail or unique ID. You can find these properties in the reference part of this Cookbook.

If you want to display the comments only if commenting is enabled, use the above code inside the following IF function:

```
1. {{ if $gimme->article->comments_enabled }}
2. [... code goes here ...]
3. {{ /if }}
```

If you want to display the comments only if the reader has access to the content of the article - either because it is available to all, or because the user is logged in and has a subscription to the content - use the above code inside the following IF function:

```
1. {{ if $gimme->article->content_accessible }}
2. [... code goes here ...]
3. {{ /if }}
```

You can also combine the two like this:

```
1. {{ if $gimme->article->comments_enabled && $gimme->article->content_acces
   sible }}
2. [... code goes here ...]
3. {{ /if }}
```

CREATING THE FORM FOR ARTICLE COMMENTS

The comment form can be styled freely. It is wrapped in {{ comment_form }} which creates the **form** tag automatically. The HTML inside is limited only by your imagination.

```
1. {{ comment_form html_code="id=\"commentform\"" submit_button="SUBMIT" but
   ton_html_code="tabindex=\"6\"" }}
2. <label for="author"><small>Name (required)</small></label>
3. {{ camp_edit object="comment" attribute="nickname" html_code="id=\"author
   \" 22\" tabindex=\"1\"" }}
4. <label for="email"><small>E-mail (will not be published) (required)</smal
   l></label>
5. {{ camp_edit object="comment" attribute="reader_email" html_code="id=\"em
   ail\" 22\" tabindex=\"2\"" }}
6. <input type="hidden" name="f_comment_subject" value="Site comment" />
7. {{ camp_edit object="comment" attribute="subject" html_code="id=\"comment
   -subject\" tabindex=\"3\"" }}
8. <label for="comment"><small>Comment</small></label>
9. {{ camp_edit object="comment" attribute="content" html_code="id=\"comment
   \" rows=\"5\" tabindex=\"4\"" }}
10. <img src="{{ captcha_image_link }}"><br />
```

```
11. <label for="f_captcha_code"><small>Enter the code:</small></label>{{ camp
    _edit object="captcha" attribute="code" html_code="id=\"comment-code\" ta
    bindex=\"5\"" }}
12. {{ /comment_form }}
```

As shown above, the display of the form can be controlled to show up only IF the article allows commenting or IF the user has access to the article.

SPAM CONTROL WITH CAPTCHA

To prevent automated spamming, you can add a CAPTCHA image to your comment form. It displays letters which the user needs to type in before posting their comment. Include a CAPTCHA by adding the following lines just before {{ /comment_form }}:

```
1. <img src="{{ captcha_image_link }}"><br />
2. <label for="f_captcha_code"><small>Enter the code:</small></label>{{ camp
   _edit object="captcha" attribute="code" html_code="id=\"comment-code\" ta
   bindex=\"5\"" }}
```

When submitting the comment, the string typed by the reader is checked against the CAPTCHA image. If it is not identical, an error can be displayed.

```
1. {{ if $gimme->submit_comment_action->is_error }}
2.   {{ $gimme->submit_comment_action->error_message }}
3.   {{ $gimme->submit_comment_action->error_code }}
4. {{ /if }}
```

The error code is machine generated and can not be changed. The error message can be changed in the administration interface, in "Localization".

SPAM CONTROL WITH RECAPTCHA NEWSCOOP PLUGIN

Instead of the solution mentioned above, you can use the ReCAPTCHA plugin (which works with Newscoop 3.5.2 and higher). In that case, you only need to use:

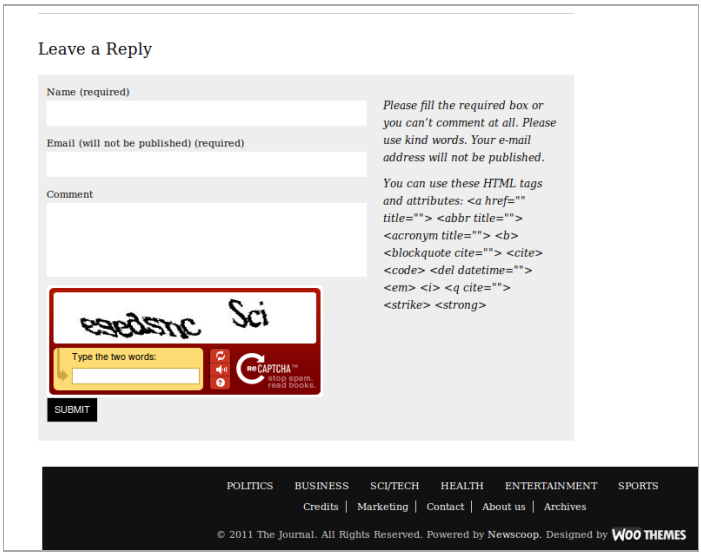
```
<p>
{{ recaptcha }}
</p>
```

You can read what reCAPTCHA is at http://www.google.com/recaptcha

Then, make sure to create the proper key for your site here:

https://www.google.com/recaptcha/admin/create

Your comment form will look like this



Steps to perform in order to work with this plugin:

1. Download the recaptcha plugin from our site (if you installed Newscoop 3.5.2 or higher, you should have this plugin already in your Plugins Manager)
2. Install and enable the plugin through the Plugins Manager in the Newscoop administration interface (in the main menu, Plugins -> Manage Plugins)
3. Configure the plugin, the options are:
    1. Enable for comments:
    2. Enable for subscriptions:
    3. Enter the public key:
    4. Enter the private key:
4. Include the appropriate template tag within your forms:

comments form:

```
{{ recaptcha }}
```

subscriptions form:

```
{{ recaptcha form='subscriptions' }}
```

Finally, enable the use of CAPTCHA for your publication in the Publication configure screen.

CHECKING FOR ERRORS AND ARTICLE MODERATION

To go through the process of submitting, checking and giving feedback on article moderation you could structure the template in the following way:

```
1. {{ if $gimme->submit_comment_action->defined && $gimme->submit_comment_action->rejected }}
2.     Your comment has not been accepted.
3. {{ /if }}
4. {{ if $gimme->submit_comment_action->is_error }}
5.     {{ $gimme->submit_comment_action->error_message }}
6.     {{ $gimme->submit_comment_action->error_code }}
7. {{ else }}
```

```
    {{ if $gimme->submit_comment_action->defined }}
        {{ if $gimme->publication->moderated_comments }}
            Your comment has been sent for approval.
        {{ /if }}
    {{ /if }}
{{ /if }}
<h2>Leave a Reply</h2>
{{ if $gimme->user->blocked_from_comments }}
    You are not allowed to comment.
{{ else }}
{{ comment_form html_code="id=\"commentform\"" submit_button="SUBMIT" but
ton_html_code="tabindex=\"6\"" }}
[...]
{{ /comment_form }}
{{ /if }}
```

In this example you can also see where and how to place feedback for banned users.

PREVIEW OF COMMENTS

If you want to allow readers to preview their comments, you need to add the code preview_button="Preview" to {{ comment_form ... }}. "Preview" is the text that will be displayed on the button, but you can change this value.

```
{{ comment_form html_code="id=\"commentform\"" submit_button="SUBMIT" preview_b
utton="Preview" button_html_code="tabindex=\"6\"" }}
```

If you add the preview_button, you will see the button show up at the end of the form. If the reader writes a comment and clicks "Preview", the preview can be displayed using the following template code:

```
{{ if $gimme->preview_comment_action->is_error }}
  <p>There was an error previewing the comment:
     {{ $gimme->preview_comment_action->error_message }}
  </p>
{{ /if }}
{{ if $gimme->preview_comment_action->ok }}
    <b>Comment preview</b><br/>
    Subject: {{ $gimme->preview_comment_action->subject }}<br/>
    Reader e-mail: {{ $gimme->preview_comment_action->reader_email }}<br/
>
    Content: {{ $gimme->preview_comment_action->content }}
{{ /if }}
```

A good place to put this code is just above the comment form.

NESTED COMMENTS: USING THREADS AND LEVELS

Comments can be displayed as nested trees.

```
<ul>
  {{ assign var="level" value="1" }}
  {{ list_article_comments order="bydate asc" }}
    {{ if $gimme->comment->level gt $level }}
      {{ assign var="level" value=$gimme->comment->level }}
      <ul>
    {{ /if }}
    {{ if $gimme->comment->level < $level }}
      {{ php }}
        $gimme = $this->get_template_vars('gimme');
        $level = $this->get_template_vars('level');
        $count = $level - $gimme->comment->level;
        for (; $count > 0; $count --) {
```

```
14.          echo "</ul>";
15.        }
16.      {{ /php }}
17.      {{ assign var="level" value=$gimme->comment->level }}
18.    {{ /if }}
19.    <li>{{ if $gimme->comment == $gimme->default_comment }}<b>{{ /if }}
20.    Level: {{ $gimme->comment->level }}
21.    <a href="{{ uri }}#comments">
22.    Subject: {{ $gimme->comment->subject }}, Reader email: {{ $gimme->com
    ment->reader_email }}
23.    </a>
24.    {{ if $gimme->comment == $gimme->default_comment }}</b>{{ /if }}<br/>
25.    Content: {{ $gimme->comment->content }}
26.    </li>
27.  {{ /list_article_comments }}
28. </ul>
```

21. SITES IN MULTIPLE LANGUAGES

In this chapter, we're working with one of Newscoop's main advantages: the ability to easily create and manage multilingual publications.

The following screenshot shows the administration interface of a publication in English, Russian and Spanish. This is what you will see when you go to the list of issues:

Number	Name (click to see sections)	URL Name	Publish Date (YYYY-MM-DD)	Configure	Translate	Preview	Delete
15	March 2011 (English)	mar2011	2011-03-01 13:34:06 Unpublish - Schedule				
15	Marzo 2011 (Español)	mar2011	2011-03-03 14:32:33 Unpublish - Schedule				
15	Март 2011 (Русский)	mar2011	2011-03-03 14:33:49 Unpublish - Schedule				
14	February 2011 (English)	feb2011	2011-03-01 13:33:32 Unpublish - Schedule				
14	Febrero 2011 (Español)	feb2011	2011-03-01 13:33:32 Unpublish - Schedule				
14	Февраль 2011 (Русский)	feb2011	2011-03-04 15:44:40 Unpublish - Schedule				
13	January 2011 (English)	jan2011	2011-03-01 13:32:53 Unpublish - Schedule				
13	Enero 2011 (Español)	enero2011	2011-03-06 03:45:51 Unpublish - Schedule				
13	Январь 2011 (Русский)	jan2011	2011-03-04 15:44:45 Unpublish - Schedule				
12	December 2010 (English)	dec2010	2011-03-01 13:31:50 Unpublish - Schedule				
12	Diciembre 2011 (Español)	dic2010	2011-03-06 03:46:38 Unpublish - Schedule				
12	Декабрь 2010 (Русский)	dec2010	2011-03-04 15:44:51 Unpublish - Schedule				
11	November 2010 (English)	nov2010	2011-03-01 13:30:30 Unpublish - Schedule				
11	Noviembre 2010 (Español)	nov2010	2011-03-06 03:47:05 Unpublish - Schedule				
11	Ноябрь 2010 (Русский)	nov2010	2011-03-04 15:44:56 Unpublish - Schedule				

List of issues in the admin interface for a multilingual publication in English, Russian and Spanish.

In Newscoop, there is the publication's default language (set in Content >Publication >Configure) and other language versions, or translations. Other languages are defined by creating translations of issues and articles. In the screenshot above, you can see that every issue is translated into two additional languages.

You don't need to create these translations with every new issue - when you add a new issue and choose the option to 'Use the structure of previous issue', a new issue will be created with all the language versions that already exist.

Note that all of an issue's language versions hold the same issue number. Similarly, sections in translated issues keep the original section's number, and translated articles keep the original article's number.

So how does Newscoop handle multilingual content? By changing the language parameter.

Take a look at these three URLs:

http://example.com/en/mar2011/posts/4/healthy-options.htm

http://example.com/es/mar2011/posts/4/opciones-saludables.htm

http://example.com/ru/mar2011/posts/4/-.htm

The most important difference regarding multilinguality lies in the language code (en, es, ru) and in the last part of the URL, generated using topics for SEO purposes. Incidentally, those links will work without that last part of the url. The article number (4) is enough to tell Newscoop which article it is.

Switching between languages

To switch from one language to another, we build a list of available languages. In our language switcher, if we change only the language parameter, all other parameters currently active will be used. If you change the language while on the article page, you will jump from translation to translation.

```
1. <ul>
2. {{ list_languages of_publication="true" }}
3.     <li style="background: transparent url(http://{{ $gimme->publication-
   >site }}/templates/_img/flags/{{ $gimme->language->code }}.png) no-repeat
    5px center">
4.         <a href="{{ uri }}/">{{ $gimme->language->name }}</a>
5.     </li>
6. {{ /list_languages }}
7. </ul>
```

This switcher lists all languages defined by the content inside a publication. Because you cannot place a Greek translation into a Spanish issue, the appropriate issue is automatically created at the moment when you try to create an article in a language which is not yet defined.

Now let's get back to our language switcher. It will try to keep the active issue, section and article (if they exist in the required language). If not, the switcher will simply fall back to the previous level where a translation exists (i.e. section, and if section doesn't exist, then issue, which definitely exists - otherwise it wouldn't be on the language list).

What happens once your user switches to another language? Newscoop tries to serve all database content in that language - not only articles, but all other elements like system values for date (day names and month names), topics, author biographies etc. It is therefore important that all these values are translated to languages you plan to use.

Template strings also need translation; this is how you treat every string:

```
2. {{ if $gimme->language->english_name == "Spanish" }}Bsqueda{{ /if }}
3. [...]
```

NOTE: There are some pieces of content that are not yet translatable:

1. Image captions
2. Form buttons
3. Some configuration settings that might be used to output values, for example {{ $siteinfo.description }}

You can solve this situation with following workarounds:

1. Create an article field of single-line text, where journalists can store translations of image captions
2. For forms, you will need to set 'if' statements to check which language page is opened and to provide a version of the form with appropriate button labels (or graphical buttons)
3. Skip using configuration values, and instead use the translatable information stored in the publication structure (in some special article maybe, or as section description, or... be creative)

22. SEARCH TEMPLATES

You can control your search form and search results with templates. This gives you design freedom for the list of results, as well as the complexity and appearance of your search form.

The Search Form

Using the default Newscoop function {{ search_form }} creates a form like the following screenshot:

You can use the following code to create the form:

```
1. <h3>Search Articles</h3>
2. {{ search_form template="search.tpl" submit_button="Search" html_code="cl
   ass=\"group\" id=\"search-form\"" button_html_code="id=\"search-button\""
    }}
3.     {{ camp_edit object="search" attribute="keywords" html_code="id=\"sea
   rch-field\"" }}
4. {{ /search_form }}
```

The search terms are sent to the sub-template specified inside the function: **search.tpl**. (Note: if your search.tpl file is inside a folder, you need to specify the full path, the same as with the include function. You can also see how different html classes/IDs can be added to the form elements (using html_code and button_html_code; quotes inside need to be escaped), and how the submit_button text is defined (submit_button="Search").

To create a search form which doesn't have a button with text, but an image like the one shown below, you can use the following approach.


```
1. <div class="search">
2. {{ search_form template="search.tpl" submit_button=" " html_code="id
   =\"topSearch\"" button_html_code="class=\"replace\"" }}
3.             <p class="fields">
4.                 {{ camp_edit object="search" attribute="keywords" html_cod
   e="id=\"s\" }}
5.             </p>
6. {{ /search_form }}
7. </div><!-- /.search -->
```

Search results

First, here is the example code for the search results template **search.tpl**:

```
1. <div id="main">
2.     <h1>Search results</h1>
3.
4. {{ list_search_results length="5" order="bypublishdate desc" constraints=
   "type is news" }}
5.
6.         <div class="post">
```

```
7.              <h2 class="post-title"><a href="{{ uri options="article" }}" rel="bookmark" title="{{ $gimme->article->name }}</a></h2>
8.              <p class="post-details">Published on {{ $gimme->article->publish_date|camp_date_format:"%e %M %Y" }} by  {{ $gimme->article->author->name }} in <a href="{{ uri options="section" }}">{{ $gimme->section->name }}</a></p>
9.              {{ if $gimme->article->has_image(1) }}<a title="{{ $gimme->article->name }}" href="{{ uri options="article" }}"><img src="{{ uri options="image 1 width 134"}}" alt="{{ $gimme->article->image->description }}" class="woo-image thumbnail"></a>{{ /if }}
10.             <p>{{ $gimme->article->deck }}</p>
11.         </div><!-- /.post -->
12.
13. {{ if $gimme->current_list->at_end }}
14.         <div class="more_entries">
15.             <div class="alignleft">{{ if $gimme->current_list->has_previous_elements }}<a href="{{ uripath options="template search.tpl" }}?{{ urlparameters options="previous_items" }} ">&laquo; Newer Entries</a>{{ /if }}</div>
16.             <div class="alignright">{{ if $gimme->current_list->has_next_elements }}<a href="{{ uripath options="template search.tpl" }}?{{ urlparameters options="next_items" }}">Older Entries &raquo;</a>{{ /if }}</div>
17.         </div>
18. {{ /if }}
19. {{ /list_search_results }}
20.
21. {{ if $gimme->prev_list_empty }}
22.       <div class="postinformation">No results found</div>
23. {{ /if }}
24.
25.             </div><!-- /#main -->
```

Inside the list of search results, we have a div container of the class "post" which is being repeated as many times as there are results for the search terms. List length is limited to 5, so if there are more than five list elements, a link to the next page is created. The link to the previous page is also created, in case the reader is not on the first page of search results. All of this is done inside an if statement:

```
1. {{ if $gimme->current_list->at_end }}
2. ....
3. {{ /if }}
```

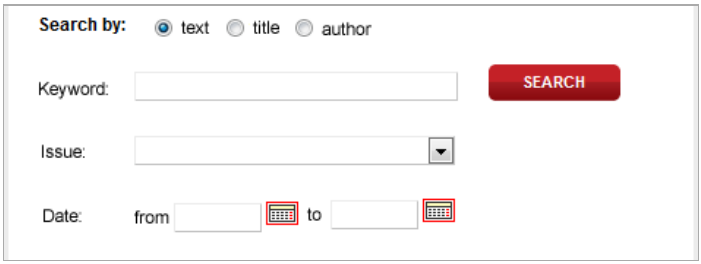
You can also find a neat pagination example in the advanced section of this manual, where pagination is explained. If no search results were found, Newscoop would display a message like this:

```
1. {{ if $gimme->prev_list_empty }}
2.       <div class="postinformation">No results found</div>
3. {{ /if }}
```

The parameter prev_list_empty refers to the list just before this statement - which was the search results list. In case the list of search results was empty, this sends an appropriately apologetic message.

Advanced search

Newscoop also offers 'advanced search' options. An advanced search form may appear to your publication's readers like this screenshot:



The code which generates this form is shown below:

```
1.  {{ search_form template="search.tpl" submit_button="Search" button_html_c
    ode="id=\"adv-search-button\" class=\"rounded\"" }}
2.  <div class="left">
3.      <div class="form-element">
4.          <label>Search by:</label>
5.          <input class="radio" name="f_search_scope" value="content" checke
    d="checked" type="radio">text
6.          <input class="radio" name="f_search_scope" value="title" type="ra
    dio">title
7.          <input class="radio" name="f_search_scope" value="author" type="r
    adio">author
8.      </div>
9.      <div class="form-element">
10.         <label for="adv-search">Keyword:</label>
11.         {{ camp_edit object="search" attribute="keywords" html_code="id=\
    "adv-search\"" }}
12.     </div>
13.     <div class="form-element">
14.         <label for="adv-select">Issue:</label>
15.         {{ camp_select object="search" attribute="issue" html_code="id=\"
    adv-select\"" }}
16.     </div>
17.     <div class="form-element">
18.         <label>Date:</label>
19.         <div class="g-left">from {{ camp_edit object="search" attribute="
    start_date" }}</div>
20.         <div class="g-right">to {{ camp_edit object="search" attribute="e
    nd_date" }}</div>
21.     </div>
22. </div><!-- /.left -->
23. <div class="right">{{ /search_form }}</div>
```

This form is different than the simple search form earlier in this chapter, as it has more options for filtering the results of the search. Firstly, the reader has the option to narrow their search only to article text, article title, or article author, by selecting the appropriate radio button. The generated HTML for this part of the form looks like this:

```
1.  <div class="form-element">
2.      <label>Search by:</label>
3.      <input class="radio" name="f_search_scope" value="content" checked="c
    hecked" type="radio">text
4.      <input class="radio" name="f_search_scope" value="title" type="radio"
    >title
5.      <input class="radio" name="f_search_scope" value="author" type="radio
    ">author
6.  </div>
```

Next, the reader has the option to select a particular issue to narrow down their search, and this part of the code is responsible for that feature:

```
1.  <div class="form-element">
2.      <label for="adv-select">Issue:</label>
```

```
3.     {{ camp_select object="search" attribute="issue" html_code="id=\"adv-
   select\"" }}
4. </div>
```

The HTML generated after template parsing is:

```
1. <div class="form-element">
2.     <label for="adv-select">:</label>
3.     <select name="f_search_issue" id="adv-select">
4.         <option value="0" selected="selected"> </option>
5.         <option value="8">8. Issue 8 (2011-03-18 08:00:08)</option>
6.         ....
7.         <option value="1">1. Issue 1 (2010-12-02 08:00:07)</option>
8.     </select>
9. </div>
```

The final option is to specify the time frame from which the reader wants to get search results. This is done with inline date choosers:

```
1. <div class="form-element">
2.     <label>Date:</label>
3.         <div class="g-left">from {{ camp_edit object="search" attribute="
   start_date" }}</div>
4.         <div class="g-right">to {{ camp_edit object="search" attribute="e
   nd_date" }}</div>
5. </div>
```

The HTML that gets generated by this piece of Newscoop code is rather too long for this book. You can style the start and end date fields in CSS with the following id's:

```
1. #advanced-search #f_search_end_date,
2. #advanced-search #f_search_start_date {
3.     width: 58px;
4. }
```

If you need to style the calendar that pops up when the reader clicks the date chooser field, there is the file javascript/jscalendar/calendar-system.css in your Newscoop installation. Feel free to play with it.

23. MAKING A 404 PAGE TEMPLATE

Good-looking, user-friendly "404 page" templates let people know that although the page they requested can't be found, the site is still up and running correctly. Another Newscoop feature at the publication configuration level enables site developers to specify the template which will be used when readers try to load an invalid URL or non-existent page.

Your job is to design a new template for this purpose, and then to select it in the drop-down menu above. Every time a site visitor tries to open a non-existent page, you can provide them with a message that something went wrong, and offer useful links to pages they might be interested in.

For example, this ccould be your 404 template:

```
1.  {{ include file="_tpl/_html-head.tpl" }}
2.  <body>
3.  <div id="top">
4.        <div id="top-meta">
5.            <div class="date">{{$smarty.now|camp_date_format:"%M %e, %Y"}}
    </div>
6.  {{ include file="_tpl/top-search-box.tpl" }}
7.        </div><!-- /#top-meta -->
8.        <div id="header">
9.            <div class="logo">
10.               <a href="http://{{ $gimme->publication->site }}" title="{{
    $gimme->publication->name }}"><img src="http://{{ $gimme->publication->si
    te }}/templates/_img/logo.png" alt=""></a>
11.           </div><!-- /.logo -->
12.       </div><!-- /#header -->
13. </div><!-- /#top -->
14. <div>
15. {{ if ! $gimme->url->is_valid }}
16.           <h1>Sorry, the requested page was not found.</h1>
17. {{ /if }}
18. </div>
19. <div id="footer">
20.       <ul id="category-nav">
21. {{ list_sections }}
22.       <li class="cat-item"><a href="{{ uri options="section" }}" title="V
    iew all posts filed under {{ $gimme->section->name }}">{{ $gimme->section
    ->name }}</a></li>
23. {{ /list_sections }}
24.           </ul>
25.           <ul id="page-nav">
26. {{ set_issue number="1" }}
27. {{ set_section number="5" }}
28. {{ list_articles }}
29.             <li class="page_item"><a href="{{ uri options="article" }}" ti
    tle="{{ $gimme->article->name }}">{{ $gimme->article->name }}</a></li>
30. {{ /list_articles }}
31. {{ unset_section }}
32.             <li style="border: medium none;" class="page_item"><a href="{
    { uri options="template set_thejournal/archive.tpl" }}" title="Archives">
    Archives</a></li>
33. {{ set_default_issue }}
34.           </ul>
35.       </div>
36. </body>
37. </html>
```

A quicker solution may be to redirect all invalid URL requests back to your home page, but visitors deserve more precise information about what's going on, as well as a choice about what they want to do next.

ADVANCED TEMPLATING

24. TOPICS, SWITCHES, KEYWORDS TO STRUCTURE CONTENT
25. MANAGING STATIC PAGES
26. LIST OF POPULAR ARTICLES (MOST READ)
27. LIST OF LATEST COMMENTS
28. USING MAPS
29. MANAGING MULTIPLE AUTHORS AND ARTICLES
30. SUBSCRIPTIONS, REGISTRATION AND CONTENT ACCESS
31. BROWSER DETECTION AND ROBOTS (SEARCH ENGINES)
32. MOBILE DEVICE DETECTION AND TEMPLATES
33. TAG CLOUDS USING TOPICS
34. DYNAMIC PAGE LAYOUTS
35. WORKING WITH IMAGES
36. PAGINATION OF LISTS AND LONG ARTICLES
37. AUDIO PLAYER FOR ATTACHED MP3 FILES
38. RSS, SITEMAP, KML AND XML
39. SEARCH ENGINE OPTIMISATION (SEO)
40. NEWSCOOP AND HTML5
41. TROUBLESHOOTING TEMPLATES

24. TOPICS, SWITCHES, KEYWORDS TO STRUCTURE CONTENT

This chapter explains how to order your content in many different ways. Newscoop's structure of issue > section > article has many advantages. Depending on your publication, you might want to add other ways to present, structure and group your content. Topics, switches and keywords allow you to do exactly that.

With topics, you can cross-reference your content, similar to what tags or categories generally do. Topics are organized like a tree, with root topics and subtopics. Topics can be translated, can be part of an article type and can become part of the URL. You can check in your template if and what topics are assigned to an article. In this way you can use topics to structure your content - and change your layout, if you want.

Some publications using Newscoop replace the concept of issue > section > article entirely and order their content exclusively using topics. Structuring with topics and using the tree of topics and subtopics offers some liberties:

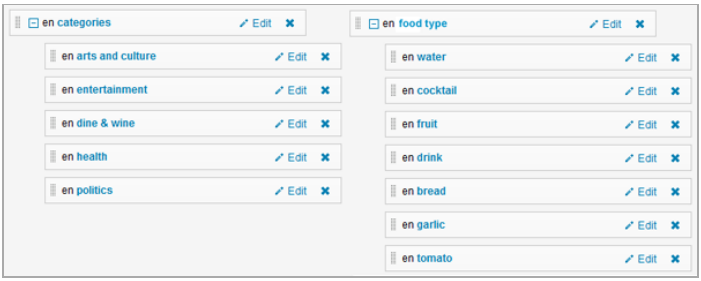
- An article can have more than one topic
- The topic tree structure allows variable depth of content
- Based on the position in the topic tree, similar articles can be grouped

Switches are another powerful way to filter your content. Beside two built-in switches ('Show article on front page' and 'Show article on section page'), you can create custom switches, and then filter your content if articles have these switches active.

Finally, keywords can be used in some cases to further fine-tune your article listings, but also as an option in creating article URLs which are human readable, increasing SEO.

TOPICS

How you organize topics in a topic tree may be very important. It's usually a good approach to create a root topic for group of topics, and then make subtopics in that branch.



This way, you can approach a specific topic branch in your templates. Let's look at an example - you are on a full article page, and want to provide information about all the categories that article is assigned to.

The code for this example is:

```
1. <div class="tags">
2.     <p>Posted in {{ list_article_topics root="categories:en" }}<a href="{
   { uri options="template index.tpl" }}" title="View all posts in category
   '{{ $gimme->topic->name }}'" rel="category tag">{{ $gimme->topic->name }}
   </a>{{ if !$gimme->current_list->at_end }}, {{ /if }}{{ /list_article_top
   ics }}</p>
3.     <p>Food type: {{ list_article_topics root="Food type:en" }}<a href="{
   { uri options="template archive-food-type.tpl" }}" title="View all posts
   for food type '{{ $gimme->topic->name }}'" rel="food-type">{{ $gimme->top
   ic->name }}</a>{{ if !$gimme->current_list->at_end }}, {{ /if }}{{ /list_
   article_topics }}</p>
4. </div>
```

You see that by specifying option root="categories:en" we can narrow down the list of topics. If we leave the root option unspecified, the listing will return all topics assigned to an article, regardless of the branch they are in.

What's more, this allows us to make topic names links to pages which list all articles assigned that topic. This is possible because the topic parameter is forwarded to the next page as one of the URL parameters, for example ?tpid=34. As we said before, when some topic is active, it heavily affects the context, because all article listings are filtered to the currently active topic only. So, the next page could have a simple article list like this:

```
{{ list_articles length="10" ignore_issue="true" ignore_section="true" order="b
ypublishdate desc" }}
```

and it will return the last ten articles that have this topic assigned, regardless of the section and issue.

You can also work with topics independently, by directly specifying the context you want to work with.

```
{{ set_topic name="garlic:en" }} .... {{ unset_topic }}
```

Everything inside set_topic and unset_topic is filtered to that specified topic, of course. Don't forget to unset the topic when you don't need it any more. Also, when you jump to some page with the topic parameter specified, be aware that the topic context affects the whole page with all its elements - main navigation, for example. So, in your included templates, be sure to think side-wide, not only about one page, and if working with articles, be sure that you locally switch off the topic parameter. You can do that using:

```
1. {{ local }}
2. {{ unset_topic }}
3. ..... your code ...
4. {{ /local }}
```

You can also make listings of subtopics. One example that we provide here creates a drop-down menu with selectable topics that will send you to the next page (category-page.tpl) with the selected topic activated.

```
1. <script type="text/JavaScript">
2. function MM_jumpMenu(targ,selObj,restore){
3.     eval(targ+".location='"+selObj.options[selObj.selectedIndex].value+"'
   ");
4.     if (restore) selObj.selectedIndex=0;
5. }
6. </script>
7.
```

```
 9.     <label for="category">Category:</label>
10.     <select id="category" name="category" onChange="MM_jumpMenu('parent',
    this,0)">
11.     {{ set_topic name="Category:en" }}
12.         <option selected>----- choose -----</option>
13.         {{ list_subtopics }}
14.         <option value="{{ uri options="template category-page.tpl" }}">
     {{ $gimme->topic->name }}</option>
15.         {{ /list_subtopics }}
16.     {{ unset_topic }}
17.     </select>
18. </form>
```

In Newscoop you don't need to specify the topic parameter directly in:

```
{{ uri options="template category-page.tpl" }}
```

because the topic is currently active (you are inside the topics list!) and is already part of {{ uri }}.

For the use of topics in SEO strategy, see the chapter on Search Engine Optimization.

SWITCHES

You can use switches in two ways - by setting an 'if' clause

```
1. {{ if $gimme->article->custom_switch_name }}
2.  .... your code ...
3. {{ /if }}
```

or in article lists, using

```
{{ list_articles constraints="custom_switch_name is on" }}
```

The list will then (beside other options that you may set) filter articles to return only those with custom_switch_name switched on.

You can, for example, use custom switches to determine if an article has to appear on the website in a breaking news block, or as an ordinary, less emphasized article. You can find an example of this in the chapter about dynamic page layouts.

Beside custom switches, you can use Newscoop's built-in switches 'On front page is on' and 'On section page is on'. Initially, these switches are meant to be used to determine if an article needs to be shown on the front and/or on the section page. Imagine a publication with sections featuring very many articles; not all articles can show up on the section page, and it would be even harder to link all articles directly from front page. So, these options can help journalists and editors promote the most exciting content in their publication.

Built-in switches can be used in article lists:

```
1. {{ list_articles constraints="onfrontpage is on" }}
2. {{ list_articles constraints="onsection is on" }}
```

or in 'if' clauses:

```
1. {{ if $gimme->article->on_front_page }}
2. {{ if $gimme->article->on_section_page }}
```

KEYWORDS

The Keyword field exists by default in every article, and you can use its content in different ways - by setting if clauses, for example:

```
1. {{ if $gimme->article->has_keyword("organic") }}
2. ...
3. {{ /if }}
```

or in an article list:

```
1. {{ list_articles constraints="keyword organic" }}
2. ...
3. {{ /list_articles }}
```

Such a list will return all articles with the keyword 'organic' assigned. For using keywords in SEO strategy, see the chapter *Search Engine Optimization (SEO)*.

25. MANAGING STATIC PAGES

Newscoop doesn't have a special place for publishing static pages. There are different ways of handling static pages, depending on the requirements of your publication. You can park them in a separate publication, in a separate issue or a dedicated section.

A simple way of handling static pages is by using a specific Article Type - in our case "page" - and place pages in any issue or section you want. Then you can list those static pages like this:

```
1. <ul>
2. {{ list_articles ignore_issue="true" ignore_section="true" constraints="t
   ype is page" }}
3.   <li><a href="{{ uri options="article" }}">{{ $gimme->article->name }}</
   a></li>
4. {{ /list_articles }}
5. </ul>
```

This will list all articles which are of the type "page", regardless of their issue or section.

A more sophisticated approach often seen in Newscoop implementations is to create a special issue, outside of the chronological ordering of "normal" issues. Most often that's issue number 1, with a structure and articles which are different from the rest of the publication.

For example, you could have issue number 1 with the name 'pages', and inside it the sections 'about' and 'legal'. To access the content in your publication, you would set the environment to the special issue, and by enclosing everything in {{ local }}...{{ /local }} you could ensure that general context of parameters is not changed.

```
1. {{ local }}
2. {{ set_issue number="1" }}
3. {{ list_sections }}
```

We have two sections, and want to create two subheads with section names, and then unordered lists with articles inside:

```
1. {{ list_articles constraints="type is page" }}
2.     {{ if $gimme->current_list->at_beginning }}
3.         <h3>{{ $gimme->section->name }}</h3>
4.         <ul>
5.     {{ /if }}
```

The subhead will be printed, and the unordered list opened only if the section has articles inside.

```
        <li><a href="{{ uri options="article" }}">{{ $gimme->article->name }}</
a></li>
```

In the code above, all the names of articles inside the section are made into links pointing to the full article pages.

```
1.         {{ if $gimme->current_list->at_end }}
2.         </ul>
3.         {{ /if }}
```

The code above will close the unordered list, once the last article is reached. Finally, the statements are closed, with the code below.

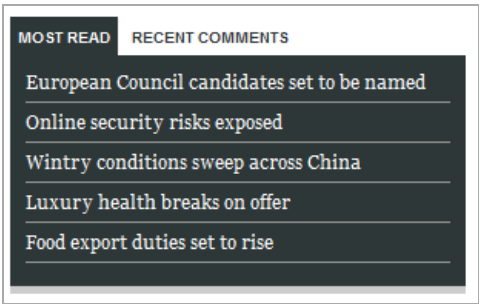
```
1. {{ /list_articles }}
2. {{ /list_sections }}
3. {{ /local }}
```

This way, editors of the publication have the option to independently update the content of these static pages. Keeping them in issue number 1, they can easily jump there with only one click from the issue listing page.

26. LIST OF POPULAR ARTICLES (MOST READ)

For many publications, a listing of the most popular articles is an easy way to keep readers on the site, thus increasing both page views and time on site. Newscoop has an internal statistics mechanism which counts page views (the mechanism counts each URL request as a "read"), and that mechanism can be called in other templates.

In this case, we will call the available functions to make a list of popular articles, which can then be arranged on a page like the screenshot below.



The following code snippet is taken from The Journal's front page, from the sub-template **set_thejournal/_tpl/front_tabs.tpl**

It is displayed inside a jQuery tab which displays comments together with the most read articles, which explains why the CSS tags are the same for both. Read the chapter *List of latest comments* for instructions on how to make that template, and read the chapter on *Tabs with jQuery* for instructions on how to make the tab.

This will do the following:

- Assign the styling for the tab box
- Set the issue to the current one
- Return a list of five articles, ordered by popularity in descending order
- Set a constraint for the article type of 'news'
- Get the URI for the article
- Display the article name

```
1.  <div id="tabs" class="block">
2.    <ul class="idTabs wrap tabs">
3.      <li><a class="selected" href="#commented">Most Read</a></li>
4.      <li><a href="#recentcomments">Recent Comments</a></li>
5.    </ul>
6.    <div class="inside">
7.      <ul style="display: block;" id="commented">
8.  {{ local }}
9.  {{ set_current_issue }}
10. {{ list_articles length="5" order="bypopularity desc" constraints="type i
    s news" }}
11.     <li><a href="{{ uri options="article" }}">{{ $gimme->article->name }}
    </a></li>
12. {{ /list_articles }}
13. {{ /local }}
```

Here is another example, which will tell Newscoop to do the following:

- List the ten most popular articles
- Those articles must have the article type of 'article'
- They must be from sections numbered between 20-300
- They must only be from the last five issues
- They will be ordered in descending order according to their popularity

```
1. {{ list_articles length="10" constraints="type is article section greater
    19 section smaller 301 issue greater `$gimme->issue->number-5`" order="b
   ypopularity desc" }}
```

Note the use of the backtick character in the code above. The next snippet will list the ten most popular articles in the last seven days. It will:

- Assign a variable for the date that is seven days in the past
- List the ten most popular articles in descending order
- Set constraints so that the dates of the articles returned are within the time we set in the variable
- Return articles regardless of issue and section

```
1. {{assign var="xdate" value="-7 days"|date_format:"%Y-%m-%d"}}
2. {{list_articles length="10" order="bypopularity desc" constraints="publis
   h_date greater $xdate reads greater 0" ignore_issue="true" ignore_section
   ="true"}}
```

The following example will:

- Assign a variable for a date one month in the past
- List the ten most popular articles in descending order
- Set constraints where the article type is 'article,' and the publish date is within the time we set in the variable
- Return articles regardless of issue
- Return articles regardless of section

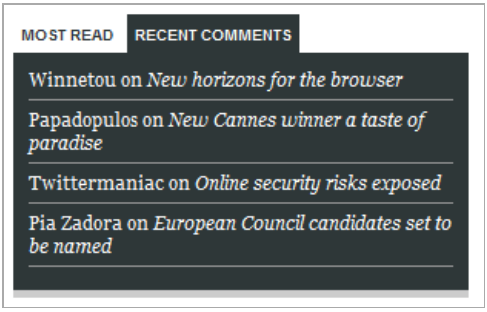
```
1. {{assign var="xdate" value="-1 month"|date_format:"%Y-%m-%d"}}
2. {{list_articles length="10" order="bypopularity desc" constraints="type i
   s article publish_date greater $xdate reads greater 0" ignore_issue="true
   " ignore_section="true"}}
```

27. LIST OF LATEST COMMENTS

The Newscoop internal statistics mechanism keeps track of the latest comments, and this mechanism can be used to create a list of the ten most recent comments on the site for all articles. The following code snippet is taken from The Journal's front page, from the sub-template **set_thejournal/_tpl/front_tabs.tpl**

It is displayed inside a jQuery tab which displays comments together with most read articles (read the chapter *List of popular articles (most read)* for instructions on how to make that template).

Here is a screenshot:



This code snippet will:

- Set the current issue as the active environment
- List five articles, ordered by the latest comment, in descending order
- Set a constraint so that the article type is 'news'
- Apply a different style for the recent comments
- List a single comment, by date
- List the nickname given by a commenter on a comment
- Combine that with the name of the article they're commenting on
- Make that article name appear in italic

```
1. <ul>
2. {{ local }}
3. {{ set_current_issue }}
4. {{ list_articles length="5" order="byLastComment desc" constraints="type
   is news" }}
5.     <li class="recentcomments">{{ list_article_comments length="1" order=
   "bydate desc"}}{{ $gimme->comment->nickname }}{{ /list_article_comments }
   } on <a href="{{ uri options="article" }}" style="font-style: italic">{{
   $gimme->article->name }}</a></li>
6. {{ /list_articles }}
7. {{ /local }}
8. </ul>
```

Here is another example which will do the following:

- List the ten most recent comments, regardless of article
- Order the comments by date, in descending order

- Provide the URI for the comment and the article's name
- Return the count of article comments

- Return an excerpt of the article comments, truncated to 400 characters
- Return the date and time the comment was posted

```
1. {{list_article_comments length="10" ignore_article="true" order="byDate d
   esc"}}
2.     <a href="{{uri}}#comments">{{$gimme->article->name}}</a><sup>{{$gimme
   ->article->comment_count}}</sup>
3.     <p>{{$gimme->comment->content|truncate:400}} <span>{{$gimme->comment-
   >submit_date|camp_date_format:"%H:%i"}}</span></p>
4. {{/list_article_comments}}
```

Finally, here is a similar, but more limited approach. We will:

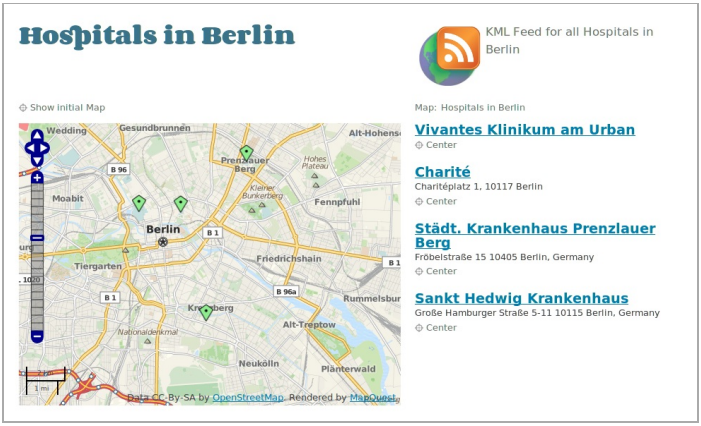
- List the ten most recently-commented articles, ordered by last comment, in descending order, regardless of the issue and section they were published in
- Return the article's URI
- Return the article's name
- Return the count of comments for the article

```
1. {{ list_articles length="10" order="byLastComment desc" ignore_issue="tru
   e" ignore_section="true" }}
2.   <a href="{{ uri }}#comments">{{ $gimme->article->name }}</a><sup>{{ $gi
   mme->article->comment_count }}</sup>
3. {{ /list_articles }}
```

To learn more about listing comments for one specific article and providing the comment form, read the chapter on *Article Comments*.

28. USING MAPS

In Newscoop, journalists can add points of interest (POIs) to a map from the article edit page, and then add additional information which will appear in a pop-up bubble when a reader clicks on the POI. This information can include HTML (including embeds from YouTube, Flickr or Soundcloud).



Map displayed in article. Screenshot taken from "Ushahidi Cooker" template package.

For more on how editors and journalists can create maps and add POIs, see the *Locations* chapter in the manual *Newscoop 3 for Journalists and Editors*, available from:

http://manuals.sourcefabric.org

Creating maps is creating content

Before we dive into how you can use Newscoop's maps and POIs in your templates, here's a bit of our philosophy on geolocation.

Newscoop's mapping is different than just opening a Google Maps embed and passing POIs to Google. This distinction is important, because it has to do with what many publishers believe will make you money now, and in the future as well.

Publishers may be making a strategic mistake when they pass their POIs to commercial map providers like Google. It will be harder to monetize that data because Google and its competitors will be likely to be doing the monetizing instead. We believe that mobile content represents a major revenue opportunity for publishers, and that location-based information is one of the keys to mobile revenue.

This was one of the main reasons we wanted to provide publishers with two important alternatives. The first is that in Newscoop 3's mapping features, publishers keep the points of interest inside their own database, and because they own their databases, publishers can monetize that information as they see fit.

SELECT THE MAP PROVIDER YOU LIKE

The second crucial point in our design of the geolocation features in Newscoop 3 has to do with supporting OpenStreetMap, as a free and open source alternative to commercial online mapping services. OpenStreetMap is to maps as Wikipedia is to encyclopedias - anyone can contribute and improve its accuracy, which means that many parts of the world are far better mapped with OpenStreetMap than they are with commercial maps - countries such as Georgia are completely blank in Google Maps and Bing, for example, but are mapped in detail with OpenStreetMap.

OpenStreetMap also provides publishers with additional strategic importance: The project will probably never compete with publishers in the same way that commercial services are doing and will continue to do. For example, publishers that are too-closely tied to a mapping service provider could fall victim to changes in terms of service. And because publishers using Newscoop 3.5 can choose between Google Maps and OpenStreetMap, their options are open (this switching is enabled by a very cool open source project called OpenLayers, by the way).

Important: OpenStreetMaps' own site and base maps are not intended for large-scale map tile serving; they'll actually throttle back heavy users. It's better to use a service like Mapquest Open (Mapquest's own implementation of OpenStreetMap, at http://open.mapquest.com) which is not limited. Mapquest Open is also supported out-of-the-box in Newscoop as an option.

TEMPLATING AND MAPS

Requirements

In order to display maps, you must include jQuery in the header of your document, with a link like this:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery
/1/jquery.min.js">
```

Displaying a Map within an Article

On the templating side, Newscoop's maps are quite easy to implement, relying on only a couple of templating directives. Here is a simple example from "The Journal" template package which includes a map inside the **article-map.tpl** sub-template:

```
1. {{ if $gimme->article->has_map }}
2.   {{ include file="set_thejournal/_tpl/article-map.tpl" }}
3. {{ else }}
4.   {{ include file="set_thejournal/_tpl/_banner300x250.tpl" }}
5. {{ /if }}
```

In the code above we first validate if the current article has a map, and if so, we include the template rendering the map. If the article does not have a map, we display an advertisement instead.

Inside the article-map.tpl file included in the snippet above, there's a one-line directive:

```
{{ map show_locations_list="true" show_reset_link="Show initial Map" width="300
" height="250" }}
```

This will display a 300px by 250px map, plus the text list of the locations (also called Points Of Interest, or POI's), with a text link which, when clicked, resets the map to the position the editor or journalist originally set.

The resulting map displays the list of points and the map with two points on it. The width and height values used here are not mandatory; if not present, then the global values set in Newscoop preferences are used. You can find more details on this in the chapter *System Preferences* of the *Newscoop 3 for Journalists and Editors* manual.

Listing Article Locations

In the previous example, you saw that it's possible to display the list of locations together with the map, but it's a simple list. What if you want to display a list with some more data about each location? You might need to publish that data even without displaying the graphical map. For this purpose, Newscoop provides a special function *list_article_locations*.

We are now going to display a more detailed list of locations for the same article, with the following code:

```
1.  {{ map show_locations_list="false" show_reset_link="Show initial Map" wid
    th="300" height="250" }}
2.  {{ list_article_locations }}
3.    {{ if $gimme->location->enabled }}
4.      <p>
5.        Location Name: {{ $gimme->location->name }}<br />
6.        Geo Position: {{ $gimme->location->longitude }},
7.                      {{ $gimme->location->latitude }}<br />
8.        Description: {{ $gimme->location->text }}<br />
9.      </p>
10.   {{ /if }}
11. {{ /list_article_locations }}
```

And the output looks like this:

Coming soon in Newscoop

Geolocation was recently introduced with the release of Newscoop 3.5.0, so it is a relatively new feature. It is already powerful, and will become even better as we get more feedback from web developers. Important improvements are being developed for the upcoming releases, most of them under the concept of *Multi-maps*.

Currently, when you are editing an article in the Newscoop administration interface you have the option of adding a map, and then you can add as many locations (point of interests) as you want. All those locations are stored in the database and can be displayed in article pages, but only together with the article those points are related to.

The concept of multi-maps goes far beyond this. You will be able to set a map using several different options, grouping multiple locations regardless of which articles they belong to. It is called multi-maps because a single map will be able to display locations from multiple articles, but also because in a single article you will be able to display multiple maps.

It will be possible to do advanced stuff by defining a map specifying, for example:

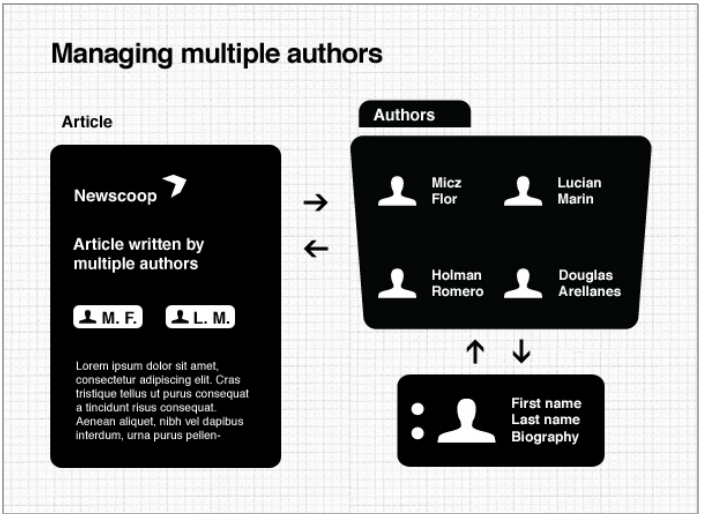
- from which articles you want to display locations, with all those points in a single map
- one or more topics, then Newscoop will find all articles connected to those topics and display all the locations from those articles, again in a single map
- the area (rectangle, polygon). Newscoop will look for all the locations matching this criteria and they will be displayed in a map

It will be also possible to select a map, and then display the list of all the articles the journalists have written corresponding to the locations in that map.

Sounds exciting? Actually, it is!

29. MANAGING MULTIPLE AUTHORS AND ARTICLES

Newscoop has a built in author management tool in which you can create and edit author accounts, with biographies, pictures and other information. You can also create different kinds of author accounts, for photographers, translators, researchers or other contributors - multiple authors can be assigned to a single article.



The following sample code will explain how to print author information when displaying an article. You can print an author's last and first name, their image URL, full name, author type (photographer, etc.), their email address and biographical text.

```
1.  <ul>
2.  {{ list_article_authors }}
3.  <li style="clear: left;">
4.  <strong>{{ $gimme->author->last_name }}</strong>, {{ $gimme->author->firs
    t_name }}<br />
5.  <img src="{{ $gimme->author->picture->imageurl }}" align="left" width="66
    px">
6.  {{ $gimme->author->name }}
7.  ({{ $gimme->author->type }}):
8.  <a href="mailto:{{ $gimme->author->email }}">{{ $gimme->author->email }}<
    /a>
9.  {{ $gimme->author->biography->text }}
10. </li>
11. {{ /list_article_authors }}
12. </ul>
```

30. SUBSCRIPTIONS, REGISTRATION AND CONTENT ACCESS

Newscoop templates can be constructed in a way that they deliver content depending on the status of the user (logged in or not) and their subscription (does the user have access to this article, section, or language?)

If you decide to use subscriptions, this chapter explains how control access to the content, and how to set up the registration process and forms. In a previous chapter we discussed the strategy of using subscriptions in your publication. Further below, you can find a quick introduction to managing subscriptions through the administration interface.

Using a subscription mechanism allows you to manage content access:

- Some content is accessible only to those users who are logged in
- An additional requirement for paid subscriptions may be that the reader has to have a valid subscription for the content they are attempting to access

Journalists and editors then need to switch off the 'Visible to non-subscribers' option on the article edit screen, in order to restrict access to the article.

CONTROLLING CONTENT AND FUNCTIONALITY ACCESS

You can check if the reader is logged in with $gimme->user->logged_in:

```
1. {{ if $gimme->user->logged_in }}
2.   {{ include file="_tpl/article-comments.tpl" }}
3. {{ else }}
4.   Register and log in to comment
5. {{ /if }}
```

You can check if the user has access to content with $gimme->article->content_accessible. What you do with this information is up to you. You can still deliver the same content if you want, or you can only deliver parts of it. Here are two examples to illustrate this. First, truncate the article for readers who are not allowed access yet:

```
1. {{ if $gimme->article->content_accessible }}
2.   {{ include file="_tpl/article-fullcontent.tpl" }}
3. {{ else }}
4.    {{ $gimme->article->full_text|truncate:300 }}
5.    Buy a subscription today to get full access!
6. {{ /if }}
```

Or display relevant information in classified ads to readers who have subscribed, like this:

```
1. {{ $gimme->article->ad_text }}<br/>Contact:
2. {{ if $gimme->article->content_accessible }}
3.   <a href="mailto:{{ $gimme->article->ad_email }}">{{ $gimme->article->ad
   _email }}</a>
4. {{ else }}
5.   Subscribe for contact information.
6. {{ /if }}
```

REGISTRATION, LOGIN AND SUBSCRIPTION TEMPLATES

Your job gets much more complicated if you need to provide the entire functionality circle:

- Login option for your publication's existing subscribers
- Register option for new subscribers
- Account page for users to check or change their personal information, including changing their password
- Option to resend lost or forgotten passwords
- Logout option

If you're also working with paid/trial subscriptions, it's not enough for the reader just to be registered; they may be required to have a valid subscription to some, or all, sections or publications. You will need to develop functionality which:

- ensures that the reader is able to choose what they want to subscribe to (which language if more than one is available; which sections; which publications)
- gives the reader information on what they can choose, and what the price will be

Login and Registration Box

The first thing that a reader needs is the option to login or register. This can be displayed in a box placed somewhere near the top of your publication's pages.

The Newscoop template for this is:

```
1.  <div class="logintop">
2.  {{ if ! $gimme->user->logged_in }}
3.      {{ login_form submit_button="login" }}
4.          <p class="fields"><a href="{{ uri options="template register.tpl"
     }}">Register</a> | {{ if $gimme->login_action->is_error }}<span >{{ $gim
    me->login_action->error_message }}!</span>{{ else }}Sign in:{{ /if }}
5.          <label for="uname"></label> {{ camp_edit object="login" attribute
    ="uname" html_code="value=\"username\"" }}
6.          <label for="uname"></label> {{camp_edit object="login" attribute=
    "password" html_code="value=\"password\"" }}
7.      {{ /login_form }}</p>
8.  {{ else }}<form><p class="fields">Welcome, <a href="{{ uri options="templ
    ate register.tpl" }}">{{ $gimme->user->name }}</a> | <a href="?logout=tru
    e">logout</a></p></form>
9.  {{ /if }}
10. </div><!-- /.login-top -->
```

This template will check to see if the user is already logged in, and if they are, it will display 'Welcome, <Full Name>', where Full Name is also a link to the page where the reader can review their personal information. Also, an option for logging out is displayed.

Logging out works by generating a new URL parameter when the link is clicked:

```
<a href="?logout=true">logout</a>
```

This will reload the current page, but the hook is in the _html-head.tpl template. When this parameter in the URL is provided, the currently logged-in user is logged out. Here is the code:

```
1.  {{ if $gimme->url->get_parameter('logout') == 'true' }}
2.  <META HTTP-EQUIV="Set-Cookie" CONTENT="LoginUserId=; path=/">
```

```
3. <META HTTP-EQUIV="Set-Cookie" CONTENT="LoginUserKey=; path=/">
4. {{ $gimme->url->reset_parameter('logout') }}
5. <META HTTP-EQUIV="Refresh" content="0;url={{ uri }}">
6. {{ /if }}
```

Now, for some more serious stuff: If the user is not logged in, this template provides the options for logging in or registering. Registering calls the register.tpl template (the same one is used for modifying an existing reader's data).

Registration Template register.tpl

First, here's the whole code:

```
1. {{ if !$gimme->edit_user_action->defined && !$gimme->edit_subscription_action->defined }}
2.     {{ include file="_tpl/user-form.tpl" }}
3. {{ /if }}
4.
5. {{ if $gimme->edit_user_action->defined && $gimme->edit_user_action->is_error }}
6.     <h5>Error registering your account: {{ $gimme->edit_user_action->error_message }}</h5>
7.     {{ include file="_tpl/user-form.tpl" }}
8. {{ /if }}
9.
10. {{ if $gimme->edit_user_action->defined && $gimme->edit_user_action->ok }}
11.     {{ if $gimme->edit_user_action->type == "add" }}
12.         <h5>Your user account was successfully added. You will soon receive a confirmation about how to access your subscription.</h5>
13.     {{else }}
14.         <h5>User data successfully added/modified</h5>
15.     {{ /if }}
16. {{ /if }}
```

The code should be self-explanatory - it basically checks whether it should load the user form immediately (if no previous action was taken, it means that the reader wants to perform a new account registration), and additionally it checks to see if an error was generated in the registration process. Finally, if the previous action was a successful registration or account modification, the template will send the appropriate message.

Registration and Profile Form: user-form.tpl

The template called in the process of registration or account modification is user-form.tpl. We're using a table in this case because it's faster to make, but you can create your own CSS-based solution if you prefer:

```
1. {{ user_form template="subscription.tpl" submit_button="submit" }}
2. <p style="margin: 15px 0">Please fill in the following form in order to create the subscription account.</p>
3.     <table id="contact" cellspacing="0" cellpadding="0">
4.         <tr>
5.             <td><label for="contact-name">Full name:</label></td>
6.             <td>{{ camp_edit object="user" attribute="name" html_code="class=\"tablefield widerone\" id=\"contact-name\"" }}</td>
7.         </tr>
8.         <tr>
9.             <td><label for="contact-email">E-mail:</label></td>
10.            <td>{{ camp_edit object="user" attribute="email" html_code="class=\"tablefield widerone\" id=\"contact-email\"" }}</td>
11.        </tr>
12.        <tr>
13.            <td><label for="contact-uname">Username:</label></td>
```

```
14.             <td>{{ camp_edit object="user" attribute="uname" html_code="c
    lass=\"tablefield widerone\" id=\"contact-uname\"" }}</td>
15.         </tr>
16. {{ if ! $gimme->user->logged_in }}
17.         <tr>
18.             <td><label for="contact-password">Password:</label></td>
19.             <td>{{ camp_edit object="user" attribute="password" html_code
    ="class=\"tablefield widerone\" id=\"contact-password\"" }}</td>
20.         </tr>
21.         <tr>
22.             <td><label for="contact-passwordagain">Password (again):</lab
    el></td>
23.             <td>{{ camp_edit object="user" attribute="passwordagain" html
    _code="class=\"tablefield widerone\" id=\"contact-passwordagain\"" }}</td
    >
24.         </tr>
25. {{ /if }}
26.         <tr>
27.             <td><label for="contact-city">City, Country:</label></td>
28.             <td>{{ camp_edit object="user" attribute="city" html_code="cl
    ass=\"tablefield widerone\" id=\"contact-city\"" }}</td>
29.         </tr>
30.         <tr>
31.             <td><label for="contact-phone">Phone:</label></td>
32.             <td>{{ camp_edit object="user" attribute="phone" html_code="c
    lass=\"tablefield widerone\" id=\"contact-phone\"" }}</td>
33.         </tr>
34.         <tr>
35.             <td><label for="second_phone">Phone (cell):</label></td>
36.             <td>{{ camp_edit object="user" attribute="second_phone" html_
    code="class=\"tablefield widerone\" id=\"second_phone\"" }}</td>
37.         </tr>
38.         <input type="hidden" name="SubsType" value="paid" />
            </table>
39.
40.     <div id="submitformdiv">
41.         {{ /user_form }}
42.     </div>
43. {{ if $gimme->user->logged_in }}<p style="margin: 15px 0"> To change your
     password, go <a href="{{ uri options="template user-chgpass.tpl" }}">her
    e</a></p>
44. {{ /if }}
```

This is the universal user form which, in the case of user registration, also includes fields to define and double-check the user's password. In the other case, when a user is logged in when submitting this form, it means that the user is modifying their data and not creating a new user. The option for changing the password is then offered in a separate template (so as not to mess with changing the other data).

After registering or modifying the data, the user is redirected to the subscribe.tpl template, where the final step of creating or changing the subscription is performed.

In this example we are presuming that your publication intends to work with paid subscriptions, so we have one hidden parameter:

```
<input type="hidden" name="SubsType" value="paid" />
```

You can change it to work with trial subscriptions as well. Or, you can create a drop-down menu so the reader can choose what sort of subscription they wants to have:

```
1. Subscription type:
2.     <select name="SubsType">
3.         <option value="trial">Trial</option>
4.         <option value="paid">Paid</option>
5.     </select>
```

Please note that this Newscoop functionality isn't payment processing; it is for the system's internal purposes to determine whether or not the reader is subscribed to the content. Action is required from the site's staff to approve or decline the subscription, based on the reader's payment data.

Subscription Areas: subscription.tpl

The final step in this process, after creating a new user, is to define the content the user is interested in registering for. In Newscoop, publishers can set different prices for different languages, sections, or publications, and you can offer potential subscribers either a global publication subscription, or subscription by sections.

But before offering these forms, you need to check if everything went well, or if there were some errors... So here is the subscription.tpl template:

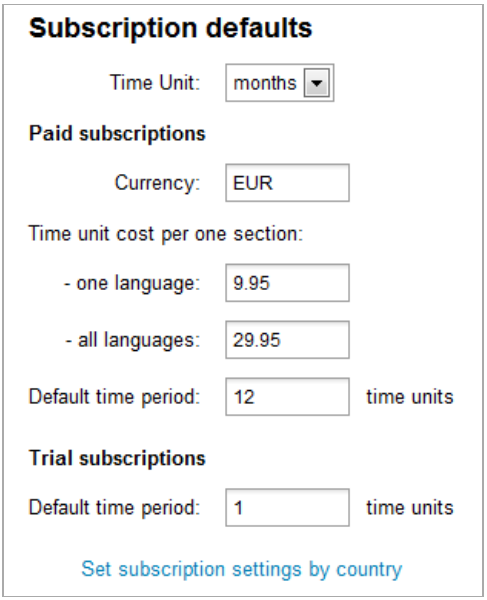
```
1. {{* no user form submitted, no subscription form submitted: display the u
   ser form *}}
2. {{ if !$gimme->edit_user_action->defined
3.       && !$gimme->edit_subscription_action->defined }}
4.           {{ include file="_tpl/user-form.tpl" }}
5. {{ /if }}
6. {{* user form submitted with errors: display the error and the user form
   *}}
7. {{ if $gimme->edit_user_action->defined
8.       && $gimme->edit_user_action->is_error }}
9.           <h5 style="margin-bottom: 20px">There was an error submitting t
   he account creation form:
10.          {{ $gimme->edit_user_action->error_message }}</h5>
11.          {{ include file="_tpl/user-form.tpl" }}
12. {{ /if }}
13. {{* user form submitted ok: display the subscription form *}}
14. {{ if $gimme->edit_user_action->defined
15.      && $gimme->edit_user_action->ok }}
16.          {{ include file="_tpl/subscription-form.tpl" }}
17. {{ /if }}
18. {{* subscription form submitted with errors: display the error and the su
   bmit form *}}
19. {{ if $gimme->edit_subscription_action->defined
20.      && $gimme->edit_subscription_action->is_error }}
21.          <p style="margin: 15px 0">There was an error submitting the sub
   scription form:
22.          {{ $gimme->edit_subscription_action->error_message }}</p>
23.          {{ include file="_tpl/subscription-form.tpl" }}
24. {{ /if }}
25. {{* subscription form submitted ok: display success message *}}
26. {{ if $gimme->edit_subscription_action->defined
27.      && $gimme->edit_subscription_action->ok }}
28.          <p style="margin: 15px 0">Your subscription was created success
   fully.</p>
29. {{ /if }}
```

The subscription form itself:

```
1. {{ subscription_form type="by_section" total="Total" template="subscripti
   on.tpl" button_html_code="class=\"submitbutton\"" }}
2. <table class="userform">
3.        <tr>
4.            <th colspan="2">Please fill in the following form in order to
    create the subscription.</th>
5.        </tr>
6.        <tr>
7.            <td colspan="2">
8.                Subscription time:
9.                {{ $gimme->publication->subscription_time }} {{ $gimme->p
   ublication->subscription_time_unit }}
```

```
10.             </td>
11.         </tr>
12.         <tr>
13.             <td colspan="2">
14.                 {{ camp_select object="subscription" attribute="alllangua
    ges" }}
15.                 <span class="formtext">Subscribe to all languages</span>
16.             </td>
17.         </tr>
18.         <tr>
19.             <td>Languages:</td>
20.             <td>{{ camp_select object="subscription" attribute="languages
    " }}
21.         </tr>
22.         <tr>
23.                 <td colspan="2">Sections</td>
24.         </tr>
25.         {{ list_sections }}
26.         <tr>
27.                 <td colspan="2">
28.                         {{ camp_select object="subscription" attribute="s
    ection" }}
29.                         <input name="tx_subs{{ $gimme->section->number }}
    " type="hidden" value="{{ $gimme->publication->subscription_time }}">
30.                         {{ $gimme->section->name }}
31.                 </td>
32.         </tr>
33.         {{ /list_sections }}
34. </table>
35. {{ /subscription_form }}
```

All subscription options (price, time, time unit, currency...) are set on the Publication configuration page in Newscoop's administration interface (you can find it in Content -> Publications, then choose the wrench-and-screwdriver 'Configure' icon)



Then on the section level, you can decide to Add section to all subscriptions or to Delete section from all subscriptions.

The following user-chgpass.tpl template is only used to change existing Newscoop readers' passwords.

```
1.  {{ user_form submit_button="submit" template="register.tpl" }}
2.      <table id="contact" cellspacing="0" cellpadding="0">
3.          <tr>
4.              <td><label for="contact-passowrd">Password:</label></td>
5.              <td>{{ camp_edit object="user" attribute="password" html_code
    ="class=\"tablefield widerone\" id=\"contact-password\"" }}</td>
6.          </tr>
7.          <tr>
8.              <td><label for="contact-passowrdagain">Password (again):</lab
    el></td>
9.              <td>{{ camp_edit object="user" attribute="passwordagain" html
    _code="class=\"tablefield widerone\" id=\"contact-passwordagain\"" }}</td
    >
10.         </tr>
11.     </table>
12.     <div style="margin: 15px 0;">
13.         {{ /user_form }}
14.     </div>
```

After changing the password, this template redirects the reader back to the register.tpl template.

At this point, you might be asking yourself: "Why the hell is the closing tag for the user_form set inside the div?" Put down your laser blaster nice and slow, and we'll explain: It's because the closing tag defines the position of the submit button.

31. BROWSER DETECTION AND ROBOTS (SEARCH ENGINES)

For advanced web design, it is helpful to know more about the browser where the site is being displayed. You can also detect if your pages are being grabbed by a search engine robot, and deliver content accordingly. For example, if you are using a subscription model for your publication, you can display more information to the search engine robot than you would to an anonymous reader. This way, the search engines will list your content more descriptively than if they only grabbed the "please subscribe" page.

This chapter will give you some ideas on how to detect the client browser, or search engine robot, and change your templates accordingly. The template reference at the end of this Cookbook provides all the options for the browser object.

This is the kind of information Newscoop can get from a browser or robot:

iPhone browser

- Browser: safari, version 528.16 | Engine: webkit, version 528.18
- Mobile: device: iphone | OS: iphone os | OS version: 3.0
- Bot: no
- Type: mobile = handheld

Google Chrome browser

- Browser: chrome, version 10.0.648.205 | Engine: webkit, version 534.16
- Mobile: no
- Bot: no
- Type: bro = normal browser

Firefox browser

- Browser: firefox, version 3.6.16 | Engine: gecko, version 1.9
- Mobile: no
- Bot: no
- Type: bro = normal browser

Google robot

- Browser: googlebot, version 2.1 | Engine: false, version false
- Mobile: no
- Bot: yes
- Type: bot = web bot

THE EASY CSS WAY: CLASSES IN THE BODY TAG

An easy way to deliver different styles to different browsers is to add classes to the body tag, and then apply changes using CSS. Here some examples of the information you can print in the body tag with Newscoop:

iPhone 3

```
1. <body class="webkit mobile safari iphone iphoneos iphoneos3 iphoneos3-0 s
   afari528 safari528-16" >
```

Firefox 3.6.16

```
<body class="moz bro gecko firefox firefox3 firefox3-6 firefox3-6-16 gecko1 geck
o1-9" >
```

Internet Explorer 8

```
<body class="ie bro msie msie8 msie8-0 trident" >
```

Google Bot 2.1

```
<body class="google bot googlebot" >
```

Chrome 10.0.648.205

```
<body class="webkit bro chrome chrome10 chrome10-0-648-205" >
```

You can get these rich body tags using the following code snippet. Note: {{ textformat wrap=200 }} is used in this example to list all the classes in one line and avoid line breaks. The first line break would be applied after 200 characters - which is beyond what this code will deliver. Throw out what you don't need. You could also place the template in a separate file and use {{ include }}.

```
 1. <body class="{{ textformat wrap=200 }}
 2. {{ $gimme->browser->browser_working }}
 3. {{ $gimme->browser->ua_type }}
 4. {{ $gimme->browser }}
 5. {{* mobile device / OS *}}
 6.   {{ $gimme->browser->mobile_data.0 }}
 7.   {{ $gimme->browser->mobile_data.3|regex_replace:"/\ /":"" }}
 8.   {{ strip }}{{ $gimme->browser->mobile_data.3|regex_replace:"/\ /":"" }}
 9.   {{ $gimme->browser->mobile_data.4|regex_replace:"/[\.][0-9]*/":"" }}{{
    /strip }}
10.   {{ strip }}{{ $gimme->browser->mobile_data.3|regex_replace:"/\ /":"" }}
11.   {{ $gimme->browser->mobile_data.4|regex_replace:"/\./":"-" }}{{ /strip
    }}
12. {{* firefox / gecko *}}
13.   {{ $gimme->browser->moz_data.0 }}
14.   {{ strip }}{{ $gimme->browser->moz_data.0 }}
15.   {{ $gimme->browser->moz_data.1|regex_replace:"/[\.][0-9]*/":"" }}{{ /st
    rip }}
16.   {{ strip }}{{ $gimme->browser->moz_data.0 }}
17.   {{ $gimme->browser->moz_data.1|regex_replace:"/[\.][0-9]*$/":""|regex_r
    eplace:"/\./":"-" }}{{ /strip }}
18.   {{ strip }}{{ $gimme->browser->moz_data.0 }}
19.   {{ $gimme->browser->moz_data.1|regex_replace:"/\./":"-" }}{{ /strip }}
20.   {{ if $gimme->browser->browser_working == "moz" }}
21.   gecko{{ $gimme->browser->moz_data.2|regex_replace:"/[\.][0-9]*/":"" }}
22.   gecko{{ $gimme->browser->moz_data.2|regex_replace:"/\./":"-" }}
23.   {{ /if }}
24. {{* internet explorer *}}
25.   {{ if $gimme->browser->browser_working == "ie" }}
26.     {{ strip }}{{ $gimme->browser->browser_name }}
27.     {{ $gimme->browser->webkit_data.2|regex_replace:"/[\.][0-9]*/":"" }}{
    { /strip }}
28.     {{ strip }}{{ $gimme->browser->browser_name }}
29.     {{ $gimme->browser->webkit_data.2|regex_replace:"/\./":"-" }}{{ /stri
    p }}
30.     {{ if $gimme->browser->webkit_data.2 > 8 }}chakra{{ else }}trident{{
    /if }}
31.   {{ /if }}
32. {{* chrome *}}
33.   {{ if $gimme->browser->browser_working == "webkit" }}
34.     {{ strip }}{{ $gimme->browser->webkit_data.0 }}
```

```
35.     {{ $gimme->browser->webkit_data.1|regex_replace:"/[\.][0-9]*/":"" }}{
    { /strip }}
36.     {{ strip }}{{ $gimme->browser->webkit_data.0 }}
37.     {{ $gimme->browser->webkit_data.1|regex_replace:"/\./":"-" }}{{ /stri
    p }}
38.   {{ /if }}
39. {{ /textformat }}" >
```

FULL TEMPLATE CONTROL: BROWSER_DETECTION.TPL

To collect the information listed at the beginning of this chapter, you can use the sub-template **browser_detection.tpl**. It should be called in the header of your pages, and will return a set of variables that make it easy to manage your templates for different browsers or robots. Here is the code for **browser_detection.tpl**:

```
 1. {{* gecko / firefox ***********************************}}
 2. {{ if $gimme->browser->browser_working == "moz" }}
 3.   {{ assign var="browserdetect_name" value=`$gimme->browser->moz_data.0`
    }}
 4.   {{ assign var="browserdetect_version" value=`$gimme->browser->moz_data.
    1` }}
 5.   {{ assign var="browserdetect_engineversion" value=`$gimme->browser->moz
    _data.2` }}
 6.   {{ assign var="browserdetect_engine" value="gecko" }}
 7. {{ /if }}
 8. {{* webkit / chrome / safari  ***********************************}}
 9. {{ if $gimme->browser->browser_working == "webkit" }}
10.   {{ assign var="browserdetect_engineversion" value=`$gimme->browser->bro
    wser_number` }}
11.   {{ assign var="browserdetect_name" value=`$gimme->browser->webkit_data.
    0` }}
12.   {{ assign var="browserdetect_version" value=`$gimme->browser->webkit_da
    ta.1` }}
13.   {{ assign var="browserdetect_engine" value="webkit" }}
14. {{ /if }}
15. {{* ie / internet explorer  ***********************************}}
16. {{ if $gimme->browser->browser_working == "ie" }}
17.   {{ assign var="browserdetect_name" value=`$gimme->browser->browser_name
    ` }}
18.   {{ assign var="browserdetect_engineversion" value=`$gimme->browser->web
    kit_data.2` }}
19.   {{ assign var="browserdetect_version" value=`$gimme->browser->webkit_da
    ta.2` }}
20.   {{ if $gimme->browser->webkit_data.2 > 8 }}
21.     {{ assign var="browserdetect_engine" value="chakra" }}
22.   {{ else }}
23.     {{ assign var="browserdetect_engine" value="trident" }}
24.   {{ /if }}
25. {{ /if }}
26. {{* bot / search engine  ***********************************}}
27. {{ if $gimme->browser->ua_type == "bot" }}
28.   {{ assign var="browserdetect_name" value=`$gimme->browser->browser_name
    ` }}
29.   {{ assign var="browserdetect_version" value=`$gimme->browser->browser_m
    ath_number` }}
30.   {{ assign var="browserdetect_engine" value="false" }}
31.   {{ assign var="browserdetect_engineversion" value="false" }}
32. {{ /if }}
33. {{* mobile devices ****************************************}}
34. {{ if $gimme->browser->ua_type != "mobile" }}
35.   {{ assign var="browserdetect_mobile_device" value="false" }}
36.   {{ assign var="browserdetect_mobile_os" value="false" }}
37.   {{ assign var="browserdetect_mobile_os_number" value="false" }}
38. {{ else }}
39.   {{ assign var="browserdetect_mobile_device" value=`$gimme->browser->mob
    ile_data.0` }}
```

```
40.   {{ assign var="browserdetect_mobile_os" value=`$gimme->browser->mobile_
      data.3` }}
41.   {{ assign var="browserdetect_mobile_os_number" value=`$gimme->browser->
      mobile_data.4` }}
42. {{ /if }}
```

Save the code snippet above inside the **_tpl** folder of your template package. If you are using "The Journal" template pack, include it like this:

```
{{ include file="set_thejournal/_tpl/browser_detection.tpl" }}
```

If your template package has a different name, adjust the path accordingly. After the template has been called, the information about the browser listed earlier can be displayed by these lines:

```
 1. <ul>
 2. <li>Browser: {{ $browserdetect_name }}, version {{ $browserdetect_version
     }}
 3. | Engine: {{ $browserdetect_engine }}, version {{ $browserdetect_engineve
    rsion }}</li>
 4. <li>Mobile: {{ if $browserdetect_mobile_device == "false" }}no{{ else }}
 5. device: {{ $browserdetect_mobile_device }} |
 6. OS: {{ $browserdetect_mobile_os }} |
 7. OS version: {{ $browserdetect_mobile_os_number }}
 8. {{ /if }}</li>
 9. <li>Bot: {{ if $gimme->browser->ua_type == "bot" }}yes{{ else }}no{{ /if
    }}</li>
10. <li>Type: {{ $gimme->browser->ua_type }} =
11. {{ if $gimme->browser->ua_type == "bot" }}web bot{{ /if }}
12. {{ if $gimme->browser->ua_type == "bro" }}normal browser{{ /if }}
13. {{ if $gimme->browser->ua_type == "bbro" }}simple browser{{ /if }}
14. {{ if $gimme->browser->ua_type == "mobile" }}handheld{{ /if }}
15. {{ if $gimme->browser->ua_type == "dow" }}downloading agent{{ /if }}
16. {{ if $gimme->browser->ua_type == "lib" }}http library{{ /if }}
17. </li>
```

32. MOBILE DEVICE DETECTION AND TEMPLATES

This chapter will give you a quick start into delivering different content to different devices, such as mobile phones. In the chapter on *Browser detection and robots (search engines)*, you will find a more in-depth introduction. The template reference at the end of this Cookbook lists all options for the browser object.

Delivering content to mobile devices is becoming an increasingly important issue for web developers. Whereas devices like tablets handle normal websites well, the majority of mobile phone users benefit from custom templates, delivering content for smaller screens.

Some design issues can be handled using the media="handheld" versus media="screen" property in the link tag:

```
1. <link rel="stylesheet" type="text/css" href="...mobile.css" media="handhe
   ld"/>
2. <link rel="stylesheet" type="text/css" href="...screen.css" media="screen
   "/>
```

However, it is more elegant to serve custom pages for different devices. Among other factors, the amount of data downloaded can be reduced if you don't deliver parts of your page which are not meant for mobile devices - rather than just "hiding" them with CSS.

Here a simple example of how Newscoop can display a sidebar on the page, but only if the client does not use a mobile device:

```
1. {{ if $gimme->browser->ua_type != "mobile" }}
2.   {{ include file="set_setname/_tpl/sidebar.tpl" }}
3. {{ /if }}
```

Change the path of the included file to match your template package. You can use the same logic for calling CSS files in the header:

```
1. {{ if $gimme->browser->ua_type == "mobile" }}
2.     <link href="http://{{ $gimme->publication->site }}/templates/set_thej
   ournal/_css/mobile.css" media="handheld" rel="stylesheet" type="text/css"
    >
3. {{ else }}
4.     <link href="http://{{ $gimme->publication->site }}/templates/set_thej
   ournal/_css/style.css" media="handheld" rel="stylesheet" type="text/css"
   >
5. {{ /if }}
```

If you want to deliver content more specifically for different browsers on mobile devices, read the chapter on browser detection. This will enable you, for example, to display links to you iPhone App only if the device is an iPhone. In a similar manner, you can handle media for different browsers as well. For instance, if the device does not support Flash, don't display the Flash player, but a link to the file.

Note: there might be issues with caching systems. If you encounter a problem with browser detection when testing your Newscoop site, switch off the cache and see if that fixes the problem.

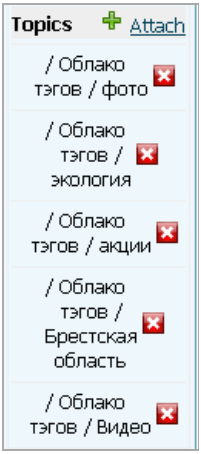
Coming soon in Newscoop

A **Mobile Browser Plugin** is in development, which will give you more detailed information like the size of the display on the client side, allowing for even better control.

33. TAG CLOUDS USING TOPICS

A tag cloud is a collection of words in one place, a depiction of the text content of your publication. Normally, tags are listed alphabetically, and the importance of each tag is shown with font size or colour. To make a tag cloud in Newscoop, we can use **topics** and **sub-topics**. First, you will need to create a list of Newscoop sub-topics, having the parent topic 'tagcloud':

Then, you will just need to attach the required sub-topics to an article:



Then, you need to create a Tag Cloud template. Please note that this template will be doing a lot of listings, so it might be a good idea to put it a cron job to produce the output, and place the result in a file for further inclusion from other templates.

The first thing to do will be to list all sub-topics (tags) of the main topic 'tagcloud' and get minimum and maximum numbers of articles having each topic. Also, we need to set up font-size ranges for the output:

```
1.  {{ local }}
2.  {{ unset_issue }}
3.  {{ unset_section }}
4.  {{ unset_article }}
5.  {{ unset_topic }}
6.  {{ set_topic name="tagcloud:en" }}
7.  {{ assign var="first_good_tag" value=true }}
8.  {{ list_subtopics }}
9.      {{ assign var="posts_count" value="0" }}
10.     {{ list_articles ignore_issue="true" }}
11.         {{ assign var="posts_count" value=$gimme->current_list->count }}
12.     {{ /list_articles }}
13.     {{ if $posts_count > 0 }}
14.         {{ if $first_good_tag }}
15.             {{ assign var="min" value=$posts_count }}
16.             {{ assign var="max" value=$posts_count }}
17.             {{ assign var="first_good_tag" value=false }}
18.         {{ /if }}
19.         {{ if $posts_count > $max }}
20.             {{ assign var="max" value=$posts_count }}
21.         {{ /if }}
22.         {{ if $posts_count < $min }}
23.             {{ assign var="min" value=$posts_count }}
24.         {{ /if }}
25.     {{ /if }}
26. {{ /list_subtopics }}
27.
28. {{ assign var="minSize" value="90" }}
29. {{ assign var="maxSize" value="240" }}
30.
31. {{ assign var="diff_max_min" value="`$max-$min`" }}
32. {{ assign var="diff_maxSize_minSize" value="`$maxSize-$minSize`" }}
```

Next, we create the actual output. List sub-topics, get articles having each sub-topic, and assign the number of times the sub-topic was used:

```
1.  <ul class="tag-cloud">
2.  {{ list_subtopics }}
3.      {{ assign var="posts_count" value="0" }}
4.      {{ list_articles ignore_issue="true" }}
5.          {{ assign var="posts_count" value=$gimme->current_list->count }}
6.          {{ assign var="tag_name" value=$gimme->topic->name }}
7.      {{ /list_articles }}
```

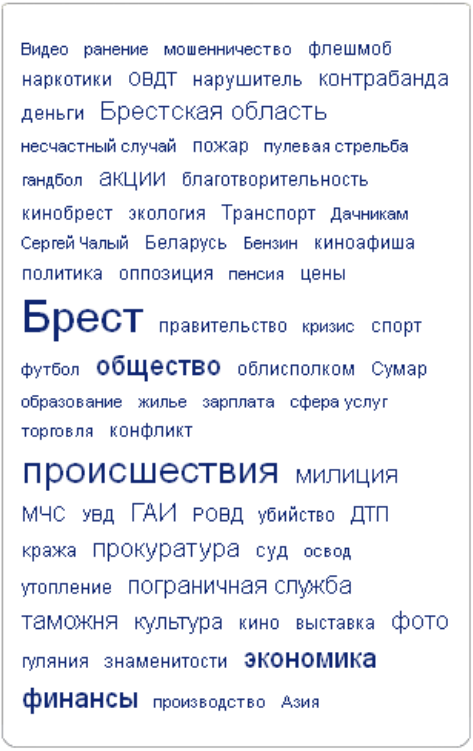
Then we calculate font size for the sub-topic in the tag cloud:

```
1.  {{ if $posts_count > 0 }}
2.      {{ if $min == $max }}
3.          {{ assign var="fontSize" value="`$diff_maxSize_minSize/2+$minSize
    `" }}
4.      {{ else }}
5.      {{ assign var="a" value="`$posts_count-$min`" }}
6.      {{ assign var="b" value="`$a/$diff_max_min`" }}
7.      {{ assign var="fontSize" value="`$b*$diff_maxSize_minSize+$minSize`"
    }}
8.  {{ /if }}
```

And finally output the result:

```
1.      <li>
2.          <span style="}}%"><a href="/?tpl=special_template_to_list_article
    s_based_on_topic.tpl&tag={{ $tag_name }}" title="{{ $posts_count }} artic
    les having topic {{ $tag_name }}">{{ $tag_name }}</a></span>
3.      </li>
4.  {{ /if }}
5.  {{ /list_subtopics }}
6.  </ul>
7.  {{ /local }}
```

As a result, you should get a Tag Cloud like the following screenshot:



Tag cloud created using topics

34. DYNAMIC PAGE LAYOUTS

In this chapter, you'll learn how to create dynamic page layouts that change automatically when a staff user clicks on a check box on the **Article Edit** page.

The example we show here features an Article Type with a custom switch 'breaking_news'. This custom switch can be added to the Article Type in the Newscoop administration interface. The journalist or editor then sees a checkbox for 'breaking_news' in the Article Edit screen, which they will click whenever they consider the story they are working on to be particularly important.

The result of a staff user clicking this checkbox can be detected in the template. Inside *list_articles* a constraint is added, collecting only articles where the custom switch called 'breaking_news' is turned on. If there's an article that fulfills that criteria, then it's listed. If not, the layout remains the same:

```
{{ list_articles length="1" ignore_section="true" order="bypublishdate desc" co
nstraints="breaking_news is on" }}
```

That tells $gimme to list one article with the following constraints:

- Display articles regardless of the section they're in
- Present the articles in descending chronological order according to their published date
- Only display articles where the "breaking_news" custom switch is on

The whole template looks like this:

```
 1. {{ list_articles length="1" ignore_section="true" order="bypublishdate de
    sc" constraints="breaking_news is on" }}
 2. <div id="breakingNews">
 3.     <h3><a href="{{ uri options="article" }}">{{ $gimme->article->name }}
    </a></h3>
 4.     {{ list_article_images length="1" }}
 5.     <div id="breakingNewsLeft">
 6.         <img src="{{ uri options="image width 435" }}" alt="{{ $gimme->ar
    ticle->image->description }}" />
 7.         <p class="footnote">{{ $gimme->article->image->description }} {{
    if $gimme->article->image->photographer }}(Photo: {{ $gimme->article->ima
    ge->photographer }})){{ /if }}</p>
 8.     </div>
 9.     {{ /list_article_images }}
10.     {{ if $gimme->prev_list_empty }}
11.     <div id="breakingNewsLeft">
12.         <img src="/templates/images/breaking-news-{{ $gimme->language->co
    de }}.jpg" alt="Breaking news" />
13.     </div>
14.     {{ /if }}
15.     <div id="breakingNewsCenter">
16.         {{ $gimme->article->publish_date|camp_date_format:"%M %e, %Y" }}
17.         <p>{{ $gimme->article->deck }}</p>
18.     </div>
19. </div>
20. {{ /list_articles }}
```

For more on using custom switches, see the chapter titled *Topics, switches, keywords to structure content.*

35. WORKING WITH IMAGES

It's an understatement to say that images are crucial to news organizations. Because of their central role, Newscoop has a powerful and user-friendly media archive for storing, retrieving and reusing images in the administration interface. Newscoop's template language can access images in a number of different ways, and the output can be sent to web pages, to jQuery slideshows or other channels (see the chapter *Image galleries with jQuery* for more on that).

In this chapter, we'll look at how Newscoop templates work with images, image attributes and metadata. According to the Newscoop template reference, "The *image* object is usually initialized inside a list of article images or a list of images. It is not initialized at the beginning of the template and cannot be initialized by other Newscoop functions." In other words, images must be used through lists or through articles.

You can make use of images through the *article* object, like when the current article has one or more images attached. One more place where you can use an *image* object is with *author* objects. We'll go into these options in detail below. What you can't do at this point is use an image object as a single isolated element, because it has to be set first by any of the means mentioned above.

DISPLAYING AN ARTICLE IMAGE

Let's start with a simple example. You have an article with only one image, and you want to display the image when the article is requested.

You can point to an image in two ways, either:

```
<img src="{{ $gimme->article->image1->imageurl }}" alt="{{ $gimme->article->imag
e->caption }}" />
```

or:

```
<img src="{{ uri options="image 1" }}" alt="{{ $gimme->article->image->caption }
}" />
```

The following code snippet does more, it:

- Gets the URL for an image attached to an article
- Puts the image caption (the description) into the ALT tag
- Puts the photo description into the title tag
- Displays the photographer's name
- Displays the image caption (the description)

```
1. <img src="{{ $gimme->article->image->imageurl }}"
2.      alt="{{ $gimme->article->image->caption }}"
3.    title="{{ $gimme->article->image->description }}"  /><br />
4.        by {{ $gimme->article->image->photographer }}<br />
5.           {{ $gimme->article->image->description }}
```

An example of what similar code looks like in the browser is below (taken from the "The Journal" template pack):

Sibling bottles
Photographer: Sourcefabric

You can even validate whether the article has an image or not. The value passed in parentheses correspond to the image number you assign to the image when attaching it to the article.

```
1. {{ if $gimme->article->has_image(1) }}
2.     <img src="{{ $gimme->article->image->imageurl }}" />
3. {{ /if }}
```

It's also possible to access the image directly by the index number. If an image with that given index does not exist, then an empty *image* object is returned, and nothing will be displayed. It is good practice to first validate whether the requested image exists or not, but you already know how to do that!

```
<img src="{{ $gimme->article->image5->imageurl }}" />
```

There are more image properties you can display; we've already used some like *imageurl*, *photographer* and *description*, but there'll be more in the following examples. You can read the entire list in the chapter Template Objects -> Image of the Newscoop Template Reference.

LISTING ALL IMAGES ATTACHED TO AN ARTICLE

This is basically the same as we did before, but within a list of article images. Let's see some code:

```
1. {{ list_article_images }}
2.     <li>
3.         <img src="{{ $gimme->article->image->imageurl }}" /><br />
4.         Caption: {{ $gimme->article->image->caption }}<br />
5.         By: {{ $gimme->article->image->photographer }}
6.     </li>
7. {{ /list_article_images }}
```

There's no need to use image indexes, because the list provides iteration over all images attached to the article.

LISTING IMAGES FROM THE MEDIA ARCHIVE

Now the fun begins :-)

You already know how to work with image objects and how to list images attached to articles; this is very useful and will allow you to use image content all over your site.

But in specific cases you'll probably want to display images not necessarily related to articles. Remember that Newscoop provides a Media Archive (read more about it in the *Newscoop 3 for Journalists and Editors* manual), and every image you attached to an article is stored there.

The Newscoop template language provides a function to build lists of images according to different criteria. Let's say you want to build a list of images from a specific photographer. This code snippet can:

- Get a list of images where the photographer name is John Doe
- Order the images by the last update
- Get the images themselves based on their URLs
- Get the image captions (descriptions)

```
1. {{ list_images photographer=" John\ Doe" order="byLastUpdate" }}
2.     <img src="{{ $gimme->image->imageurl }}" /><br />
3.     <p>{{ $gimme->image->description }}</p>
4. {{ /list_images }}
```

Now here's list of images where the string "Prague" is present:

```
1. {{ list_images caption_like="Prague" order="byPhotographer" }}
2.     <img src="{{ $gimme->image->imageurl }}" /><br />
3.     <p>{{ $gimme->image->caption }}</p>
4. {{ /list_images }}
```

There are many other criteria you can use. A detailed list can be found in the Newscoop Template Reference in the *List Images* chapter.

THUMBNAILS

Newscoop automatically generates a 64-pixel-wide thumbnail for every image when the image file is uploaded into the Media Archive. Displaying a thumbnail is as easy as this:

```
<img src="{{ $gimme->image->thumbnailurl }} />
```

AUTHOR PICTURE

The *author* object has the property *picture*, which is an *image* object, so that you can use it exactly as an article image. There's more on this in the chapter *Managing multiple authors and articles* in this Cookbook.

SCALING IN PERCENT AND ABSOLUTE SIZE

Images in Newscoop are only stored in two different sizes: the original size and the automatically generated thumbnail. Usually it is necessary to display those images in a different size than those two available, so Newscoop provides a simple way to do this on the fly:

```
<img src="{{ $gimme->image->imageurl }}&ImageRatio=70" />
```

Here we just specified a ratio equal to 70, which means the image will be resized to 70 percent of its original size. You also have the option to provide an specific width or height value.

```
<img src="{{ $gimme->image->imageurl }}&ImageWidth=350&ImageHeight=310" />
```

Any image provided by *$gimme* can be scaled as shown above, including the thumbnail and author pictures. The logic of how Newscoop processes the values when both width and height are provided (and even the ratio) at the same time is explained, with more examples, in the wiki page: http://wiki.sourcefabric.org/x/c4EH

36. PAGINATION OF LISTS AND LONG ARTICLES

Pagination means dividing content into discrete pages, each displaying a page number. When a long list of search results is divided into more than one page, allowing the reader to proceed from page to page, that's pagination.

In this chapter, you will learn an advanced method for list pagination, and how to use subtitles to break up long articles.

PAGINATION OF A LONG ARTICLE LIST

Sometimes you might want to split a long list of articles into pages. You might do this on Section pages to show all articles in that particular section, without requiring the reader to scroll down too much.

To implement pagination we need to add an extra page parameter to a URL, something like the following example:

```
http://site_name/language_code/issue_number/section_number/?page=number
```

Then we can use the GET parameter to check the page we are on. We will use Smarty directly to check GLOBAL variables:

```
{{ if $smarty.get.page }}{{ assign var="page" value=`$smarty.get.page` }}{{ els
e }}{{ assign var="page" value="1" }}{{ /if }}
```

To split a list of articles into pages we will use a column parameter, which will control how many articles should be shown on a page. In this case, row parameter will be the actual page.

```
... {{ if $gimme->current_list->count > 10 }} ...
```

```
... {{ if $gimme->current_list->row == $page }} ...
```

In this example, we also need to add the ignore_issue parameter to get articles from all issues. The first part of the code, which will show the articles, would be:

```
1. {{ if $smarty.get.page }}
2.     {{ assign var="page" value=`$smarty.get.page }}{{ else }}{{ assign va
   r="page" value="1" }}{{ /if }}
3.     {{ list_articles columns="10" ignore_issue="true" }}
4.     {{ if $gimme->current_list->row == $page }}
5.         <div class="list-articles"><a href="{{ uri options="article" }}">
   {{ $gimme->article->name }}</a>{{ $gimme->article->intro }}</div>
6.     {{ /if }}
7.     {{ /list_articles }}
8.     ...
```

In the second part of the code we will show page numbers, and add the page parameter to links. To do this, we will need to list the articles once again:

```
1. {{ list_articles columns="10" ignore_issue="true" }}
2. {{ if $gimme->current_list->count > 10 }}
3.     {{ if $gimme->current_list->at_beginning }}
4.         <div class="list-page-numbers">Page:
5.     {{ /if }}
6.     {{ if $gimme->current_list->column == 1 }}
```

```
7.          {{ if $gimme->current_list->row == $page }}
8.              <span>{{ $gimme->current_list->row }}</span>
9.          {{ else }}
10.             <a href="{{ uri options="article" }}?page={{ $gimme->current_
    list->row }}">{{ $gimme->current_list->row }}</a>
11.         {{ /if }}
12.     {{ /if }}
13.     {{ if $gimme->current_list->at_end }}
14.         </div>
15.     {{ /if }}
16. {{ /if }}
17. {{ /list_articles }}
```

You will now have a list of articles and a list of page numbers, as shown in the screenshot below. You are free to add additional styles and controls to make your list shine.

SUBTITLES INSIDE LONG ARTICLE FIELDS

You can include subtitles (also known as subheads) in longer articles, to break the article into pages. The journalist uses the WYSIWYG editor, marks the subtitle in the text and selects "campsite_subhead" from the pulldown menu under "styles". A simple list of all subtitles inside the article can be displayed on the article page like this:

```
1. <ul>
2. {{ list_subtitles field_name="full_text" }}
3.     <li>{{ $gimme->subtitle->name }}</li>
4. {{ /list_subtitles }}
5. </ul>
```

Note that inside list_subtitles you need to specify the article field which you are working with, in this case the field full_text. The subtitles listed will all be from this particular article field, rather than any other field that may contain subtitles.

Adding a simple "previous" and "next" navigation from article subtitle to article subtitle can be done like this:

```
1. {{ if $gimme->article->full_text->has_previous_subtitles }}
2.     <a href="{{ uri options="previous_subtitle full_text" }}">Previous</a
   >
3. {{ else }}
4.     Previous
5. {{ /if }}
6. |
7. {{ if $gimme->article->full_text->has_next_subtitles }}
8.     <a href="{{ uri options="next_subtitle full_text" }}">Next</a>
```

```
 9. {{ else }}
10.     Next
11. {{ /if }}
```

This navigation will display active links to "Previous" or "Next" only if there is a previous or next item in the list. Otherwise, the words Previous and Next are not clickable, and are just displayed for design reasons, so the eye does not need to jump back and forth.

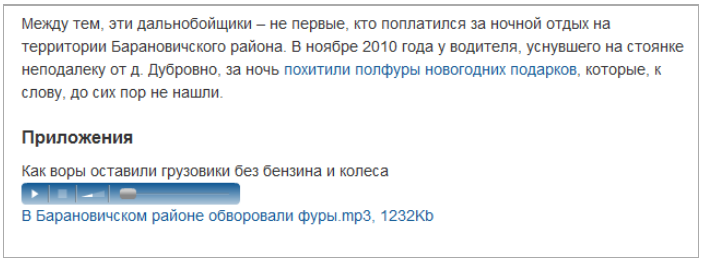
Once you are using subtitles in an article with pagination, the content will no longer be displayed in a single page format. In order to display the entire article in a single page (e.g. for printing) you can use a link with options like this:

```
<a href="{{ uri options="all_subtitles full_text" }}">View entire article</a>
```

37. AUDIO PLAYER FOR ATTACHED MP3 FILES

Newscoop has an easy mechanism for attaching files to articles. In the Article Edit screen, on the right, you can attach files either from the existing file archive or upload new files. In this example we will embed an MP3 player in the page, playing audio files which have been attached to the article.

We offer two different examples of using an embedded Flash player. The first example embeds a player at the end of an article, like this:



Audio player on intex-press.by

The player we're using in this example needs to be placed in this folder:

```
/templates/_swf/player_mp3_maxi.swf
```

Any other player and any other folder location is possible. You only need to change the path in the code accordingly.

This code snippet will:

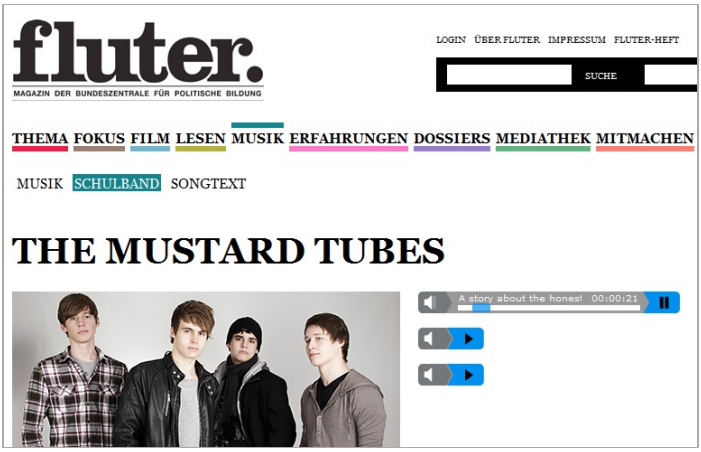
- Check to see if the article has any attachments; if it does, it will list them
- Check to see if the attachment has an extension of "mp3"
- If it is mp3, it will:
  - Display the attachment's file description
  - Call the Flash player and set its values
  - Give the Flash player the URI for the article attachment
  - Create a download link for the article attachment with its URI, file name and size

```
1.  {{ if $gimme->article->has_attachments }}
2.  {{ list_article_attachments }}
3.  {{ if $gimme->attachment->extension == "mp3" }}
4.      {{ $gimme->attachment->description }}<br />
5.      <object height="20" width="200" data="/templates/_swf/player_mp3_maxi
    .swf" type="application/x-shockwave-flash" align="top">
6.          <param value="/templates/_swf/player_mp3_maxi.swf" name="movie" /
    >
7.          <param name="wmode" value="opaque" />
8.          <param value="mp3={{ uri options="articleattachment" }}&amp;shows
    top=1&amp;showvolume=1&amp;buttonwidth=25&amp;sliderwidth=15&amp;bgcolor1
    =5EA65E&amp;bgcolor2=365E36" name="FlashVars" />
9.      </object><br />
10.     <a href="{{ uri options="articleattachment" }}" />{{ $gimme->attachme
    nt->file_name }}, {{ $gimme->attachment->size_kb }}Kb</a>
```

```
11.  {{/if}}
12.  {{/list_article_attachments}}
13.  {{/if}}
```

USING MULTIPLE PLAYERS

The second example comes from Fluter.de, and involves keywords and multiple players. Here is a screenshot of how the players appear on the page:



Audio players on fluter.de

This example does the following:

- Uses an IF statement to see if the article has the keyword of 'mp3podcast1'
- If it does, Newscoop calls a Flash player
- Newscoop passes the Flash player the URL of the MP3 file

The audio files on Fluter.de are not article attachments. In the editorial workflow of that publication, the audio files are created separately and uploaded to a special location on the server. The code example below will point to the location:

http://www.fluter.de/medien/podcast_audio/

There, the audio files are sorted in subfolders by issue number, which we can retrieve using {{ $gimme->issue->number }}. Each file name contains a combination of issue number and article number, and finishes with the track number, so we can retrieve a specific file using:

```
podcast_mp3_{{ $gimme->issue->number }}_{{ $gimme->article->number }}_f1.mp3
```

The publication editor adds a keyword to the article containing the number of the audio files; mp3podcast1 means one file, mp3podcast2 two files, and so on. Depending on this keyword, the template will create one or more players. Here is the code for one player:

```
1.  {{ if $gimme->article->has_keyword(mp3podcast1) }}
2.  <object type="application/x-shockwave-flash" data="/medien/player/player.
    swf" width="290" height="24" id="audioplayer1"><param name="movie" value=
    "http://www.fluter.de/audio-player/player.swf" />
```

```
3.      <param name="FlashVars" value="playerID=1&amp;bg=0x999999&amp;leftbg=
    0x74787A&amp;lefticon=0xffffff&amp;rightbg=0x0090f0&amp;rightbghover=0x74
    787A&amp;righticon=0x000000&amp;righticonhover=0x0090f0&amp;text=0xffffff
    &amp;slider=0x0090f0&amp;track=0xFFFFFF&amp;border=0xffffff&amp;loader=0x
    ffffff&amp;soundFile=http%3A%2F%2Fwww.fluter.de%2Fmedien%2Fpodcast_audio%
    2F{{ $gimme->issue->number }}%2Fpodcast_mp3_{{ $gimme->issue->number }}_{
    { $gimme->article->number }}_f1.mp3" />
4.      <param name="quality" value="high" />
5.      <param name="menu" value="false" />
6.      <param name="bgcolor" value="#ffffff" />
7.  </object>
8.  {{ /if }}
```

This approach can be used for any file attachments; you can apply similar methods for PDF attachments or movies, for example. See the section *Article object and attachment, comment, location* in the Template Reference for more information on working with article attachments.

38. RSS, SITEMAP, KML AND XML

The Newscoop template engine allows you deliver many kinds of structured content, including HTML, XML, CSV, vCard and more. In this chapter we will explain a few of these formats and how they are created. The process is always the same: develop your business logic, and then wrap the presentation logic around it.

RSS FEED

We will start with producing an RSS feed for syndication of the latest 15 articles from the current section in the publication. First, let's look at the basic structure of an RSS document. The channel tag contains three elements; channel metadata, an image block and an items list:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom" xmlns:media="
   http://search.yahoo.com/mrss/">
3.     <channel>
4.         [channel metadata]
5.         [image block]
6.         [items list]
7.     </channel>
8. </rss>
```

Channel metadata is a required element which can include the channel's name or title, description, language, URL and copyright information.

```
1. <title>{{ $gimme->publication->name }}</title>
2. <link>http://{{ $gimme->publication->site }}</link>
3. <description>{{ $siteinfo.description }}</description>
4. <language>{{ $gimme->language->code }}</language>
5. <copyright>Copyright {{ $smarty.now|date_format:"%Y" }, {{ $gimme->public
   ation->name }}</copyright>
6. <lastBuildDate>{{ $smarty.now|date_format:"%a, %d %b %Y %H:%M:%S" }} +010
   0</lastBuildDate>
7. <generator>Newscoop</generator>
```

The Newscoop snippet above grabs the name and site attributes of the publication, as well as the language code. Also, we make use of Smarty with some special modifiers to display the current date and time in two different formats (for more information on this, please read the Smarty manual).

Image block - the image's URL, title, link, width, and height tags allow RSS viewers to translate the file into HTML. This block is very straight forward, using the site and name attributes of the Newscoop publication.

```
1. <image>
2.     <url>http://{{ $gimme->publication->site }}/templates/classic/img/log
   o-rss.jpg</url>
3.     <title>{{ $gimme->publication->name }}</title>
4.     <link>http://{{ $gimme->publication->site }}</link>
5.     <width>144</width>
6.     <height>19</height>
7. </image>
```

Items list - this is the most important piece of the RSS document, as the actual feeds are generated here. For this purpose we will use the *list_articles* statement with $gimme to request the latest 15 published articles. The list is built using the current language, publication, issue and section from the context.

```
1. {{ list_articles length="15" order="bypublishdate desc" }}
2.     [list of items]
3. {{ /list_articles }}
```

Now we need to define the list of items.

```
1. <item>
2.  <title>{{ $gimme->article->name }}</title>
3.   <link>{{ url options="article" }}</link>
4.    <description>
5.    {{ $gimme->article->intro|strip_tags:false|strip|escape:'html':'utf-8'
    }}
6.    </description>
7.     <category domain="{{ url options="section" }}">
8.     {{ $gimme->section->name }}
9.     </category>
10.    {{ if $gimme->article->author->name }}
11.    <atom:author>
12.    <atom:name>{{ $gimme->article->author->name }}</atom:name>
13.    </atom:author>
14.    {{ /if }}
15.   <pubDate>
16.   {{ $gimme->article->publish_date|date_format:"%a, %d %b %Y %H:%M:%S" }}
    +0100
17.   </pubDate>
18.  <guid isPermaLink="true">{{ url options="article" }}</guid>
19. </item>
```

Let's explain where we get the data from for each item tag:

title: The article name.

link: The article URL from Newscoop.

description: The text of the article introduction, which is a custom field for the particular article type used here. In addition, we use some Smarty modifiers in order to clean up the text, making sure it will be displayed properly.

category: We use the section name here.

atom author: If the article has an author defined, then we display their full name.

pubDate: The publication date of the article, with some special formatting. Please read up on the Smarty *date_format* modifier if you aren't sure about this.

guid: The article URL as global unique identifier for the item.

Finally, this is how the whole template file should look:

```
1. <rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom" xmlns:media="
   http://search.yahoo.com/mrss/">
2.     <channel>
3.         <title>{{ $gimme->publication->name }}</title>
4.         <link>http://{{ $gimme->publication->site }}</link>
5.         <description>{{ $siteinfo.description }}</description>
6.         <language>{{ $gimme->language->code }}</language>
7.         <copyright>Copyright {{ $smarty.now|date_format:"%Y" }}, {{ $gimm
   e->publication->name }}</copyright>
```

```
8.          <lastBuildDate>{{ $smarty.now|date_format:"%a, %d %b %Y %H:%M:%S" }} +0100</lastBuildDate>
9.          <generator>Newscoop</generator>
10.         <image>
11.             <url>http://{{ $gimme->publication->site }}/templates/classic/img/logo-rss.jpg</url>
12.             <title>{{ $gimme->publication->name }}</title>
13.             <link>http://{{ $gimme->publication->site }}</link>
14.             <width>144</width>
15.             <height>19</height>
16.         </image>
17.         <atom:link href="http://{{ $gimme->publication->site }}/templates/feed/index-en.rss" rel="self" type="application/rss+xml" />
18.         {{ list_articles length="20" order="bypublishdate desc" }}
19.         <item>
20.             <title>{{ $gimme->article->name|html_entity_decode|regex_replace:'/&(.*?)quo;/':'&quot;' }}</title>
21.             <link>http://{{ $gimme->publication->site }}/ru/{{ $gimme->issue->number }}/{{ $gimme->section->url_name }}/{{ $gimme->article->number }}</link>
22.             <description>
23.                 {{ $gimme->article->intro|strip_tags:false|strip|escape:'html':'utf-8' }}
24.             </description>
25.             <category domain="http://{{ $gimme->publication->site }}/{{ $gimme->language->code }}/{{ $gimme->issue->number }}/{{ $gimme->section->url_name }}">{{ $gimme->section->name }}</category>
26.             {{ if $gimme->article->author->name }}
27.             <atom:author><atom:name>{{ $gimme->article->author->name }}</atom:name></atom:author>
28.             {{/if}}
29.             <pubDate>{{ $gimme->article->publish_date|date_format:"%a, %d %b %Y %H:%M:%S" }} +0100</pubDate>
30.             <guid isPermaLink="true">http://{{ $gimme->publication->site }}/{{ $gimme->language->code }}/{{ $gimme->issue->number }}/{{ $gimme->section->url_name }}/{{ $gimme->article->number }}</guid>
31.         </item>
32.     {{/list_articles}}
33.     </channel>
34. </rss>
```

How to get it working

There are different ways to achieve this, however what we recommend is to use Google FeedBurner as it provides significant benefits like statistics, social sharing, among others. Read more about FeedBurner at http://feedburner.google.com/.

Now that you already have your RSS template file you need to configure it in FeedBurner. You can do this easily by inputting the URL to the RSS in your site. Usually the URL looks like this, as it is a direct reference to the RSS template file:

http://your.site.url/?tpl=123

FeedBurner will give you back a new "burned" URL that looks something like this:

http://feeds.feedburner.com/YourFeedName

Where *YourFeedName* is the name you input when burning your RSS feed in FeedBurner. Now all what you need to do is to include the following line in the header template of your site.

```
<link rel="alternate" type="application/rss+xml" title="Newscoop News" href="http://feeds.feedburner.com/YourFeedName" />
```

GOOGLE SITEMAPS

The Sitemaps protocol, introduced by Google, allows you to inform search engines about links on your site that are available for crawling. A Sitemap is a file in a specific XML format that lists the URLs of your site, including metadata about each URL carrying data like last updated, the relevance of the resource compared to other URLs in the site, and so on. This allows search engines to index the site in a more optimal way.

How to build a site map depends on the content structure you have designed for your specific site, but whatever it is, $gimme will allow you to generate it.

Goal / Task

To display the Site Map for our particular use case, listing: All articles regardless of language, issue and section, of type news, All articles regardless of language, issue and section, of type show, All subtopics in English and French languages with parent topic equal to Countries, All sections from the current issue in English and French languages.

Implementation

All articles regardless of language, issue and section, of type news.

```
1. {{ list_articles ignore_language="true" ignore_issue="true" ignore_sectio
   n="true" order="bypublishdate desc" constraints="type is news" }}
2. <url>
3.     <loc>http://wadr.org{{ uri options="article" }}</loc>
4.     <lastmod>{{ $gimme->article->publish_date|camp_date_format:"%Y-%m-%d"
    }}</lastmod>
5.     <changefreq>daily</changefreq>
6.     <priority>0.6</priority>
7. </url>
8. {{ /list_articles }}
```

Notice the use of ignore_language, ignore_issue and ignore_section. All these are set to *true* so that those values from the context are ignored when building the list. We use the same approach for the next list.

All articles regardless of language, issue and section, of type show.

```
1. {{ list_articles ignore_issue="true" ignore_section="true" ignore_languag
   e="true" order="bypublishdate desc" constraints="type is show" }}
2. <url>
3.     <loc>http://wadr.org{{ uri options="article" }}</loc>
4.     <lastmod>{{ $gimme->article->publish_date|camp_date_format:"%Y-%m-%d"
    }}</lastmod>
5.     <changefreq>daily</changefreq>
6.     <priority>0.8</priority>
7. </url>
8. {{ /list_articles }}
```

This list is similar to the first one. The only difference is we are requesting articles of a different type: show this time.

All subtopics in the English language with parent topic equal to Countries.

```
1. {{ set_language name="English" }}
2. {{ set_topic name="Countries:en" }}
3. {{ list_subtopics }}
4. <url>
5.     <loc>http://wadr.org/{{ $gimme->language->code }}/?tpid={{ $gimme->to
   pic->identifier }}&amp;tpl=1442/</loc>
```

```
6.      <changefreq>daily</changefreq>
7.      <priority>0.4</priority>
8.  </url>
9.  {{ /list_subtopics }}
```

All subtopics in the French language with parent topic equal to Countries.

```
1.  {{ set_language name="French" }}
2.  {{ set_topic name="Countries:en" }}
3.  {{ list_subtopics }}
4.  <url>
5.      <loc>http://wadr.org/{{ $gimme->language->code }}/?tpid={{ $gimme->to
    pic->identifier }}&amp;tpl=1442/</loc>
6.      <changefreq>daily</changefreq>
7.      <priority>0.4</priority>
8.  </url>
9.  {{ /list_subtopics }}
```

All sections from the current issue in the English language.

```
1.  {{ set_current_issue }}
2.  {{ set_language name="English" }}
3.  {{ list_sections }}
4.  <url>
5.      <loc>http://wadr.org/{{ $gimme->language->code }}/{{ $gimme->issue->u
    rl_name }}/{{ $gimme->section->url_name }}/</loc>
6.      <changefreq>weekly</changefreq>
7.      <priority>0.3</priority>
8.  </url>
9.  {{ /list_sections }}
```

All sections from the current issue in the French language.

```
1.  {{ set_language name="French" }}
2.  {{ list_sections }}
3.  <url>
4.      <loc>http://wadr.org/{{ $gimme->language->code }}/{{ $gimme->issue->u
    rl_name }}/{{ $gimme->section->url_name }}/</loc>
5.      <changefreq>weekly</changefreq>
6.      <priority>0.3</priority>
7.  </url>
8.  {{ /list_sections }}
```

This is how the full Sitemap file should look:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
3.  {{ list_articles ignore_language="true" ignore_issue="true" ignore_sectio
    n="true" order="bypublishdate desc" constraints="type is news" }}
4.      <url>
5.          <loc>http://wadr.org{{ uri options="article" }}</loc>
6.          <lastmod>{{ $gimme->article->publish_date|camp_date_format:"%Y-%m
    -%d" }}</lastmod>
7.          <changefreq>daily</changefreq>
8.          <priority>0.6</priority>
9.      </url>
10. {{ /list_articles }}
11. {{ list_articles ignore_issue="true" ignore_section="true" ignore_languag
    e="true" order="bypublishdate desc" constraints="type is show" }}
12.     <url>
13.         <loc>http://wadr.org{{ uri options="article" }}</loc>
14.         <lastmod>{{ $gimme->article->publish_date|camp_date_format:"%Y-%m
    -%d" }}</lastmod>
15.         <changefreq>daily</changefreq>
16.         <priority>0.8</priority>
17.     </url>
18. {{ /list_articles }}
```

```
19. {{ set_language name="English" }}
20. {{ set_topic name="Countries:en" }}
21. {{ list_subtopics }}
22.     <url>
23.         <loc>http://wadr.org/{{ $gimme->language->code }}/?tpid={{ $gimme
    ->topic->identifier }}&amp;tpl=1442/</loc>
24.         <changefreq>daily</changefreq>
25.         <priority>0.4</priority>
26.     </url>
27. {{ /list_subtopics }}
28. {{ set_language name="French" }}
29. {{ set_topic name="Countries:en" }}
30. {{ list_subtopics }}
31.     <url>
32.         <loc>http://wadr.org/{{ $gimme->language->code }}/?tpid={{ $gimme
    ->topic->identifier }}&amp;tpl=1442/</loc>
33.         <changefreq>daily</changefreq>
34.         <priority>0.4</priority>
35.     </url>
36. {{ /list_subtopics }}
37. {{ set_current_issue }}
38. {{ set_language name="English" }}
39. {{ list_sections }}
40.     <url>
41.         <loc>http://wadr.org/{{ $gimme->language->code }}/{{ $gimme->issu
    e->url_name }}/{{ $gimme->section->url_name }}/</loc>
42.         <changefreq>weekly</changefreq>
43.         <priority>0.3</priority>
44.     </url>
45. {{ /list_sections }}
46. {{ set_language name="French" }}
47. {{ list_sections }}
48.     <url>
49.         <loc>http://wadr.org/{{ $gimme->language->code }}/{{ $gimme->issu
    e->url_name }}/{{ $gimme->section->url_name }}/</loc>
50.         <changefreq>weekly</changefreq>
51.         <priority>0.3</priority>
52.     </url>
53. {{ /list_sections }}
54. </urlset>
```

How to get it working

There are different ways to inform the search engines to crawl your site using the XML Sitemap we just created. However, this is out of the scope of this guide as it has nothing to do with Newscoop templating.

You can find detailed information on how to do it in these following links:

http://www.sitemaps.org/

https://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=78808&ctx=cb

KML DOCUMENTS FOR EXPORTING MAP LOCATIONS

KML is like an RSS feed for geo-location data. It is a XML specific notation for expressing geographic annotation and visualization within maps, Earth browsers and other applications.

One of the sample Template Packages on the Sourcefabric website is the "Ushahidi Cooker" which is dedicated to mapping content and generating KML feeds. You will come across KML feeds when you export from mapping services or - in the case of Ushahidi - when you want to import additional information into a map. Newscoop handles maps, and using the following code it can deliver valid KML feeds.

Goal / Task

To generate a simple KML file containing the list of locations for the current article.

Implementation

The structure of a basic KML file breaks down as follows:

- An XML header and a KML namespace declaration. You will see these two lines in the full version of the KML file at the end of this section.
- A Placemark object that contains the following elements: A name used as the label for the Placemark, a description that appears in the "balloon" attached to the Placemark, and a Point that specifies the position of the Placemark on the Earth's surface.

KML is much more complex than this, and allows you to provide more information about the locations in your map and style those as much as you want, but that is KML-specific and out of the scope of this document. What we want to show you here is how you can use $gimme to pull the data and display the locations, which ultimately is what matters the most.

First you need to provide basic info about the article. In this case we use the *article name* and *section name* as the name for our KML document, and the custom field *intro* as the description.

```
1. <kml xmlns="http://www.opengis.net/kml/2.2">
2. <Document>
3.   <name>{{ $gimme->article->name }} in {{ $gimme->section->name }}</name>
4.   <description><![CDATA[{{ $gimme->article->intro }}]]></description>
```

Next, we provide some custom styling for the marker icons. For this we need to generate a unique id per location, so we use the article number plus the incremental index in the list. We also validate inside the list if the location is enabled or not.

```
1. {{ list_article_locations }}
2.     {{ if $gimme->location->enabled }}
3.     <Style id="style{{ $gimme->article->number }}-{{ $gimme->current_list
   ->index }}">
4.         <IconStyle>
5.             <Icon>
6.                 <href>http://www.sourcefabric.org/geolocation/markers/mar
   ker-gold.png</href>
7.             </Icon>
8.         </IconStyle>
9.     </Style>
10.    {{ /if }}
11. {{ /list_article_locations }}
```

Now the most important section, the list of locations to be displayed on the map. We use *list_article_locations*, and after making sure the location is enabled we build the Placemark block. For the Placemark name we are using the location name together with the article name. Then we use *styleUrl* to reference the custom style defined in the block above.

The last sub-block is Point, which is where we actually specify the location positioning on the map. Notice the use of $gimme->location->longitude and $gimme->location->latitude, it could not be more intuitive :-) The value 0.000000 after latitude corresponds to altitude, which is not relevant for this example but must be there as per KML specifications.

```
1. {{ list_article_locations }}
2.     {{ if $gimme->location->enabled }}
3.     <Placemark>
4.         <name>{{ $gimme->location->name }} @ {{ $gimme->article->name }}<
   /name>
5.         <description></description>
```

```
6.          <styleUrl>#style{{ $gimme->article->number }}-{{ $gimme->current_
    list->index }}</styleUrl>
7.          <Point>
8.              <coordinates>{{ $gimme->location->longitude }},{{ $gimme->loc
    ation->latitude }},0.000000</coordinates>
9.          </Point>
10.     </Placemark>
11.     {{ /if }}
12. {{ /list_article_locations }}
```

This is how the full KML file should look:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <kml xmlns="http://www.opengis.net/kml/2.2">
3.  <Document>
4.      <name>{{ $gimme->section->name }} in {{ $gimme->article->name }}</nam
    e>
5.      <description><![CDATA[{{ $gimme->article->description }}]]></descript
    ion>
6.      {{ list_article_locations }}
7.          {{ if $gimme->location->enabled }}
8.          <Style id="style{{ $gimme->article->number }}-{{ $gimme->current_
    list->index }}">
9.              <IconStyle>
10.                 <Icon>
11.                     <href>http://www.sourcefabric.org/geolocation/markers
    /marker-gold.png</href>
12.                 </Icon>
13.             </IconStyle>
14.         </Style>
15.         {{ /if }}
16.     {{ /list_article_locations }}
17.
18.     {{ list_article_locations }}
19.         {{ if $gimme->location->enabled }}
20.         <Placemark>
21.             <name>{{ $gimme->location->name }} @ {{ $gimme->article->name
     }}</name>
22.             <description></description>
23.             <styleUrl>#style{{ $gimme->article->number }}-{{ $gimme->curr
    ent_list->index }}</styleUrl>
24.             <Point>
25.                 <coordinates>{{ $gimme->location->longitude }},{{ $gimme-
    >location->latitude }},0.000000</coordinates>
26.             </Point>
27.         </Placemark>
28.         {{ /if }}
29.     {{ /list_article_locations }}
30. </Document>
31. </kml>
```

Put KML feeds into action

KML files are meant to be used in browsers and any other kind of application supporting this format, like for example Google Earth, Google Maps, or Ushahidi Layers. So, once Newscoop generates this file for you, you just need to use it in any of these tools, depending on your needs.

You can try this with the Template Package "Ushahidi Cooker". You will also find a video tutorial about the "Ushahidi Cooker" here:

http://www.sourcefabric.org/en/products/newscoop_templates/

39. SEARCH ENGINE OPTIMISATION (SEO)

Search Engine Optimisation is about improving the way your content is visible on the Internet. This is often understood as making your publication show up as highly as possible in search results. But limiting SEO to tricking search engines would be missing the point. Think about SEO as part of the service that you provide to your readers, not just a mechanism to jump the queue in search algorithms.

Imagine you have published an article about the impact the fall of the Berlin Wall has had on urban planning in that city today. It is named "Right in the middle" and because your web design uses big, trendy letters this short title just looks really good. Your Newscoop template is using the article name in the title tag in the header of the HTML document.

Imagine a potential reader who is typing "Berlin Wall" into their favourite search engine. Amongst the results, somewhere, your article shows up. The search engine will display the content of the title tag in the long list of results. What are the chances that the reader would click "Right in the middle" when looking for specific information about the Berlin Wall? The reader would probably be more likely to click "Fall of Berlin Wall heats up property speculation 20 years later".

This descriptive content increases the chances that readers will click on your article. At the same time, search engines value the content of the title tag highly. This little bit of extra work is likely to catapult your page upwards in the ranks of search results. Where the old title tag did not even mention the Berlin Wall, your new title tag does, and provides additional key words that will have an impact on your article's ranking and your publication's visibility.

This chapter will help you to make most of your publication's most valuable asset: your content. The following examples will cover a number of small modifications to your templates and other parts of your website which can deliver improved page rank and visibility. The examples will focus on SEO practices involving your publication or template structure, with a few journalistic guidelines.

CREATING DESCRIPTIVE PAGE TITLES

Add the field "seo_title" to your article type. This field can be displayed with $gimme->article->seo_title in the header region of your document.

```
1. <head>
2.     <title>{{ $gimme->article->seo_title }}</title>
3. </head>
```

However, if the journalist forgot to fill in this field, the title tag of the page would be empty. So you should present a fallback option. A simple way of doing this, providing a reasonable solution for section pages and the home page at the same time, would be:

```
1. <head>
2.     <title>{{ strip }}
3.         {{ if $gimme->article->seo_title|trim !== "" }}
4.             {{ $gimme->article->seo_title|escape:'html'|trim }} |
5.         {{ else }}
6.             {{ $gimme->article->name|escape:'html'|trim }} |
7.         {{ /if }}
8.          {{ $gimme->section->name }} in {{ $gimme->publication->name
 }}
```

```
 9.     {{ /strip }}
10. </title>
11. </head>
```

The functions trim and escape:'html' are used to make sure the content is clean and HTML. If the seo_title field is not filled in, the article name is displayed instead. If you are on a section page, the article values are not displayed if you link to the section using option=section.

USE THE "DESCRIPTION" META TAG

The description is a summary of what your article is about. The description meta tag goes into the header of your document. Many times, the text in this description will be given as an introduction to the page in a search result. The meta tag looks like this:

```
<meta name="description=" content="...">
```

Ideally, you should add a field to the Article Type that holds the description content. If this field is empty, you should use text from the main text of the article. A custom description will often be more inviting to a reader, in a list of search results, than the first lines of the main text.

Because the description will most probably come from a WYSIWYG textarea field, it is important to strip_tags. Opinions on the ideal length for meta descriptions vary. In the following example, we set the length to the first 150 characters of the article's main text, if no custom description has been provided.

```
 1. <meta name="description=" content="{{ strip }}
 2. {{ if $gimme->article->description_tag|strip_tags|trim !== "" }}
 3.     {{ $gimme->article->description_tag|strip_tags|escape:'html'|trim }}
 4. {{ else }}
 5.     {{ $gimme->article->full_text|strip_tags|escape:'html'|trim|truncate:
    150 }}
 6. {{ /if }}
 7. {{ /strip }}" />
```

HUMAN READABLE URLS REFLECTING THE CONTENT

Information in the URL describing the content of the page is valued highly by search engines. You can control the URL for each issue and section, setting short names. So instead of the section number "/12/", this part of the URL might read "/culture/". You can find these options in the Newscoop administration interface. Select "Settings" in the list of issues and sections.

The article content can be reflected in three different human readable ways in the URL. You can select the option to use the article title, article keywords, or topics linked with the article. If your publication requires it, you can also create a combination of these options. The configuration for the URL display is done in the administration interface under "Configure Publication".

Here some examples of what these URLs could look like:

- by article title - http://yoursite.com/en/mar2011/posts/4/Healthy-options-for-your-sweet-tooth.htm (the article title is "Healthy options for your sweet tooth")
- by article keywords - http://yoursite.com/en/mar2011/posts/4/healthy-options.htm (the article has keywords "healthy" and "options")

- by article topics - http://yoursite.com/en/mar2011/posts/4/health-dine-wine-tomato-garlic-bread.htm (the article has topics "health", "dine", "wine", "tomato", "garlic" and "bread")
- by combining some - or all - of these options

STRUCTURE HEADING TAGS PROPERLY

Heading tags (h1, h2, h3, ...) reflect the hierarchy of the content on a page. This is how search engines read them, so you should design your page in the same way for humans. For example, when designing a page, don't use heading tags to control the layout.

HTML 5 is not very different from HTML 4 when it comes to SEO, so the rules are almost the same:

- Use only one H1 element on any page
- You can use any number of H2, H3, H4, H5, H6 elements on any page, as long they follow this hierarchy
- Use <ul> or <ol> tags for lists
- For menus, use the <ul> tag in HTML 4 and the <nav> tag in HTML 5
- Use tags for styling blocks inside a template
- Use inline tags like <p> or only for content

The logic for using only one H1 element is derived from the fact that search engines identify <h1> tags as page titles. Sometimes, search engines ignore <title> tags because they have been abused by webmasters.

HTML 5 also introduces new tags like <header>, <footer>, <nav>, <article>, <aside> and <section>. Search engines disqualify the use of multiple <header> tags if they are positioned one after another, but not if they are used as headers for each <article> tag. The same thing happens for <nav> and <footer> tags. A page can have multiple <article>, <aside> and <section> tags, each of these containing just one <header>, <footer> and <nav> tag.

XML SITEMAP FOR YOUR PUBLICATION

Providing a sitemap in a specific XML format will make it easy for search engines to gain access to your content. The XML sitemap delivers all content that you wish to be indexed in a machine readable file.

Providing a sitemap also makes sure that search engines will find all of your content. A simple example: if you are using Flash to link from one page to another, that link is not being followed, because it is invisible to spiders (search engine robots). Such "invisible" pages will be listed in the sitemap, and help search engines to understand where these pages are.

In order to create a sitemap for your publication, see the chapter about XML, RSS, KML and sitemaps.

UNIQUE URLS: THE CANONICAL TAG

Canonical tags have one important purpose: tell search engines what the "clean" URL of the page is. The canonical tag sits in the header of your page. It was introduced in February 2009 by Google, Yahoo and Microsoft and it looks like this:

```
<link rel="canonical" href="http://www.example.com/" />
```

This is meant to put an end to the issues related to duplicate content. In short: duplicate content was used by some sites to increase their page rank. To prevent this kind of spamming, search engines rated domains with duplicate content lower. But any CMS will need different URLs for the same page, for example when passing on a parameter in the URL for browsing history, login, related items and others. The canonical tag now allows sites to make sure they are not ranked lower because they produce some duplicate content. Using the canonical tag will result in higher page ranks.

You need to adjust the following example to the template and folder names you are using for your publication.

```
1. {{ if $gimme->template->name == "package_name/article.tpl" }}
2.     <link rel="canonical" href="{{ url options="article" }}" />
3. {{ /if }}
4. {{ if $gimme->template->name == "package_name/section.tpl" }}
5.     <link rel="canonical" href="{{ url options="section" }}" />
6. {{ /if }}
7. {{ if $gimme->template->name == "package_name/index.tpl" }}
8.     <link rel="canonical" href="http://{{ $gimme->publication->site }}" /
   >
9. {{ /if }}
```

ADDITIONAL CHECKLIST FOR YOUR JOURNALISTS AND EDITORS

The following list is not relevant for making templates. But while you are working on SEO, you might as well pass on some tips to your colleagues who are contributing content. At the end of the day, their input will guide the audience to your site.

Explain to the journalists and editors that their input into SEO could dramatically increase the readership of their articles on your publication's site. A little extra effort increases advertiser value, extends the shelf life of the article, and makes the journalist who wrote it much more famous.

When writing descriptions (for both articles and images)...

- Summarize the content accurately. Below are a few pointers that might help journalists and editors create a good description.
- Write unique descriptions. While this might appear impossible in large publications with thousands of articles and a large contributing staff, keeping this idea in your head will help avoid being boring and generic.
- Write for your audience, not for a search engine. Avoid writing a description tag that bears no resemblance to the content of the article. Don't write up a list of keywords, but form a sentence or two.
- Do not use the same description across your site. This could actually be worse than using no description tag at all. Imagine all search results in a list saying: "Simply the best magazine in the world".

When writing an article or image description, or providing a custom SEO title, include the following elements:

- Who or what is being shown? What is the name of the person, the animal, the building, the event? "Bird" is better than nothing, but "Eagle" would be better
- Where is it happening? "Eagle in the sky" is good, "Eagle in the sky above the Alps" is better
- Why is it happening? Why are you writing about it? Are you writing about an "Endangered Eagle in the sky above the Alps"

- When is this happening? Are we looking at an "Endangered Eagle in the sky above the Alps in Spring"?

You don't need to go overboard with building endless descriptions. Prioritize according to your story. The above questions are a good way to get to the essence quickly. In the end, you might settle for "Endangered Eagle flys in the Alps" - this captures the story better and would attract far more readers than "Bird" would ever do.

Make use of the image alt description

When adding an image, do not leave the alt tag empty. Firstly, screen reader programs for people with visual impairments rely on this information. Secondly, image searches on the Internet will categorise and rank images based on information in the alt tag. Readers finding your publication through image searches may be more common than you might think.

Link text should relate to the page it links to

When linking to a page, make sure the link text is related to the content of the page you link to. "You can download Newscoop **here**" is bad. "**Download Newscoop** for free" is better.

40. NEWSCOOP AND HTML5

The big challenge for web publishing today is the migration of sites to the HTML5 and CSS3 standards. Fortunately, Newscoop is HTML5 ready.

Other content management systems rely on HTML code provided by third-party widgets, modules or plugins. Changing the HTML output of these external modules often requires digging into their code. In Newscoop, because of the strict separation of business logic and presentation logic in the template engine, you have total control over HTML5 and CSS3 template output.

Below, we introduce a few examples of HTML5 / CSS3 in Newscoop templates to illustrate the ease of implementation of HTML5 features.

HTML metatags and header information

A typical HTML5 header in Newscoop would start like this:

```
1.  <!DOCTYPE html>
2.   <html lang="{{ $gimme->language->code }}">
3.    <head>
4.     <meta charset="utf-8" />
5.      <meta name="description" content="{{ $siteinfo.description }}" />
6.       <meta name="keywords" content="{{ if $gimme->article->keywords }}
7.                                      {{ $gimme->article->keywords }},
8.                                     {{ /if }}
9.                                    {{ $siteinfo.keywords }}" />
10.      <title>{{ $gimme->publication->name }}</title>
11.     <link rel="stylesheet" href="screen.css" media="all" />
12.   </head>
```

{{ $siteinfo.description }} is the global variable which holds the site description specified in the Newscoop administration interface.

{{ $gimme->article->keywords }} and {{ $siteinfo.keywords }} are two ways to include meta tag keywords. In Newscoop, an individual article can have its own keywords specified. If that's the case, the meta tag for keywords will be filled with these article specific keywords. Otherwise, only global site keywords will be used.

Finally, the site title is defined showing the name of the publication using {{$gimme->publication->name}}.

Multiple columns in HTML5

Multi-Column Layout defines new CSS properties for columns of text. Content can flow into multiple columns, with a gap between them. You can put the content in the appropriate places inside the HTML layout.

```
1.  <header>
2.     <hgroup>
3.            <h1>{{ $gimme->article->name }}</h1>
4.     </hgroup>
5.     <p>By: {{ $gimme->article->author->name }}</p>
6.  </header>
7.  <div class="text1">
8.     <p>{{ $gimme->article->lead }}</p>
9.  </div>
```

```
10. <div class="text2" id="customCols">
11.     <p>{{ $gimme->article->body }}</p>
12. </div>
```

Accompanied by some CSS3 styling, this piece of code is enough to divide text into columns, gaps, and even rules between columns.

```
 1. .text2 {
 2.   -webkit-column-count: 2;
 3.   -moz-column-count: 2;
 4.   column-count: 2;
 5.   -webkit-column-gap: 20px;
 6.   -moz-column-gap: 20px;
 7.   column-gap: 20px;
 8.   -webkit-column-rule: 1px solid #999;
 9.   -moz-column-rule: 1px solid #999;
10.   column-rule: 1px solid #999;
11.   -webkit-border-radius: 10px;
12.   -moz-border-radius: 10px;
13.   border-radius: 10px;
14. }
```

Video and audio handling

Not only text content is HTML5 ready - Newscoop can handle media attachments in a HTML5 way. The following code shows how an audio file which has been attached to an article in the Newscoop administration interface can be handled in HTML5:

```
1. <audio controls>
2.     <source src="{{ uri options="articleattachment" }}"
3.             type="{{ $gimme->attachment->mime_type }}">
4. </audio>
```

The code above displays a player for the specified audio attachment. To help the browser to determine the MIME type of the audio file, the 'type' attribute is specified, although it can be omitted. The same technique is possible for video content:

```
1. <video controls width="520" height="330">
2.     <source src="{{ uri options="articleattachment" }}"
3.             type="{{ $gimme->attachment->mime_type }}">
4. </video>
```

In both cases, {{ uri options="articleattachment" }} is replaced with the audio or video file URL. With HTML5, you don't need to specify Flash players, because modern browsers have a built-in player for embedded multimedia content.

41. TROUBLESHOOTING TEMPLATES

Newscoop's administration interface has a built-in preview option on the article edit page which also shows template parsing errors, if there are any. When you browse your publication in the preview window, you can see error messages for all parts of your publication.

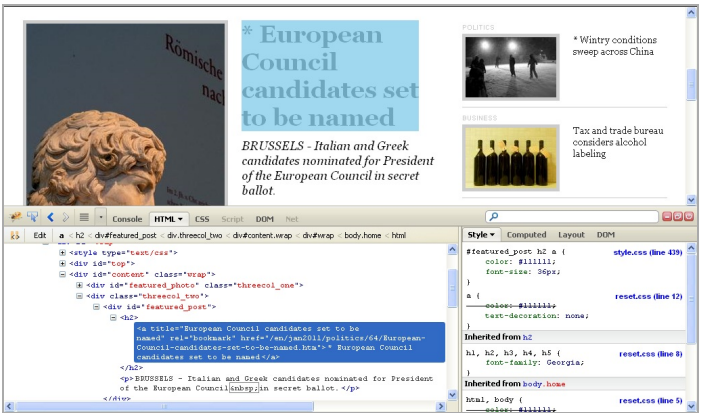
At the bottom of the preview window you can see error messages from the template parser.

The second frame on this page displays any parsing errors, which can be very useful for debugging templates. It immediately narrows down possible errors to the specific main or included template. It also displays the line and type of error.

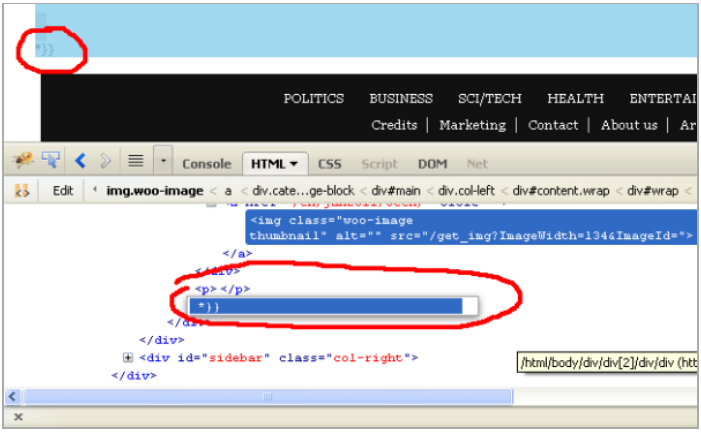
The error messages are not always precise. The point where the parser encounters an error may only be a reflection of a problem coming from further up in the template. If the message shown by the parse error frame is not useful to find the problem, you'll have to find an alternate way to locate it.

For example, in some situations, checking the source of the generated HTML page helps. It may be that the content you expect to be shown somewhere doesn't appear on the page, but it's in the HTML source. This obviously shows that the problem is not in Newscoop but somewhere in HTML or CSS. The opposite is also true; if the source contains all the and <li> tags that it should but their content is empty, you can be sure that Newscoop isn't working like it should.

There are lots of very handy tools out there in browsers nowadays. You can use them to easily parse your HTML code and find an error. The one used in Firefox is Firebug. It is an extension to Firefox, all you need to do is to find it in the repository and install to the browser:

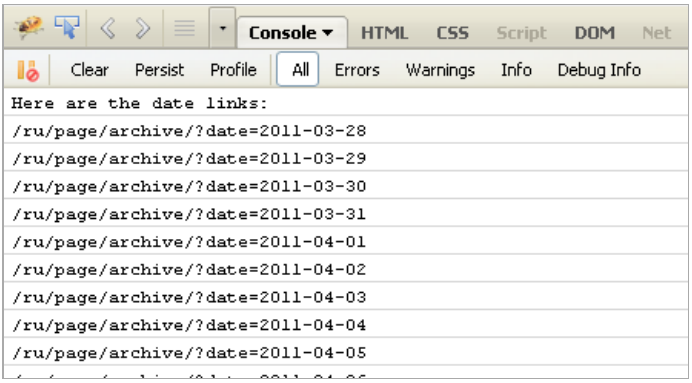


The usage is pretty easy - open, inspect element, find error:

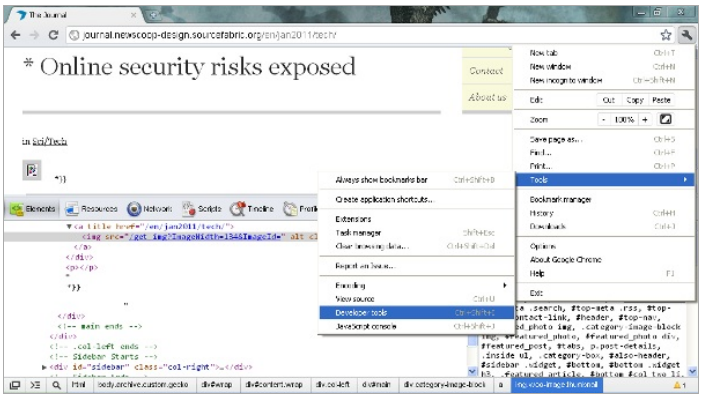


But what is really good, is that you can debug your jQuery templates with it. Just adding a few lines of code to your JavaScript will provide Console feedback:

```
1. console.log ("Here are the date links:\r\n");
2. ...
3. console.log (link + "\r\n");
```



There's almost the same tool in Chrome, which can be found in the menu under Tools -> Developer tools:



Another debugging strategy may be inserting temporary tags for printing the values of Newscoop variables.

The following code checks the values of variables in the registration process. It doesn't make any sense to use on a live site, but it may be useful for debugging purposes:

```
<h2>Add new subscription</h2>
 {{* debug user add/edit/subscribe *}}
  <h5>Edit_user_action:</h5>
   <p>Edit user action defined: {{ if $gimme->edit_user_action->defined }}defined{{ else }}not defined{{/if}}</p>
   <p>Edit user action error: {{ if $gimme->edit_user_action->is_error }}is_error, code: {{ $gimme->edit_user_action->error_code }}, message: {{ $gimme->edit_user_action->error_message }}{{ else }}not error{{/if}}</p>
   <p>Edit user action ok: {{ if $gimme->edit_user_action->ok }}ok{{ else }}not ok{{/if}}</p>
   <p>edit user action type: {{ $gimme->edit_user_action->type }}</p>

  <h5>Edit_subscription_action:</h5>
   <p>Subs action defined: {{ if $gimme->edit_subscription_action->defined }}defined{{ else }}not defined{{/if}}</p>
   <p>Subs action error: {{ if $gimme->edit_subscription_action->is_error }}is_error, code: {{ $gimme->edit_subscription_action->error_code }}, message: {{ $gimme->edit_subscription_action->error_message }}{{ else }}not error{{/if}}</p>
   <p>Subs action ok: {{ if $gimme->edit_subscription_action->ok }}ok{{ else }}not ok{{/if}}</p>
   <p>Subs action {{ if $gimme->edit_subscription_action->is_trial }}is_trial{{ else }}not trial{{/if}}</p>
   <p>subs action {{ if $gimme->edit_subscription_action->is_paid }}is_paid{{ else }}not paid{{/if}}</p>
 {{* end debugging *}}
```

This would be the output:

Add new subscription

Edit_user_action:

Edit user action defined: defined
Edit user action error: not error
Edit user action ok: ok
edit user action type: edit

Edit_subscription_action:

Subs action defined: not defined
Subs action error: not error
Subs action ok: not ok
Subs action not trial
subs action not paid

Similarly to this, you can use this temporary outputs wherever you feel you need to check what's really happening. For example, inside a list to check how values of list index and list count change with every new cycle -

```
1. {{ list_articles }}
2.     <div class="post wrap">
3.         <p>Current list index: {{ $gimme->current_list->index }}, list co
   unt: {{ $gimme->current_list->count }}</p>
4.         <h2 class="post-title"><a href="{{ uri options="article" }}" rel=
   "bookmark" title="{{ $gimme->article->name }}">{{ if ! $gimme->article->c
   ontent_accessible }}* {{ /if }}{{ $gimme->article->name }}</a></h2>
5.     </div>
6. {{ /list_articles }}
```

Here we put a temporary <p> tag with values of index and count above every title, and the result would be like this screenshot:

SECTION: POLITICS (JANUARY 2011)

Current list index: 1, list count: 3

* European Council candidates set to be named

Current list index: 2, list count: 3

* Wintry conditions sweep across China

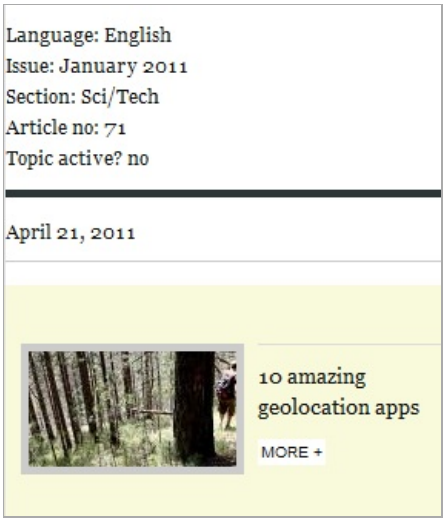
Current list index: 3, list count: 3

Food export duties set to rise

Or, you may want to check the values of parameters forwarded to a page. You can do this at the template's very beginning:

```
1. <p>Language: {{ $gimme->language->name }}<br />
2.  Issue: {{ $gimme->issue->name }} <br />
3.  Section: {{ $gimme->section->name }}<br />
4.  Article no: {{ $gimme->article->number }}<br />
5.  Topic active? {{ if $gimme->topic->defined }}yes{{ else }}no{{ /if }}
6. </p>
7.
8. {{ include file="set_thejournal/_tpl/top.tpl" }}
```

The output displays the values of the parameters, so you can check if everything is OK:



MOST COMMON ERRORS / PROBLEMS

- Missing a closing element for {{ if }} statements - this especially happens when several 'if' clauses are nested. You can simplify your expression by using {{ elseif }}; that will reduce the need for closing 'if's
- Missing list closing element - this also happens more often when using lists inside lists
- Wrong syntax inside an 'article list' constraints option. Because there are a lot of possible options, always consult the template reference at the end of this book for options and their correct formatting
- Syntax errors in general - again, the safest way to avoid errors is to always consult the template reference
- Context problems - these are harder to discover because they don't produce parse errors in the preview window. To discover context-related values problems, try inserting temporary tags to output the values of variables. Do it at the beginning of the template, then do it again after an action was performed. This way, you can see what really happens with those variables.

WHERE TO GO FOR MORE HELP

There are a few places you can turn for troubleshooting assistance on your templates. The first place to look is the Newscoop support forum, which you can find at http://forum.sourcefabric.org

Sourcefabric also offers paid support. As the maintainer of Newscoop (and its largest code contributor), Sourcefabric can provide a broad and deep body of knowledge to our customers. For example, we can help troubleshoot your templates or installation, or we can create custom templates for you. You can find more details about Sourcefabric's services on the website: http://services.sourcefabric.org

TEMPLATES USING JQUERY, PHP AND THIRD PARTY SERVICES

42. USING JQUERY AND PHP IN TEMPLATES
43. PULLDOWN MENU WITH SUPERFISH USING JQUERY
44. TABS WITH JQUERY
45. TICKER WITH JQUERY
46. IMAGE GALLERIES WITH JQUERY
47. CALENDARS WITH JQUERY
48. USING POLLS WITH AJAX
49. WIDGETS: YOUTUBE, VIMEO, TWITTER, FLICKR & CO.
50. PROFILE PICTURES: GRAVATAR, FACEBOOK, TWITTER
51. COMBINING NEWSCOOP AND SOUNDCLOUD
52. WORKING WITH FACEBOOK
53. WORKING WITH ADVERTISING
54. WAYS TO CREATE ARCHIVES

42. USING JQUERY AND PHP IN TEMPLATES

The Newscoop template engine comes with a set of custom functions which are used in examples in this Cookbook. A full template reference can be found at the end of the book. Everything involving the $gimme object is Newscoop related.

You can also find a number of examples in the advanced section, which use jQuery and PHP. This chapter explains some of the background information you need to know to make your work with advanced functions easier to understand. It will also allow you to venture into making your own. If you do, by all means, tell us about it! You can mail us at contact@sourcefabric.org and share your ideas with the community in our forums at http://forums.sourcefabric.org

USING JQUERY IN YOUR TEMPLATES

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. If you haven't used jQuery yet, you could start getting familiar with it by reading the tutorial on the jQuery site: http://docs.jquery.com/Tutorials:How_jQuery_Works

To use jQuery with Newscoop, you just need to add libraries in your _html-header.tpl file, and that's it! It will be much easier if you follow our recommendations and put all your own jQuery code in a separate file (/templates/set_name/_js/functions.js).

So the first thing to do will be to include all required jQuery libraries in the html header:

```
1.  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2.  <html xmlns="http://www.w3.org/1999/xhtml">
3.  <head>
4.    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5.    <meta http-equiv="Content-Language" content="en" />
6.    ...
7.    <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.2/jquery.min.js"></script>
8.    <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.11/jquery-ui.min.js"></script>
9.    <link type="text/css" href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.11/themes/base/jquery-ui.css" rel="stylesheet" />
10.   <script src="http://{{ $gimme->publication->site }}/templates/set_name/_js/functions.js"></script>
11.   ...
12. </head>
```

The first <script> line is the jQuery library itself. It is required for all functions to work.

The next two lines are for jQuery UI, which provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery JavaScript Library.

As you can see in the above example, we included all libraries from Google's CDN (see http://code.google.com/apis/libraries/devguide.html#jqueryUI). You also can download jQuery (see http://docs.jquery.com/Downloading_jQuery) and copy it to the right place on your server:

```
1.   <!-- Grab Google CDNs jQuery. fall back to local if necessary -->
```

```
2.   <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.2/jquery.m
     in.js"></script>
3.   <script>!window.jQuery && document.write(unescape('%3Cscript src="http:
     //{{ $gimme->publication->site }}/templates/set_thejournal/_js/jquery-1.5
     .2.min.js"%3E%3C/script%3E'))</script>
4.   ...
5.   <script src="http://{{ $gimme->publication->site }}/templates/set_name/
     _js/functions.js"></script>
```

Your part of the magic is created inside the functions.js file:

```
1. $(document).ready(function(){
2. ...
3. });
```

If you need to place a piece of jQuery inside some particular template, just use the <script> tag:

```
1. <script type="text/javascript">$(document).ready(function(){
2. ...
3. });</script>
```

jQuery, like other JavaScript frameworks, is a very handy tool to make your designs work, so that you can concentrate on other things. Here is, just for example, a small piece of code to make an "Increase/Decrease Font Size" control. First, the JavaScript, then the template code:

functions.js

```
 1. var articleBox = $(".block-article");
 2. var originalFontSize = $(articleBox).css('font-size');
 3.   // Increase Font Size
 4. $("a[href=#zoomin]").click(function(){
 5.   var currentFontSize = $(articleBox).css('font-size');
 6.   var currentFontSizeNum = parseFloat(currentFontSize, 10);
 7.   var newFontSize = currentFontSizeNum*1.1;
 8.   $(articleBox).css('font-size', newFontSize);
 9.   return false;
10. });
11.   // Decrease Font Size
12. $("a[href=#zoomout]").click(function(){
13.   var currentFontSize = $(articleBox).css('font-size');
14.   var currentFontSizeNum = parseFloat(currentFontSize, 10);
15.   var newFontSize = currentFontSizeNum*0.9;
16.   $(articleBox).css('font-size', newFontSize);
17.   return false;
18. });
```

article-cont.tpl

```
1. <div class="block-fonts-control">
2.  <a href="#zoomin">Increase</a> / <a href="#zoomout">Decrease</a> font
3. </div>
4. ...
5. <div class="block-article">
6.   {{$gimme->article->intro}}
7.   {{$gimme->article->full_text}}
8. </div>
```

USING PHP IN YOUR TEMPLATES

The {{ php }} tag allows PHP code to be embedded directly into the template. This is for advanced users only, not normally needed and not recommended. The following information was taken from http://www.smarty.net

Example PHP code within {{ php }} tags:

```
1. {{ php }}
2.   // including a php script directly from the template.
3.   include('/path/to/display_weather.php');
4. {{ /php }}
```

PASSING ON VARIABLES BETWEEN $GIMME AND PHP

When using PHP, you might want to work with $gimme values inside PHP, as well as pass on PHP variables to the template. In order to do this, you need to assign the variables first, then you can access them inside PHP. Important: use backticks ` in the assign function. Here's a little example:

Assigning a variable in the template to use with PHP

```
{{ assign var="profile_email" value=`$gimme->comment->reader_email` }}
{{ php }}
$profile_email = $this->get_template_vars('profile_email');
print $profile_email;
{{ /php }}
```

Note: there seems to be no way to pass on variables from an included file into the parent. We could not figure it out. If you can, please add your code here!

Assign a variable in PHP to use in the template

```
1. {{* a {{php}} block that assigns the variable $varX *}}
2. {{php}}
3.   $this->assign('varX','Toffee');
4. {{/php}}
5. {{* output the variable in the template *}}
6. <strong>{{$varX}}</strong> is my favourite ice cream :-)
```

43. PULLDOWN MENU WITH SUPERFISH USING JQUERY

Download and documentation: http://users.tpg.com.au/j_birch/plugins/superfish/

Pulldown menus with sub-menus opening on mouse-over are very popular. We recommend using Superfish for elegance, flexibility and SEO compatible design. We show you how to add this to your Newscoop publication in five simple steps.

Superfish is a jQuery menu plugin developed by Joel Birch which adds neat usability to an existing CSS drop-down menu (so it degrades gracefully without JavaScript). You can see it in action at www.sourcefabric.org. Go to the Superfish website for more details, examples and documentation. Download the zip-file before we start.

In our example, we use the Template Package "Ushahidi Cooker". The Superfish navigation will be added beneath the header and above the breadcrumbs. You can download the "Ushahidi Cooker" at www.sourcefabric.org under Products > Newscoop > Templates. Without the pulldown menu, the design looks like this:

The "Ushahidi Cooker" is a directory of maps. We will now improve usability with a pulldown menu.

After you downloaded the Superfish zip-file, fire up your text editor and open the example.html file.

1. Adding JavaScript and CSS links to header

When you open the file example.html in the Superfish zip, you will see these lines in the header of the document, linking in the CSS files, the JavaScript files and initialise the jQuery plugin:

```
1. <link rel="stylesheet" type="text/css" href="css/superfish.css" media="sc
   reen">
2. <script type="text/javascript" src="js/jquery-1.2.6.min.js"></script>
3. <script type="text/javascript" src="js/hoverIntent.js"></script>
4. <script type="text/javascript" src="js/superfish.js"></script>
5. <script type="text/javascript">
6. // initialise plugins
7.   jQuery(function(){
8.     jQuery('ul.sf-menu').superfish();
```

```
  9.   });
 10. </script>
```

Following the recommended folder structure described in an earlier chaper, I change these to:

```
  1. <link rel="stylesheet" type="text/css" href="/templates/set_ushahidicooke
     r/_css/superfish/css/superfish.css" media="screen">
  2. <script type="text/javascript" src="/templates/set_ushahidicooker/_js/jqu
     ery-1.2.6.min.js"></script>
  3. <script type="text/javascript" src="/templates/set_ushahidicooker/_js/hov
     erIntent.js"></script>
  4. <script type="text/javascript" src="/templates/set_ushahidicooker/_js/sup
     erfish.js"></script>
```

As you can see in this example, Superfish comes with a jQuery file (jquery-1.2.6.min.js). Note: there is a new folder now inside _css by the name of superfish with a subfolder css. Superfish has a css and an images folder, they both go into the new superfish folder. You'll make your life easier if you keep files in separate folders.

These changes need to be added in the sub-template for the header. For the "Ushahidi Cooker" this is:

```
set_ushahidicooker/_tpl/header.tpl
```

Because this file already calls jQuery (see below), I leave this line out. I add the lines just beneath the linked jquery.min.js and add the initialising JavaScript before the header closes.

```
  1. <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/
     jquery/1/jquery.min.js"></script>
  2. <!-- Superfish files -->
  3. <link rel="stylesheet" type="text/css" href="/templates/set_ushahidicooke
     r/_css/superfish/css/superfish.css" media="screen">
  4. <script type="text/javascript" src="/templates/set_ushahidicooker/_js/hov
     erIntent.js"></script>
  5. <script type="text/javascript" src="/templates/set_ushahidicooker/_js/sup
     erfish.js"></script>
  6.     <script type="text/javascript">
  7.     // initialise plugins
  8.     jQuery(function(){
  9.       jQuery('ul.sf-menu').superfish();
 10.     });
 11.     </script>
 12. </head>
```

2. Copying files to correct folders

Create the folder superfish inside _css and move the css and images folder into is. These are the files and folders you should now have:

```
  1. set_ushahidicooker/_css/superfish/css/
  2.   - superfish.css
  3.   - superfish-navbar.css
  4.   - superfish-vertical.css
  5. set_ushahidicooker/_css/superfish/images/
  6.   - arrows-ffffff.png
  7.   - shadow.png
```

Also move the JavaScript files into the _js folder.

3. Create the sub-template to build the navigation

The file you create is called sf-menu.tpl and is saved inside set_ushahidicooker/_tpl/

4. Generate the menu HTML with Newscoop

In our case, we are listing the sections as the visible menu items running horizontally, and the articles in each section are in the pulldown item. The code inside sf-menu.tpl looks like this:

```
1.     <ul class="sf-menu">
2. {{ list_sections }}
3.       <li>
4.         <a href="{{ uri }}">{{ $gimme->section->name }}</a>
5. {{ list_articles }}
6. {{ if $gimme->current_list->at_beginning }}
7.         <ul>
8. {{ /if }}
9.           <li>
10.            <a href="{{ uri }}">{{ $gimme->article->name }}</a>
11.          </li>
12. {{ if $gimme->current_list->at_end }}
13.        </ul>
14. {{ /if }}
15. {{ /list_articles }}
16.      </li>
17. {{ /list_sections }}
18.    </ul>
```

Again, the same list. Now we focus on the Newscoop action that is taking place:

List all sections and open a list item:

```
1.  {{ list_sections }}
2.       <li>
```

Create a link to the section, displaying the section name:

```
        <a href="{{ uri }}">{{ $gimme->section->name }}</a>
```

List all articles. These will be the articles inside the current section from the list_section command. In other words: only the ones in the section we just printed:

```
{{ list_articles }}
```

Now we open another unordered list (ul). But only when we start with the listing of articles:

```
1. {{ if $gimme->current_list->at_beginning }}
2.         <ul>
3. {{ /if }}
```

Now the articles are listed as list items:

```
1.           <li>
2.             <a href="{{ uri }}">{{ $gimme->article->name }}</a>
3.           </li>
```

Then we close the unordered list, once we reached the last item and finish list_articles:

```
1. {{ if $gimme->current_list->at_end }}
2.         </ul>
```

```
{{ /if }}
{{ /list_articles }}
```

Now we have left the sub-loop of articles, we are back again with the section. We close the list item and then the section:

```
        </li>
{{ /list_sections }}
```

Note: if a section is empty, the sub-list will not be built by Newscoop.

5. Include sf-menu.tpl in the page

You can call this page now anywhere you want with:

```
{{ include file="set_ushahidicooker/_tpl/sf-menu.tpl" }}
```

The screenshot show the Superfish pulldown at the end of the template _tpl/title.tpl. Open the file title.tpl and add this line at the end. Then it should end like this:

```
...
<h1>{{ $gimme->publication->name }}</h1>
<span>Location and Map Directory with Geographic Data Feed (KML)</span>
</div>
{{ include file="set_ushahidicooker/_tpl/sf-menu.tpl" }}
```

Done. Now you could adjust the CSS files if you like. If you don't, the Superfish will look like this:

The "Ushahidi Cooker" now has a pulldown menu, showing the city maps for each section.

44. TABS WITH JQUERY

jQuery tabs are an efficient way to show content like *Most Recent Articles* or *Most Recent Comments* in one place. You can read a lot about the jQuery UI Tabs widget at their site, http://jqueryui.com/demos/tabs/

Here is an example jQuery tab (see the chapters on *Most Recent Articles* or *Most Recent Comments* to see how those are generated):

Here's the jQuery part:

```
1.  $(function(){
2.    $('#container-1 ul').tabs();
3.  });
```

The rest is done inside Newscoop templates.
Here's the logic:

```
1.  <div id="container-1">
2.
3.  <ul>
4.    <li><a href="#fragment-1">Title 1</a></li> {{* first tab title *}}
5.    <li><a href="#fragment-2">Title 2</a></li> {{* second tab title *}}
6.  </ul>
7.
8.  <div id="fragment-1">
9.    {{* first tab content *}}
10. </div>
11.
12. <div id="fragment-2">
13.   {{* second tab content *}}
14. </div>
15.
16. </div>
```

And here's the actual example:

```
1.  <div id="container-1">
2.
3.  <ul>
4.    <li><a href="#fragment-1"><span> </span></a></li> {{* first tab title *
    }}
5.    <li><a href="#fragment-2"><span> </span></a></li>  {{* second tab title
     *}}
6.  </ul>
7.
8.  <div id="fragment-1">  {{* first tab content *}}
9.  {{set_section number="1"}}
10. {{list_articles length="12" constraints="onfrontpage is off onsection is
    on" ignore_issue="true"}}
11.   <h4><a href="{{uri options="article"}}">{{ $campsite->article->name}}</
    a>{{if $campsite->article->has_video}}<i class="ico video"></i>{{/if}}{{i
    f $campsite->article->has_photo}}<i class="ico photo"></i>{{/if}}{{if $ca
    mpsite->article->has_audio}}<i class="ico audio"></i>{{/if}}</h4>
12.   <p class="container-date">{{$campsite->article->publish_date|camp_date_
    format:"%d %M %Y."}}</p>
13.   <p>{{$campsite->article->Intro_front}}{{if $campsite->article->comment_
    count>0}}<a href="{{uri options="article"}}#comments" class="comments_cou
    nter">{{$campsite->article->comment_count}}</a>{{/if}}</p>
14. {{/list_articles}}
15. {{set_default_section}}
16. </div>
17.
18. <div id="fragment-2">  {{* second tab content *}}
19. {{local}}
20. {{unset_issue}}
21. {{list_issues length="2" order="bynumber desc"}}
22. {{list_sections constraints="number greater 9 number smaller 70"}}
23.   {{list_articles length="4" constraints="topic is \ :ru" order="bynumber
     desc"}}
24.   <h4><a href="{{uri options="article"}}">{{$campsite->article->name}}</a
    >{{if $campsite->article->has_video}}<i class="ico video"></i>{{/if}}{{if
     $campsite->article->has_photo}}<i class="ico photo"></i>{{/if}}{{if $cam
    psite->article->has_audio}}<i class="ico audio"></i>{{/if}}</h4>
25.   <p class="container-date">{{$campsite->article->publish_date|camp_date_
    format:"%d %M %Y."}}</p>
26.   <p>{{$campsite->article->intro}}{{if $campsite->article->comment_count>
    0}}<a href="{{uri options="article"}}#comments" class="comments_counter">
    {{$campsite->article->comment_count}}</a>{{/if}}</p>
27.   {{/list_articles}}
28. {{/list_sections}}
29. {{/list_issues}}
30. {{/local}}
31. </div>
32.
33. </div>
```

45. TICKER WITH JQUERY

Do you want a BBC-style ticker like the one on http://wadr.org/ driven by Newscoop? Of course you do!

Everything important is in this code:

```
1. <div id="ticker-wrapper" class="no-js">
2.  <ul id="js-news" class="js-hidden">
3.   {{ list_articles length="5" order="bypublishdate desc" constraints="section is 5" }}
4.    <li class="news-item">
5.     <a href="{{ uri options="article" }}" >{{ $gimme->article->name }}</a>
6.    </li>
7.   {{ /list_articles }}
8.  </ul>
9. </div>
```

This code selects the last five published articles from section number 5 and creates an unordered list of article names, also linking them to the full article page.

Add some jQuery magic:

```
1. $(document).ready(function(){
2.   $('#js-news').ticker({
3.     titleText: 'WADR Highlights'
4.   });
5. });
```

The engine behind this is the jQuery News Ticker (http://www.jquerynewsticker.com/), so be sure to have all necessary includes in their place, in _html-head.tpl

```
1. <!-- Grab Google CDN's jQuery. fall back to local if necessary -->
2. <script src="/ajax.googleapis.com/ajax/libs/jquery/1.5.1/jquery.js"></script>
3. <script>!window.jQuery && document.write(unescape('%3Cscript src="http://{{ $gimme->publication->site }}/templates/js/jquery-1.5.1.min.js"%3E%3C/script%3E'))</script>
4. <script type="text/javascript" src="http://{{ $gimme->publication->site }}/templates/js/jquery-ui-1.8.10.custom.min.js"></script>
5. <script type="text/javascript" src="http://{{ $gimme->publication->site }}/templates/js/jquery.ticker.js"></script>
```

and also the css file that belongs to it:

```
 <link type="text/css" rel="stylesheet" href="http://{{ $gimme->publication->site }}/templates/css/highlights.css" />
```

This is of course just one of hundreds of different solutions that you can find on the Internet, so feel free (and encouraged) to explore and play with other toys!

46. IMAGE GALLERIES WITH JQUERY

This example is using a specific gallery solution working with jQuery. Once you understand the logic behind the code, you can use any gallery you want. Just download the package and add Newscoop template code where needed. This chapter will show you how it's done.

Our gallery example will display a number of images attached to an article in your publication. Again, you can modify the logic easily and, for example, call in one image from a list of articles instead.

Start with attaching a number of *Images* to an *Article*, which you want to show to the world. In this case we think of the *Article* as a container for your images. All you need to do is *list_article_images* and decide, how you want them to show up (Lightbox, Slimbox, whatever) - and that's it!

An important thing before you start is to decide what size (in pixels) your images and thumbnails will have. You will need these numbers set to make all images and thumbnails appear unified.

It's up to you to decide which tool you will use to make your images shine. In this example we will use Fancybox *(http://fancybox.net/)* and the jQuery plugin. But you are not limited in any way, and can choose any gallery you like - Newscoop will just provide images for your favourite plugin.

SIMPLE ARTICLE IMAGE GALLERY

This type of image gallery could be used to show additional images, which are part of an *Article*, but do not fit in the main text.

Here's a real life example *(http://ganc-chas.by/by/page/history/323)* of how it's done (you can copy & paste it to your page, and edit it later):

```
1.  {{list_article_images columns="4"}}
2.  {{if $gimme->current_list->count > 1}}
3.    {{if $gimme->current_list->at_beginning}}
4.      <div class="image-gallery-title">Article Images</div>
5.      <div class="image-gallery-container">
6.    {{/if}}
7.    {{if $gimme->image->article_index > 2}}
8.      <a href="{{uri options="image"}}" rel="gallery" class="gallery_thumbn
    ail" title="{{$gimme->image->description|escape}}">
9.        <img src="{{uri options="image"}}&ImageRatio=20" alt="{{$gimme->ima
    ge->description|escape}}" width="100" height="100" />
10.     </a>
11.   {{/if}}
12.   {{if $gimme->current_list->at_end}}
13.     </div>
14.   {{/if}}
15. {{/if}}
16. {{/list_article_images}}
17.
18. <script type="text/javascript">
19. $(document).ready(function(){
20.
21. $("a.gallery_thumbnail").fancybox({
22.   type: 'image',
23.   titlePosition: 'inside',
24.   transitionIn: 'none',
```

```
25.   transitionOut: 'none',
26.   centerOnScroll: 'true'
27. });
28. $("a.gallery_thumbnail").live("mouseenter",function(){$(this).animate({op
    acity:1},200);});
29. $("a.gallery_thumbnail").live("mouseleave",function(){$(this).animate({op
    acity:0.8},200);});
30.
31. });
32. </script>
```

Let's now take a look at parts of code:

```
1. {{list_article_images columns="4"}}
2. {{if $gimme->current_list->count > 1}}
3.   {{if $gimme->current_list->at_beginning}}
4.     <div class="image-gallery-title">Article Images</div>
5.     <div class="image-gallery-container">
6.   {{/if}}
```

We want to have article images placed in 4 columns. First, we check if the *Article* has images attached. Then we provide a header for the gallery and container element.

```
1. {{if $gimme->image->article_index > 2}}
2.   <a href="{{uri options="image"}}" rel="gallery" class="gallery_thumbnai
     l" title="{{$gimme->image->description|escape}}">
3.     <img src="{{uri options="image"}}&ImageRatio=20" alt="{{$gimme->image
     ->description|escape}}" width="100" height="100" />
4.   </a>
5. {{/if}}
```

In this example we will list only images with numbers 3 and higher - this is done to separate gallery images from the main article images. You can decide to have images for the gallery numbered 100 and above - it's up to you.

We need to provide Fancybox with a link to an original image (full size), and also a thumbnail image (**&ImageRatio**) to show to the user. We also supply **class** & **rel** names for Fancybox to define a group of image objects that should be used for the gallery. Other *Image* properties could also be included, like *description* in the above example. Finally, we close the list:

```
1.   {{if $gimme->current_list->at_end}}
2.     </div>
3.   {{/if}}
4. {{/if}}
5. {{/list_article_images}}
```

That's pretty much it for Newscoop. Now let's go down to the jQuery part:

```
1. <script type="text/javascript">
2. $(document).ready(function(){
3.
4. $("a.gallery_thumbnail").fancybox({
5.   type: 'image',
6.   titlePosition: 'inside',
7.   transitionIn: 'none',
8.   transitionOut: 'none',
9.   centerOnScroll: 'true'
10. });
11. $("a.gallery_thumbnail").live("mouseenter",function(){$(this).animate({op
    acity:1},200);});
12. $("a.gallery_thumbnail").live("mouseleave",function(){$(this).animate({op
    acity:0.8},200);});
13.
```

```
14. });
15. </script>
```

This example assumes that you already read the Fancybox HowTo *(http://fancybox.net/howto)* and included all necessary jquery and fancybox libraries. The rest is pretty simple - you just need to fire up the plugin, using jquery selector (**a.gallery_thumbnail**) and provide it with the different options you want to have.

Here we also added a few lines of code to make our thumbnails look even fancier. So the final result should look like this:

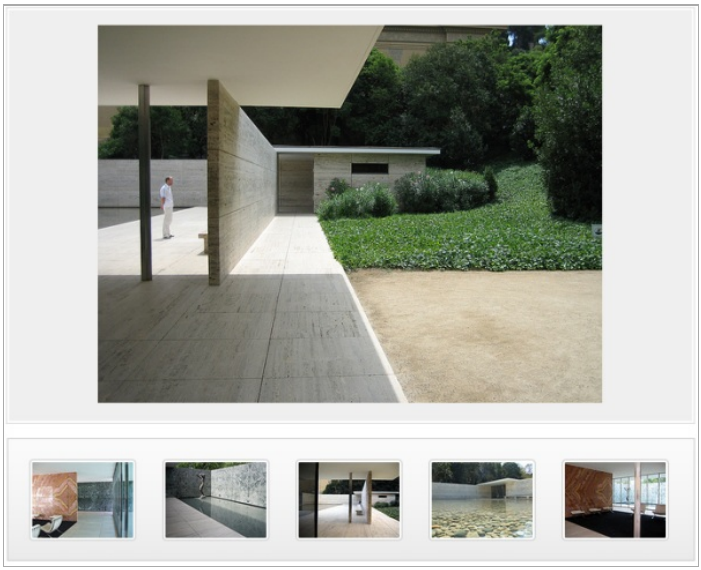
And when you click on an image, you have a bigger one with a description and arrows to navigate through the others:

ADVANCED IMAGE GALLERIES

You can also make an advanced presentation of *Images*, using a few more lines of code and plugins. The approach should be the same:

- Provide a list of links to full-size images (*<a href="{{ uri options="image" }}"...*)
- Provide a list of thumbnail images (**)

For example, you can combine thumbnails + average size images + full-size images:



47. CALENDARS WITH JQUERY

Usually you will want to use a calendar to show a visual presentation of published articles in the archive. In Newscoop, we added a special function to provide a Calendar control, which will highlight all days that have articles (or issues) published. When a user clicks on a date, a link action fires up to open a defined URL with an additional parameter *date*. It could be used to address an archive template and sort articles according to the provided date.

The calendar function is based on the jQueryUI widget **Datepicker** (http://jqueryui.com/demos/datepicker/). So the first thing to do will be to include all required jQuery libraries:

```
1. <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/
   jquery/1.5.2/jquery.min.js"></script>
2. <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/
   jqueryui/1.8.11/jquery-ui.min.js"></script>
3. <link type="text/css" href="http://ajax.googleapis.com/ajax/dlibs/jqueryu
   i/1.8.11/themes/base/jquery-ui.css" rel="stylesheet" />
4. <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/
   jqueryui/1.8.11/i18n/jquery.ui.datepicker-ru.min.js"></script>
```

As you can see in this example, we included all libraries from Google's CDN. You can use your own custom libraries, downloaded from the jQuery site. The last one is only required if you want to show a calendar in a language other than English. By default, the Calendar will be presented in the current site language. A function detects what language the website content is, and therefore what language the Calendar will be.

The following construction will create a Calendar control::

```
{{ calendar container="cal1Container" url="/?tpl=archive.tpl" style="my_calenda
r" min_date="2009-03-01" max_date="2009-03-31" clickable_dates="articles" }}
```

- container: This attribute is mandatory. It allows you to define an unique HTML identifier (id) for the calendar container.
- url: This attribute is mandatory. Define the action link of the Calendar once the user clicks a date.
- clickable_dates: This attribute is optional. There are two possible values for this attribute: "articles" and "issues", with the latter being the default. If you pass "articles", the Calendar will show as highlighted all dates where at least one article is published (publish_date). If you pass "issues" or nothing at all, thr Calendar will show as highlighted all dates when an issue has been published.
- style: This attribute is optional. Sets a custom CSS class for the calendar container. If it is not defined, the default class "calendar" will be used.
- min_date: This attribute is optional. It sets the Calendar's minimum selectable date. The date format should be "yyyy-mm-dd".
- max_date: This attribute is optional. It sets the Calendar's maximum selectable date. The date format should be "yyyy-mm-dd".

The resulting output will be like this (remember, we included the Russian language in the example above):

You can then style the Calendar the way you want it to be (remember, we used the class name "my_calendar"). Also, the look is based on the UI theme that you use, so you can adjust the theme according to your needs (we used the base theme in this example).

The source code of the Calendar output is the following:

```
1.  <!-- Begin Calendar widget //-->
2.  <div class="my_calendar">
3.  <div id="calIContainer"></div>
4.  <script type="text/javascript">
5.  $(function() {
6.    var issueDates = new Array();
7.    var articleDates = new Array("2009-03-27","2009-03-28");
8.    $.datepicker.setDefaults($.datepicker.regional["en"]);
9.    $("#calIContainer").datepicker({
10.     dateFormat: "yy-mm-dd",
11.     minDate: "2009-03-01",
12.     maxDate: "2009-03-31",
13.     beforeShowDay: function(displayedDate) {
14.                     var dDate = "";
15.                     var dDay = displayedDate.getDate();
16.                     var dMonth = displayedDate.getMonth() + 1;
17.
18.                     if (dDay < 10) dDay = "0" + dDay;
19.                     if (dMonth < 10) dMonth = "0" + dMonth;
20.                     dDate = displayedDate.getFullYear() + "-" + dMonth + "-
    " + dDay;
21.
22.                     for (i = 0; i < issueDates.length; i++) {
23.                         if (dDate == issueDates[i]) {
24.                             return [false,"ui-state-active",""];
25.                         }
26.                     }
27.                     for (i = 0; i < articleDates.length; i++) {
28.                         if (dDate == articleDates[i]) {
29.                             return [true, "ui-state-active", ""];
30.                         }
31.                     }
32.                     return [false, ""]; // disable all other days
33.                 },onSelect: function(selectedDate) {
34.                     var contentDates = new Array("2009-03-27","2009-03-28")
    ;
35.                     for (var i in contentDates) {
36.                         if (selectedDate == contentDates[i]) {
37.                             var linkTo = "/?tpl=archive.tpl&date=" + select
    edDate;
38.                             window.location = linkTo;
39.                         }
```

```
                    }
                }});
});
</script>
</div>
<!--End Calendar widget //-->
```

EXTENDING FUNCTIONALITY

You can see that all the hard work of creating the visual presentation is given to jQuery UI Datepicker. All we need to do is provide dates in correct format. You can take this approach and provide any data you want. Just note that it should be readable by the plugin.

Of course, it is much more easier to use a Calendar function supplied in Newscoop, but sometimes you will need more. Here is how we made a Calendar of PDF versions in the archive. First we need to provide variables in JSON format. For that purpose we have made a small template (_json-pdf-items.tpl):

```
{
"pdf":{
  {{list_articles constraints="type is pdf issue greater 1 section is 300 publish_date greater 1900-01-01"}}
    "{{$gimme->article->publish_date|date_format:"%Y-%m-%d"}}":{
      "link":
        "{{list_article_attachments length="1"}}http://{{$gimme->publication->site}}{{uri options="articleAttachment"}}{{/list_article_attachments}}",
        "title":" {{$gimme->issue->name}}({{$gimme->issue->number}})  {{$gimme->issue->publish_date|camp_date_format:"%d.%m.%Y"}} -  pdf"}{{if !$gimme->current_list->at_end}},{{/if}}{{/list_articles}}
      }
}
```

Next, we need to read variables from that template and save them for jQuery. Actually, we use a cron-job to grab the data and put it into the .json file. But it could also be done directly, addressing the template:

```
/* Datepickers */
var pdf_date;
$.ajax({async:false,dataType:'json',url:'/?tpl=_json-pdf-items.tpl',success: function(d){pdf_date=d;}});
```

The contents of the JSON-formatted file will be something like this:

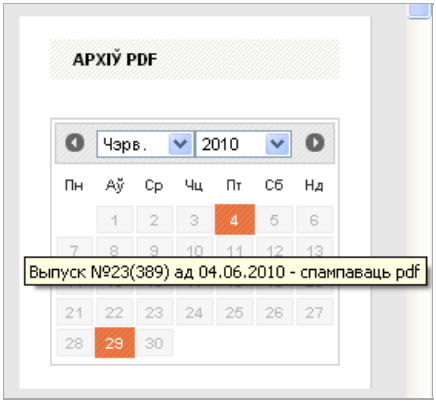
```
{
  "pdf":{
    "2010-06-29":{"link":"http://ganc-chas.by/attachment/000000026.pdf","title":" 25(391)  17.06.2010 -  pdf"},
    "2010-06-04":{"link":"http://ganc-chas.by/attachment/000000025.pdf","title":" 23(389)  04.06.2010 -  pdf"}
  }
}
```

And here's a jQuery part of the stuff (we use a slightly modified version which could be found at http://ganc-chas.by/templates/js/functions.js):

```
// PDF Archive Calendar
$("#PDFArchive").datepicker({
  changeMonth: true,
  changeYear: true,
  beforeShowDay: function(d){
    var date = $.datepicker.formatDate("yy-mm-dd",d);
    show=false;;caption="";link="#";
```

```
 8.     if (pdf_date.pdf[date]){
 9.       show=true;
10.       style="ui-state-attention active";
11.       caption=pdf_date.pdf[date].title;
12.       link=pdf_date.pdf[date].link;
13.     }
14.     return [show,style,caption,link];
15.   }
16. });
```

The final output will be a small nifty Calendar containing all your PDF versions (you can see it in action at http://ganc-chas.by/by/page/archive/):



Calendar interface using jQuery.

48. USING POLLS WITH AJAX

In this chapter you will find an advanced way to use the polls provided with Newscoop. Using AJAX and jQuery, the user interaction becomes smooth, and no visible page reload is required to add a vote and display the poll results. Any modern jQuery version does the trick, and needs to be called in the header of the page. You can find the documentation of the original templates on the Sourcefabric website, in the wiki.

First, we will need to list the current Poll, in our case, the latest one. When loading the page, no poll action has happened yet (meaning: voting) so we display the poll. And most importantly, a DIV container should be defined, which will be used to send and receive data via AJAX.

```
1. {{list_polls length="1" order="bynumber desc"}}
2. {{if !$gimme->poll_action->defined}}
3.   <p class="title">Poll of the day</p>
4.   <p class="poll-question">{{$gimme->poll->question}}</p>
5.   <div id="poll" class="poll">
6. {{/if}}
7. ...
```

Next we need to check if the poll is still open to votes, and if no data is sent yet. We do not use a submit_button, as the data will be sent by an AJAX call.

```
1. {{if $gimme->poll->is_votable && !$gimme->poll_action->defined}}
2.  {{poll_form template="poll.tpl" submit_button=false}}
3.   {{list_poll_answers order="byOrder asc"}}
4.     {{pollanswer_edit}}{{$gimme->pollanswer->answer}}<br />
5.   {{/list_poll_answers}}
6.  <input type="submit" id="submit_poll" class="button" value="I think so!"
    />
7.  {{/poll_form}}
8. ...
```

The same template is used to provide output, if the data was sent by the user.

```
1. {{else}}
2.   <p>Voted: {{$gimme->poll->votes}}</p>
3.   {{list_poll_answers order="byOrder asc"}}
4.     <div class="answer">{{$gimme->pollanswer->percentage|string_format:"%
   d"}}%: {{ $gimme->pollanswer->answer }}
5.       <div style="width:{{$gimme->pollanswer->percentage|string_format:"%
   d"}}%;background:#5d4040;"> </div>
6.     </div>
7.   {{/list_poll_answers}}
8. {{/if}}
```

And finally: the closing tag.

```
1. ...
2. {{if !$gimme->poll_action->defined}}
3.   </div>
4. {{/if}}
5. {{/list_polls}}
```

The jQuery part of the code is pretty small. We use $.post to parse the form data, send it via AJAX to the server, and place the resulting output where it should be ( ).

```
1. $(document).ready(function(){
```

```
  $('#submit_poll').click(function(){
    $.post(
      $('form[name=poll]').attr("action"),
      $('form[name=poll]').serialize(),
      function(data){$('#poll').html(data);
    });
    return false;
  });
});
```

Below you can see the entire template poll.tpl, which you can include in your page.

```
{{list_polls length="1" order="bynumber desc"}}
{{if !$gimme->poll_action->defined}}
  <p class="title">Poll of the day</p>
  <p class="poll-question">{{$gimme->poll->question}}</p>
  <div id="poll" class="poll">
{{/if}}
{{if $gimme->poll->is_votable && !$gimme->poll_action->defined}}
  {{poll_form template="poll.tpl" submit_button=false}}
  {{list_poll_answers order="byOrder asc"}}
    {{pollanswer_edit}}{{$gimme->pollanswer->answer}}<br />
  {{/list_poll_answers}}
  <input type="submit" id="submit_poll" class="button" value="I think so!" />
  {{/poll_form}}
{{else}}
  <p>Voted: {{$gimme->poll->votes}}</p>
  {{list_poll_answers order="byOrder asc"}}
    <div class="answer">{{$gimme->pollanswer->percentage|string_format:"%d"}}%: {{ $gimme->pollanswer->answer }}
      <div style="width:{{$gimme->pollanswer->percentage|string_format:"%d"}}%;background:#5d4040;"> </div>
    </div>
  {{/list_poll_answers}}
{{/if}}
{{if !$gimme->poll_action->defined}}
  </div>
{{/if}}
{{/list_polls}}

<script type="text/javascript">
$(document).ready(function(){
  $('#submit_poll').click(function(){
    $.post(
      $('form[name=poll]').attr("action"),
      $('form[name=poll]').serialize(),
      function(data){$('#poll').html(data);
    });
    return false;
  });
});
</script>
```

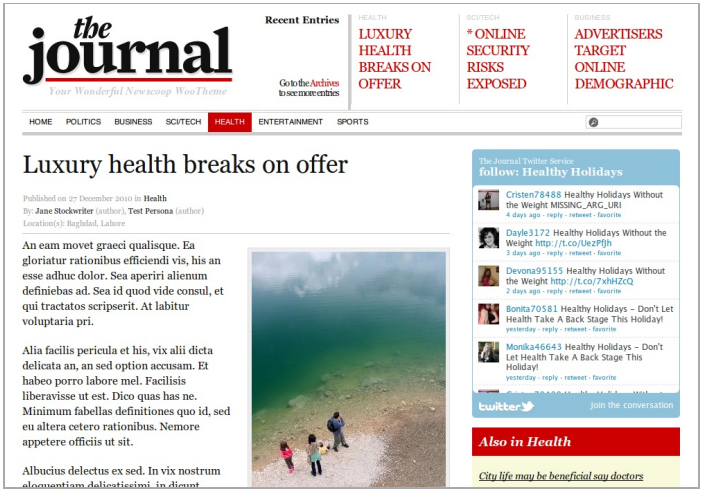
49. WIDGETS: YOUTUBE, VIMEO, TWITTER, FLICKR & CO.

If you want to include streams from Twitter, Flickr or Vimeo, relax. Just copy and paste their code into your templates. No magic. You don't need to install any widgets or plugins to do this. Just use their embed code. If you want to embed a YouTube video, just copy the embed code into your WYSIWYG editor and save. Done.

Now we cleared the air, and can dive into more interesting ideas...

TWITTER FEED WITH CUSTOM HASHTAG / SEARCH FOR EACH ARTICLE

Wouldn't it be cool to have a Twitter Feed for each article following a certain Hashtag or search word related to the story, like #sourcefabric, #climate change, #wadr, #elvis (these are examples of what your journalists would type in).



The journalist can set the Hashtag or search term in the Article Edit screen. The template will check if the field is empty, and then either include the Twitter Feed or not. In our example, the Article Field for the Hashtag is called *twittersearch* and needs to be added in the Newscoop administration interface under **Configure Article Types**.

```
{{ if $gimme->article->twittersearch|strip_tags|trim !== "" }}
<script src="http://widgets.twimg.com/j/2/widget.js"></script>
<script>
new TWTR.Widget({
  version: 2,
  type: 'search',
  search: '\"{{ $gimme->article->twittersearch }}\"',
  interval: 6000,
  title: '{{ $gimme->publication->name }} Twitter Service',
  subject: 'follow: {{ $gimme->article->twittersearch }}',
  width: 250,
  height: 300,
  theme: {
```

```
14.     shell: {
15.       background: '#8ec1da',
16.       '#ffffff'
17.     },
18.     tweets: {
19.       background: '#ffffff',
20.       '#444444',
21.       links: '#1985b5'
22.     }
23.   },
24.   features: {
25.     scrollbar: false,
26.     loop: true,
27.     live: true,
28.     hashtags: true,
29.     timestamp: true,
30.     avatars: true,
31.     toptweets: true,
32.     behavior: 'default'
33.   }
34. }).render().start();
35. </script>
36. {{ /if }}
```

VIDEO EMBEDDING IN CUSTOM SIZES

You can copy and paste embed code into your article without any problems. If you follow the examples below, you will get an idea how you can have more control over the layout in the template - and possibly embed the same video in different sizes, depending on where it is shown. Also, you might not want to include embed code inside the article body for various reasons, like delivering to different platforms and devices, or copyright.

Vimeo video embed from video ID

In this example the journalist copies and pastes the video ID from the vimeo URL into the article. The template will check if a video ID is given, and then create the embed code accordingly - you can control the size in the template. If you take the video embed code from vimeo, it starts like this:

```
<iframe src="http://player.vimeo.com/video/12790651?title=0&amp;byline=0&amp;po
rtrait=0" width="400" height="225" frameborder="0"></iframe>
```

If you compare this with the URL, you see that the video ID is the last part of the URL:

```
http://vimeo.com/12790651
```

The value the journalist has added to the article is 12790651 in the field *vimeoid*. Now you can create the embed code like this:

```
1. {{ if $gimme->article->vimeoid|strip_tags|trim !== "" }}
2. <iframe src="http://player.vimeo.com/video/{{ $gimme->article->vimeoid }}
   ?title=0&amp;byline=0&amp;portrait=0" width="300" height="180" frameborde
   r="0"></iframe>
3. <em><a href="http://vimeo.com/{{ $gimme->article->vimeoid }}" target="_bl
   ank">Watch video in separate window</a></em>
4. {{ /if }}
```

Make sure that the URL in the iframe is on one line.

YouTube video embed from URL

For this example, journalists will copy and paste the share URL from YouTube into an article field of their story. This way you can display Youtube videos on your site, but without your journalists embedding code in the WYSIWYG editor field. (This is for various different reasons; one of them might be the custom size of your space dedicated to the YouTube embed). You can use the link which YouTube offers by default (click on "share" under the video):

```
http://youtu.be/NokMkmthduY
```

What you need to do then is to extract just the important part *NokMkmthduY* from the field and put it in a wrapper with predefined dimensions, transparency options etc. This is how you can do it:

```
1. <script type="text/javascript">
2. var embedParts="{{ $gimme->article->youtubeurl }}".split("/");
3. document.write("<iframe title=\"YouTube video player\" width=\"435\" heig
   ht=\"356\" src=\"http://www.youtube.com/embed/"+embedParts[3]+"?wmode=opa
   que\" frameborder=\"0\" allowfullscreen></iframe>");
4. </script>
```

CUSTOM FLICKR STREAM FOR INDIVIDUAL ARTICLES

Imagine a journalist writes an article about an event which has its own flickr account. The widget below allows you to call in the latest images from that flickr stream and display them alongside the article. This widget does not show up if a flickr name is not provided by the journalist.

In our example, the Article Field for the flickr user name is called *flickrname* and needs to be added in the Newscoop administration interface under **Configure Article Types**.

Firstly, the template checks if the name is set, and if so then (prior to the PHP code) the value is assigned to a variable of the same name. This is necessary because inside PHP the environment has no access to the $gimme object. The styling in this example is included in the code, but you can place it elsewhere. If you want to change the number of images, you can do this inside the widget by changing $rows and $columns.

```
1.  {{ if $gimme->article->flickrname|strip_tags|trim !== "" }}
2.  <style>
3.  .widget-flickr-wrapper, .widget-flickr-wrapper table { width: 100%; }
4.  .widget-flickr-wrapper table,
5.  .widget-flickr-wrapper table td img { background: #f0f0f0; border: 1px so
    lid #e7e7e7; }
6.  .widget-flickr-wrapper table td { padding: 10px;  }
7.  </style>
8.  <h3>Latest images from &raquo;{{ $gimme->article->flickrname }}&laquo;</h
    3>
9.  {{ assign var="flickrname" value=`$gimme->article->flickrname` }}
10. {{ php }}
11. $rows = 4;
12. $colums = 3;
13. $api = '40a7ac8e64e9bc034d0f72b9a64e1f3a';
14. $username = $this->get_template_vars('flickrname');
15. $total = $rows * $colums;
16. // get user ID from user name
17. $user_id = simplexml_load_file("http://flickr.com/services/rest/?method=f
    lickr.people.findByUsername&api_key=".$api."&username=".$username)->user[
    'nsid'];
18. // get images from user ID
19. $xml = simplexml_load_file("http://flickr.com/services/rest/?method=flick
    r.photos.search&api_key=".$api."&user_id=".$user_id."&per_page=".$total);
20. if (isset($xml->photos)) {
21.   $photos = array();
22.   foreach ($xml->photos->photo as $photo) {
23.     $photos[] = array(
24.       'src' => "http://farm1.static.flickr.com/".$photo['server']."/".$ph
    oto['id']."_".$photo['secret']."_m.jpg",
25.       'link' => "http://www.flickr.com/photos/".$user_id."/".$photo['id']
    ,
26.       'title' => $photo['title']
```

```
    );
  }
  echo "<div class=\"widget-flickr-wrapper\">\n<table border=\"0\" cellpadding=\"0\" cellspacing=\"0\">\n";
  for ($i = 0; $i < $rows; $i++) {
    echo "<tr>\n";
    for ($j = 0; $j < $colums; $j++) {
      $index = $colums * $i + $j;
      if (isset($photos[$index])) {
        $photo = $photos[$index];
        echo "<td>\n<div class=\"widget-flickr-thumb\">\n<a href=\"".$photo['link']."\" target=\"_blank\"><img src=\"".$photo['src']."\" height=\"75\" width=\"75\" alt=\"".$photo['title']."\" /></a>\n</div>\n</td>\n";
      } else {
        echo "<td> </td>\n";
      }
    }
    echo "</tr>\n";
  }
  echo "</table>\n</div>\n";
}
{{ /php }}
{{ /if }}
```

50. PROFILE PICTURES: GRAVATAR, FACEBOOK, TWITTER

Displaying profile pictures alongside comments can make your article a more personal and engaging space. Below you can find ways to call in images from Gravatar, Facebook or Twitter.

Gravatar

Gravatar is a widely used service, providing profile images based on an e-mail address. You can use this service in comments to provide images alongside a comment. If an email is given by the user, the standard image (80 x 80 pixel) is retrieved by calling PHP from the template like this:

```
1. {{ assign var="profile_email" value=`$gimme->comment->reader_email` }}
2. {{ php }}
3. $profile_email = $this->get_template_vars('profile_email');
4. print "<img src=\"http://www.gravatar.com/avatar/".md5( strtolower( trim(
    $profile_email ) ) )."\" />";
5. {{ /php }}
```

If the user does not have a Gravatar account, the default Gravatar logo is displayed. You can change the size of the image provided by adding a parameter as shown below. Gravatar images are always square. The following example will deliver an image which is 200 x 200 pixels in size:

```
1. {{ assign var="profile_email" value=`$gimme->comment->reader_email` }}
2. {{ php }}
3. $profile_email = $this->get_template_vars('profile_email');
4. print "<img src=\"http://www.gravatar.com/avatar/".md5( strtolower( trim(
    $profile_email ) ) )."?s=200\" />";
5. {{ /php }}
```

Facebook

Calling an image from Facebook is very straightforward. The example below calls in the Sourcefabric image:

```
<img src=\"http://graph.facebook.com/Sourcefabric/picture\" />
```

In order to make this work with comments, you could use the *nickname* input field and label it something like "*Facebook name (optional)*". The code below checks if an image is available on Facebook, and if not, does not display anything.

```
1. {{ assign var="profile_fbname" value=`$gimme->comment->nickname` }}
2. {{ php }}
3. $profile_fbname = $this->get_template_vars('profile_fbname');
4. if (!preg_match("/error/i", file_get_contents("http://graph.facebook.com/
   ".$profile_fbname."/picture"))) {
5.   print "<img src=\"http://graph.facebook.com/".$profile_fbname."/picture
   \" />";
6. }
7. {{ /php }}
```

If you are providing a reader registration form for your publication, you can add a custom field to the user profile for the Facebook name. In the template you can check if the user is logged in, and call the image from the Facebook name provided in the user profile.

Twitter

The easiest way to retrieve the profile image from Twitter goes like this:

```
<img src="http://img.tweetimag.es/i/Sourcefabric" />
```

You can make this work with comments by using the *nickname* input field and labelling it something like "Twitter name (optional)". To display the image, include the following code:

```
1. {{ assign var="profile_twname" value=`$gimme->comment->nickname` }}
2. {{ php }}
3. $profile_twname = $this->get_template_vars('profile_twname');
4. print "<img src=\"http://img.tweetimag.es/i/". $profile_twname ."\" />";
5. {{ /php }}
```

You might want to check if the name provided actually does exist:

```
1. {{ assign var="profile_twname" value=`$gimme->comment->nickname` }}
2. {{ php }}
3. $profile_twname = $this->get_template_vars('profile_twname');
4. $api_call = "http://twitter.com/users/show/".$profile_twname.".json";
5. $results = json_decode(file_get_contents($api_call));
6. if (!empty($results)) {
7. print "<img src=\"".$results->profile_image_url."\" />";
8. }
9. {{ /php }}
```

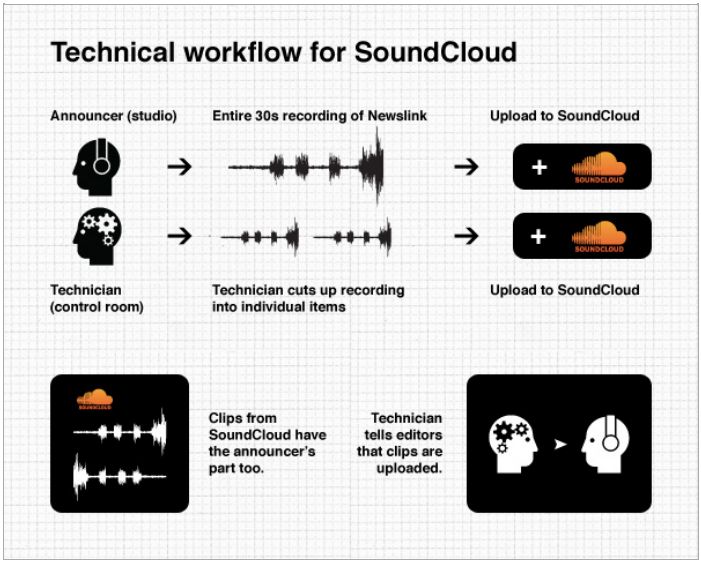
51. COMBINING NEWSCOOP AND SOUNDCLOUD

Newscoop 3.6 includes a plugin for uploading, managing and displaying sound clips which are stored on the **SoundCloud** service (http://soundcloud.com/). This function was first implemented by Sourcefabric for its partner, **West Africa Democracy Radio** (http://www.wadr.org), a news and talk radio station which provides programming to 30 affiliates in West Africa. WADR creates 5-6 hours per day of original content, and all of this content is uploaded to SoundCloud.

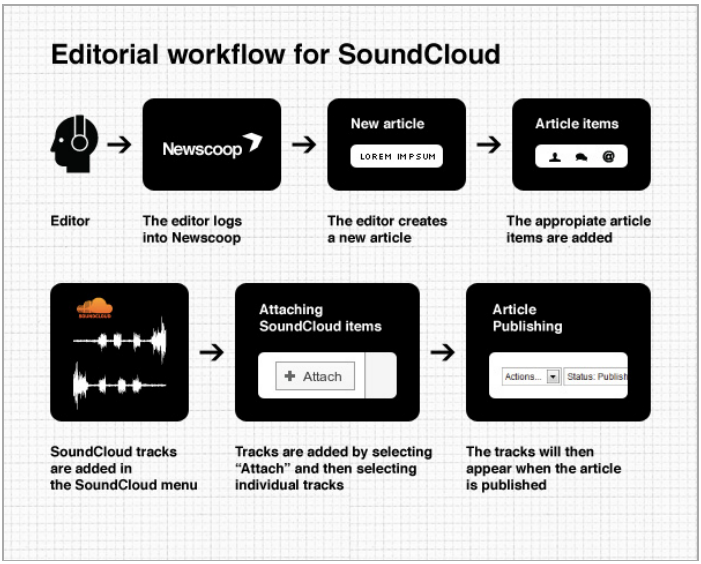
There were three major benefits for WADR in using SoundCloud as a distribution channel.

- The SoundCloud distribution network is fast, so listeners do not have to wait a long time for files to download
- The SoundCloud players allow third parties to embed shows and clips on blogs and other websites, thus allowing WADR to reach a larger audience than it would if the clips were stored exclusively on its own site
- SoundCloud's paid premium accounts allow for a large amount of items to be stored on the service, and the prices were competitive with other hosted solutions

Here is a diagram showing the workflow stages for technical and editorial staff. The first shows workflow for technical staff:



On the editorial side, editors make use of the SoundCloud plugin to attach uploaded clips to their articles:



Using SoundCloud in your templates

Once SoundCloud tracks are uploaded and attached to articles in Newscoop, they can then be called by the templates just like any other content on the site. The following example lists any SoundCloud tracks that have been attached to an article and calls the SoundCloud player.

This directive gets the track's secret URI, which is then passed to the SoundCloud player for playing out the track:

```
{{ $soundcloud->track.secret_uri }}
```

This directive tells Newscoop to list the available SoundCloud tracks for the given article:

```
{{ list_soundcloud_tracks article=$gimme->article->number }}
```

Here is the entire template for listing the available tracks for the article, calling the SoundCloud player, and setting up the player for the track:

```
1. {{ list_soundcloud_tracks article=$gimme->article->number }}
2. <object height="81" width="100%">
3. <param name="wmode" value="transparent">
4. <param name="movie" value="http://player.soundcloud.com/player.swf?url={{
    $soundcloud->track.secret_uri }}&amp;show_comments=true&amp;auto_play=fa
   lse&amp;></param>
5. <param name="allowscriptaccess" value="always"></param>
6. <embed allowscriptaccess="always" height="81" src="http://player.soundclo
   ud.com/player.swf?url={{ $soundcloud->track.secret_uri }}&amp;show_commen
   ts=true&amp;auto_play=false&amp; type="application/x-shockwave-flash" wid
   th="100%"></embed>
7. </object>
8. {{ /list_soundcloud_tracks }}
```

WORKING WITH SOUNDCLOUD SETS

SoundCloud can also prepare and play groups of tracks (like playlists), which it calls sets. In the WADR project, every program has a set player, and then every individual broadcast is added to the set, with the newest broadcast presented at the top of the set.

For its premium customers, SoundCloud provides a special Set Player which can be embedded instead of a track player. The premium player also allows custom artwork and dimensions. Here is an example of how Newscoop works with a SoundCloud Mini Player which handles a set:

```
1.  <div class="miniPlayer">
2.  <object height="300" width="300">
3.  {{ if $gimme->language->english_name == "English" }}
4.  <param name="movie" value="http://player.soundcloud.com/player.swf?url=ht
    tp%3A%2F%2Fapi.soundcloud.com%2Fplaylists%2F662124&amp;auto_play=false&am
    p;player_type=artwork&amp;></param>
5.  <param name="allowscriptaccess" value="always"></param>
6.  <param name="wmode" value="window"></param>
7.  <embed wmode="window" allowscriptaccess="always" height="300" src="http:/
    /player.soundcloud.com/player.swf?url=http%3A%2F%2Fapi.soundcloud.com%2Fp
    laylists%2F662124&amp;auto_play=false&amp;player_type=artwork&amp; type="
    application/x-shockwave-flash" width="300"></embed>
8.  {{ else }}
9.  <param name="movie" value="http://player.soundcloud.com/player.swf?url=ht
    tp%3A%2F%2Fapi.soundcloud.com%2Fplaylists%2F714819&amp;auto_play=false&am
    p;player_type=artwork&amp;></param> <param name="allowscriptaccess" value
    ="always"></param> <embed allowscriptaccess="always" height="300" src="ht
    tp://player.soundcloud.com/player.swf?url=http%3A%2F%2Fapi.soundcloud.com
    %2Fplaylists%2F714819&amp;auto_play=false&amp;player_type=artwork&amp; ty
    pe="application/x-shockwave-flash" width="300"></embed>
10. {{ /if }}
11. </object>
12. </div>
```

For further reference, the SoundCloud API documentation is available at:

http://developers.soundcloud.com/docs/api

52. WORKING WITH FACEBOOK

It's very easy to integrate Facebook's social plugins with Newscoop pages. In most cases, it's a matter of adding a few lines of the code Facebook provides.

Most of the features that Facebook provides can be implemented in two different ways: the classic way using the <iframe> tag, and the modern way using JavaScript and XFBML. The new way of implementing Facebook features has just two lines of code:

```
1. <script src="http://connect.facebook.net/en_US/all.js#xfbml=1"></script>
2. <fb:facepile href="example.com" width="200" max_rows="1"></fb:facepile>
```

MAKING YOUR PAGE LOOK GOOD IN FACEBOOK USING OPEN GRAPH PROTOCOL

Open Graph Protocol makes sure your Newscoop-powered pages are handled by Facebook in the correct way. If they are implemented right, they can be used on Facebook profiles under the "Likes and Interests" section.

You'll need to add Open Graph protocol <meta> tags and the *Like* button to your header template in order to turn your pages into graph objects. The example below shows which <meta> tags can be used, and what their content should be.

```
1. <title>{{ if $gimme->article->defined }}{{ $gimme->article->name }} | You
   r Site
2. {{ else }}Your Site
3. {{ /if }}
4. </title>
5. {{ if $gimme->article->defined }}
6. <meta property="og:title" content="{{$gimme->article->name|html_entity_de
   code|regex_replace:'/&(.*?)quo;/':'&quot;'}}" />
7. <meta property="og:type" content="article" />
8. <meta property="og:url" content="http://{{ $gimme->publication->site }}{{
    uri }}" />
9. <meta property="og:site_name" content="YourSite.com" />
10. {{ if $gimme->article->type_name == "news" }}
11.   <meta property="og:description" content="{{$gimme->article->deck|strip_
    tags:false|strip|escape:'html':'utf-8' }}" />
12. {{ elseif $gimme->article->type_name == "show" }}
13.   <meta property="og:description" content="{{$gimme->article->short_descr
    iption|strip_tags:false|strip|escape:'html':'utf-8' }}" />
14. {{ /if }}
15. {{ list_article_images }}
16. <meta property="og:image" content="{{ $gimme->article->image->imageurl }}
    " />
17. {{ /list_article_images }}
18. {{ if $gimme->prev_list_empty }}
19. <meta property="og:image" content="http://{{ $gimme->publication->site }}
    /templates/assets/images/logo.png" />
20. {{ /if }}
21. {{ /if }}
```

We've set up an IF statement that first handles whether an article has been defined. In the first <meta> tag, we're doing a few things to clean up the text and get it ready for Facebook:

```
<meta property="og:title" content="{{$gimme->article->name|html_entity_decode|r
egex_replace:'/&(.*?)quo;/':'&quot;'}}" />
```

We're combining three things together in this meta tag:

- a directive to get the article name: $gimme->article->name
- a Smarty function to decode the HTML entity
- a Smarty function to replace any illegal characters

The next one is pretty clear. Our Open Graph type is an article:

```
<meta property="og:type" content="article" />
```

The next meta tag is for the URL:

```
<meta property="og:url" content="http://{{ $gimme->publication->site }}{{ uri }
}" />
```

The og:site tag is really straightforward:

```
<meta property="og:site_name" content="YourSite.com" />
```

The og:description can be kind of complicated. In this case, it's conditional depending on the article type. If the article type is "news", it will display one description from the *deck* field, but if the article type is "show" the description will be different, taken from the *short_description* field. In the case of "news", it strips out any illegal characters from the *deck* field and uses the result as what Open Graph Protocol calls *content*:

```
1. {{ if $gimme->article->type_name == "news" }}
2.   <meta property="og:description" content="{{$gimme->article->deck|strip_
   tags:false|strip|escape:'html':'utf-8' }}" />
3. {{ elseif $gimme->article->type_name == "show" }}
4.   <meta property="og:description" content="{{$gimme->article->short_descr
   iption|strip_tags:false|strip|escape:'html':'utf-8' }}" />
5. {{ /if }}
```

The final conditions have to do with the display of article images. We're taking the list of available article images and passing that to Facebook as the potential images that a user sharing a link can flip through. But if there isn't an image attached to the article (that's the IF statement for {{ if $gimme->prev_list_empty }} command), then it displays the site logo:

```
1. {{ list_article_images }}
2. <meta property="og:image" content="{{ $gimme->article->image->imageurl }}
   " />
3. {{ /list_article_images }}
4. {{ if $gimme->prev_list_empty }}
5. <meta property="og:image" content="http://{{ $gimme->publication->site }}
   /templates/assets/images/logo.png" />
6. {{ /if }}
```

You can get more info about the Open Graph protocol on Facebook's developer pages: http://developers.facebook.com/docs/opengraph/

ADDING A LIKE BUTTON TO YOUR PAGE

The Like button is the most popular Facebook feature. The team at Facebook made adding a Like button very easy, by going to a particular page and generating the code needed.

Here's an excerpt from a Newscoop article page template that includes a Like button:

```
<iframe src="http://www.facebook.com/plugins/like.php?href={{ url|escape:"url"
}}&amp;layout=standard&amp;show_faces=true&amp;width=400&amp;action=recommend&a
mp;font=arial&amp;colorscheme=light" scrolling="no" frameborder="0" allowTransp
```

```
arency="true" style="border:none; overflow:hidden; width:400px; height:px"></if
rame>
```

In this case, we're using Newscoop to generate the article's URL and pass that to Facebook so that it knows where to send its users when they see the article in their pages. We're also adding a Smarty function to escape the URL so that Facebook can handle it in the right way. That's what's going on in this part:

```
 src="http://www.facebook.com/plugins/like.php?href={{ url|escape:"url" }}
```

More info about the Like button can be found on the Facebook developer pages: http://developers.facebook.com/docs/reference/plugins/like/

ADDING A LIKE BOX

The *Like Box* feature enables Facebook users to bring the "like" activity to their own site. The Like Box enables readers of your publication to:

- See how many people liked your Facebook page, and which of their friends like it too
- Read recent posts from the Facebook page
- Like the Facebook page with just one click, without needing to visit Facebook directly

Here is an example of a Like Box used in Newscoop templating:

```
<script src="http://connect.facebook.net/en_US/all.js#xfbml=1"></script><fb:lik
e href="{{ uri options="article" }}" layout="button_count" show_faces="true" wi
dth="450" font=""></fb:like>
```

The other options allow you to:

- Set the Like Box's width in pixels
- Show your fans' profile pictures (show_faces="true")
- Set the type of button layout (see more about this on the Facebook developer pages)

More information about the Like Box can be found on the Facebook developer pages: http://developers.facebook.com/docs/reference/plugins/like-box/

ADDING FACEBOOK COMMENTS

The Comments Box feature can help your replace the built-in commenting system of Newscoop with the one provided by Facebook. Facebook can also provide moderation and distribution tools with this feature.

The Comments Box can be used only with XFBML and the JavaScript file provided by Facebook:

```
1. <div id="fb-root"></div>
2. <script src="http://connect.facebook.net/en_US/all.js#appId=APP_ID&amp;xf
   bml=1"></script>
3. <fb:comments href="sourcefabric.org" num_posts="2" width="500"></fb:comme
   nts>
```

More info about Facebook Comments can be found on the Facebook developer pages: http://developers.facebook.com/docs/reference/plugins/comments/

53. WORKING WITH ADVERTISING

Generating revenue with advertisements usually means including code from advertisers or ad networks in your pages. In Newscoop this means incorporating ads in an existing template, or creating special templates for different campaigns. As shown in this Cookbook, there are many ways you can make content related decisions on which advertisement to include, for example based on article type, keywords, topics, section and so on.

At the end of this chapter you'll find a suggestion about how to handle your own banners. But first, let's look at external providers.

INCLUDE INSERT CODES FROM A LOCAL OPENX SERVER

Using insert codes from OpenX is quite easy. Once you have OpenX installed, you will need to define what they call "zones" for your advertising. Once defined, you can copy and paste the resulting insert code into your template. Here is a sample script for including a banner ad served from OpenX:

```
1.  <script type='text/javascript'><!--//<![CDATA[
2.     var m3_u = (location.protocol=='https:'?'https://openx.YOURSITE.COM/de
    livery/ajs.php':'http://openx.YOURSITE.COM/delivery/ajs.php');
3.     var m3_r = Math.floor(Math.random()*99999999999);
4.     if (!document.MAX_used) document.MAX_used = ',';
5.     document.write ("<scr"+"ipt type='text/javascript' src='"+m3_u);
6.     document.write ("?zoneid=66&amp;target=_top&amp;charset=UTF-8");
7.     document.write ('&amp;cb=' + m3_r);
8.     if (document.MAX_used != ',') document.write ("&amp;exclude=" + docume
    nt.MAX_used);
9.     document.write ('&amp;charset=UTF-8');
10.    document.write ("&amp;loc=" + escape(window.location));
11.    if (document.referrer) document.write ("&amp;referer=" + escape(docume
    nt.referrer));
12.    if (document.context) document.write ("&context=" + escape(document.co
    ntext));
13.    if (document.mmm_fo) document.write ("&amp;mmm_fo=1");
14.    document.write ("'><\/scr"+"ipt>");
15. //]]>--></script>
16. <noscript>
17.    <a href='http://openx.YOURSITE.COM/delivery/ck.php?n=a16e8b28&amp;cb=I
    NSERT_RANDOM_NUMBER_HERE' target='_top'>
18.    <img src='http://openx.YOURSITE.COM/delivery/avw.php?zoneid=66&amp;cha
    rset=UTF-8&amp;cb=INSERT_RANDOM_NUMBER_HERE&amp;n=a16e8b28' border='0' al
    t='' />
19.    </a>
20. </noscript>
```

For more info on OpenX, check their documentation at http://www.openx.org/support/documentation

INCLUDE INSERT CODES FROM GOOGLE ADSENSE

Once you have successfully created a publisher account with Google AdSense, you can copy and paste the insert code into your template with your publisher ID and ad slot info. Here is an example:

```
1.  <script type="text/javascript">
2.     <!--
3.     google_ad_client = "pub-XXXXXXXXXXXXXXXX";
```

```
   /* 300x250, created 6/25/09 */
   google_ad_slot = "YYYYYYYYYY";
   google_ad_width = 300;
   google_ad_height = 250;
   //-->
</script>

<script type="text/javascript"
   src="http://pagead2.googlesyndication.com/pagead/show_ads.js">
</script>
```

You can find out more about Google AdSense at http://www.google.com/adsense

HANDLING BANNERS IN NEWSCOOP TEMPLATES

You can also build your own logic for banner ads in your site. In the following example we will display different banners in different sections.

1. Create banner templates

Inside the folder of your template pack, create a new folder *_banners*. In this example we are building banners for the "Ushahidi Cooker" so the folder path is:

```
set_ushahidicooker/_banners
```

Inside this folder, create template files with banners. There is no Newscoop code inside these templates, just simple HTML displaying a banner with link or embedding a Flash banner - anything you like, we won't explain how to do these. In our example the banner templates are named:

- banner_hospitals.tpl
- banner_libraries.tpl

2. Create a template to handle the banner display

Inside the same folder, create a file called check_banner.tpl. This file contains code similar to the following (adjust it for your publication):

```
{{ if $gimme->section->number == 10 }}
  {{ include file="set_ushahidicooker/_banners/banner_hospitals.tpl" }}
{{ /if }}
{{ if $gimme->section->number == 11 }}
  {{ include file="set_ushahidicooker/_banners/banner_libraries.tpl" }}
{{ /if }}
```

This template can be extended to cover more sections.

3. Include the master banner

Now all you need to do is include the check_banner.tpl template in your publication at the place where you want to display banner ads:

```
{{ include file="set_ushahidicooker/_banners/check_banner.tpl" }}
```

Using the Newscoop template language, you can modify the IF condition any way that fits your publication's needs.

54. WAYS TO CREATE ARCHIVES

There are different approaches to creating archives. It can be:

- a list of Articles
- a list of Issues
- a list of PDF versions
- a list of Polls
- a calendar of Articles / Issues / PDF versions
- a search form to look through all materials

You would usually combine different options or make your own, depending on your needs.

GENERAL ARCHIVE TYPE BASED ON ISSUES

This is a very basic and widely used type of Archive. It is convenient even if you have many issues. You can see an example in our "Journal" template package:

```
1.  <div id="accordion">
2.  {{ list_issues constraints="number not 1" order="bypublishdate desc" }}
3.    <h3 style="margin-bottom: 20px"><a href="#">{{ $gimme->issue->name }}</
    a></h3>
4.      <div>
5.  {{ list_sections }}
6.  {{ list_articles }}
7.  {{ if $gimme->current_articles_list->at_beginning }}
8.        <h4><a href="{{ uri options="section" }}">{{ $gimme->section->name
    }}</a></h4>
9.          <ul>
10. {{ /if }}
11.           <li><a href="{{ uri options="article" }}">{{ $gimme->article->n
    ame }}</a> - {{ $gimme->article->publish_date|camp_date_format:"%e %M %Y"
     }} - {{ $gimme->article->comment_count }} comment(s)</li>
12. {{ if $gimme->current_articles_list->at_end }}
13.         </ul>
14. {{ /if }}
15. {{ /list_articles }}
16. {{ /list_sections }}
17.     </div>
18. {{ /list_issues }}
19. </div><!-- /#accordion -->
```

And after adding just a few lines of jQuery code you get a nice accordion effect:

```
1.  $(function() {
2.    $( "#accordion" ).accordion();
3.  })
```

The result will look like this:

January 2011

Politics

- European Council candidates set to be named - 27 December 2010 - 1 comment(s)
- Wintry conditions sweep across China - 27 December 2010 - 0 comment(s)
- Food export duties set to rise - 27 December 2010 - 0 comment(s)

Business

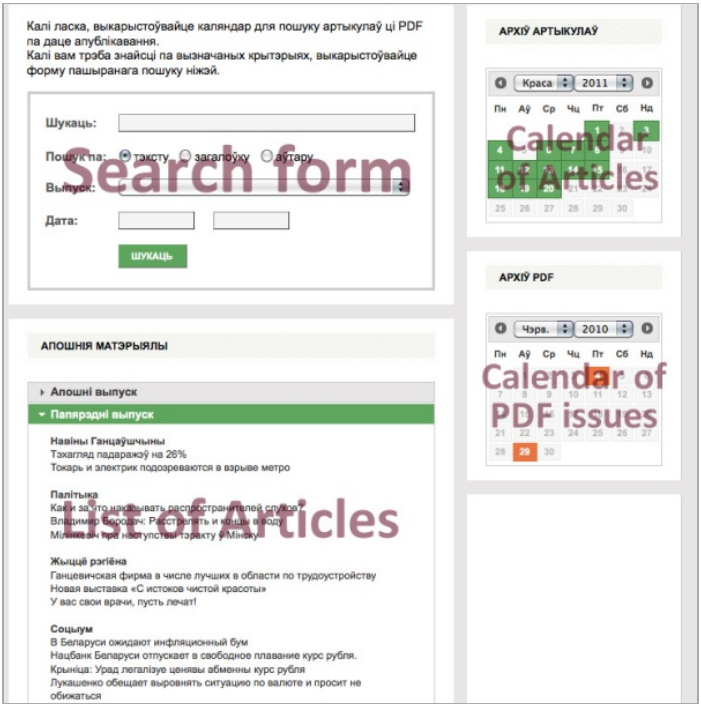
- Private aviation companies take flight - 24 December 2010 - 0 comment(s)
- Tax and trade bureau considers alcohol labeling - 27 December 2010 - 0 comment(s)
- Where to locate your internet start-up - 27 December 2010 - 0 comment(s)

Sci/Tech

- Commercial space flight becomes a reality - 24 December 2010 - 0 comment(s)

COMBINATION OF DIFFERENT TYPES OF ARCHIVES

If you have lots of issues, and especially if you plan to maintain the Archive for a long time, you might want to have a combination of different types of archives. You can take a look at an example of how it's done at: http://ganc-chas.by/by/page/archive



Implementing a search-based archive is a good practice, as it gives your readers the opportunity to find the required material in a click. We use an Advanced Search form for that purpose (please take a look at the *Template Reference* for more information about *Advanced Search*).

archive-search-form.tpl

```
1.  <div id="archive_search_form">
2.  <form method="post" action="{{uri}}" name="search_articles">
3.  <input type="hidden" value="{{$gimme->template->identifier}}" name="tpl"
    />
4.  <input type="hidden" value=true name="f_search_articles" />
5.  <fieldset>
6.          <div><label>Find:</label><input type="text" value="{{$gimme->sear
    ch_articles_action->search_phrase}}" name="f_search_keywords" /></div>
7.          <div><label>Search by:</label>
8.      <span>
9.        <input type="radio" name="f_search_scope" value="content" checked="
    checked" /> text  
10.       <input type="radio" name="f_search_scope" value="title" /> title &n
    bsp;
11.       <input type="radio" name="f_search_scope" value="author" /> author
12.     </span>
13.   </div>
14.         <div><label>Issue:</label>{{camp_select object="search" attribute
    ="issue"}}</div>
15.         <div><label>Date:</label><input type="text" id="f_search_start_da
    te" name="f_search_start_date" /><input type="text" id="f_search_end_date
    " name="f_search_end_date" /></div>
16.         <div><input type="submit" class="button-greent1" name="f_search_a
    rticles" value="Find" /></div>
17. </fieldset>
18. </form>
19. </div>
```

To make other search controls (please read the chapter on *Using Calendars*) and control the looks and flow we utilize jQuery a lot:

functions.js

```
1.  /* Archive search form & results */
2.
3.  $("form[name=search_articles] select[name=f_search_issue] option[value=1]
    ").remove(); /* remove first issue from the list */
4.  // Submitting a form
5.  $('#archive_search_form .button-greent1').live("click",function(){
6.    var url = $('#archive_search_form form[name=search_articles]').attr("ac
    tion") + '?tpl=archive-search-results.tpl';
7.    var form = $('#archive_search_form form[name=search_articles]').seriali
    ze();
8.    $.post(url, form, function(data){$('#archive_search_results').html(data
    );});
9.    return false;
10. });
11.
12. /* Datepickers */
13.
14. // Load results with AJAX if a date was clicked in a calendar
15. $("#ArticleArchive .ui-state-highlight a").live("click",function(){
16.   $("#archive_search_results").load( $(this).attr("href") + '&tpl=archive
    -search-results.tpl' );
17.   return false;
18. });
19.
20. var cs_date;var pdf_date;
21. $.ajax({async:false,dataType:'json',url:'/templates/js/calendar-items.jso
    n',success: function(d){cs_date=d;}});
```

```
22. $.ajax({async:false,dataType:'json',url:'/templates/js/pdf-items.json',su
    ccess: function(d){pdf_date=d;}});
23.
24. // Article Archive Calendar
25. $("#ArticleArchive").datepicker({
26.   changeMonth: true,
27.   changeYear: true,
28.   beforeShowDay: function(d){
29.     var date = $.datepicker.formatDate("yy-mm-dd",d);
30.     var link = "page_url?date="+date;
31.     show=false;;caption="";
32.     if ($.inArray(date,cs_date.article)!=-1){
33.       var x=0;
34.       $.grep(cs_date.article,function(n,i){if(n==date) x++;});
35.       show=true;style="ui-state-highlight active";caption+="Articles foun
    d: "+x;
36.     }
37.     return [show,style,caption,link];
38.   }
39. });
40.
41. // PDF Archive Calendar
42. $("#PDFArchive").datepicker({
43.   changeMonth: true,
44.   changeYear: true,
45.   beforeShowDay: function(d){
46.     var date = $.datepicker.formatDate("yy-mm-dd",d);
47.     show=false;;caption="";link="#";
48.     if (pdf_date.pdf[date]){
49.       show=true;
50.       style="ui-state-attention active";
51.       caption=pdf_date.pdf[date].title;
52.       link=pdf_date.pdf[date].link;
53.     }
54.     return [show,style,caption,link];
55.   }
56. });
```

And the whole process is going on in one template, which receives data and provides output.

archive-search-results.tpl

First, we need to check if the *date* parameter was supplied. In this case, we provide a list of articles published on that date:

```
1. {{if $smarty.get.date}}
2. {{assign var="get_date" value=$smarty.get.date|camp_date_format:"%Y-%m-%d
   "}}
3.   <div class="header-diag">
4.     Materials on: {{$get_date|camp_date_format:"%e %M %Y"}}
5.   </div>
```

We are trying to look for articles on a given date in sections that contain articles of type 'article':

```
1.   {{list_articles columns="15" constraints="type is article issue greater
      1 section smaller_equal 120 publish_date is `$get_date`"}}
2.     {{if $gimme->current_list->count > 15}}{{assign var="navigation" valu
    e=true}}{{/if}}
3.   {{if $gimme->current_list->row==$page}}
4.     {{if $gimme->current_list->column==1}}
5.       <dl class="b-comm">
6.     {{/if}}
7.       {{if $gimme->section->name != $section}}<dt>{{$gimme->section->name
    }}</dt>{{/if}}
8.         <dd><a href="{{uri}}">{{$gimme->article->name}}</a></dd>
9.       {{assign var="section" value=$gimme->section->name}}
10.    {{if $gimme->current_list->column==15}}
```

```
11.       </dl>
12.     {{/if}}
13.   {{/if}}
14.   {{/list_articles}}
15.   {{if $gimme->prev_list_empty}}<em>Nothing was found.</em>{{/if}}
```

Otherwise, if there was a search action, we should output search results:

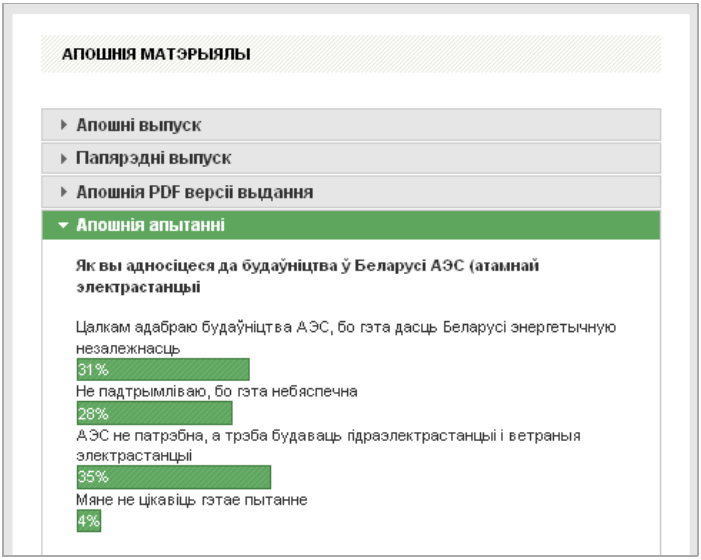
```
1.  {{elseif $gimme->search_articles_action->defined}}
2.
3.  <div class="header-diag">Search results {{if $gimme->search_articles_action->search_phrase}}for "{{$gimme->search_articles_action->search_phrase}}"{{/if}}</div>
4.
5.    {{list_search_results columns="10" order="bypublishdate desc"}}
6.      {{if $gimme->current_list->column==1}}
7.        <dl class="b-comm">
8.      {{/if}}
9.          <dt><span><a href="{{uri}}">{{$gimme->article->name}}</a></span> (#{{$gimme->issue->name}}({{$gimme->issue->number}}) {{$gimme->article->publish_date|camp_date_format:"%d.%m.%Y"}})</dt>
10.         <dd>{{$gimme->article->intro|strip_tags:false|strip|truncate:300}}</dd>
11.     {{if $gimme->current_list->column == 10}}
12.       </dl>
13.     {{/if}}
14.   {{/if}}
15.   {{/list_search_results}}
16.   {{if $gimme->prev_list_empty}}<em>Nothing was found.</em>{{/if}}
```

Else, if there's no search action or date provided, just output the list of articles in the current and previous issues:

```
1.  {{else}}
2.
3.    <div class="header-diag">Latest materials</div>
4.    <dl class="ui-accordion ui-widget">
5.
6.      <dt class="ui-accordion-header ui-state-default">Last Issue</dt>
7.      <dd class="ui-accordion-content">
8.      {{local}}{{set_current_issue}}{{assign var="issn" value=$gimme->issue->number}}{{/local}}
9.      {{list_articles constraints="type is article issue is $issn section smaller_equal 120"}}
10.       {{if $gimme->section->name != $section}}{{if $gimme->current_list->index>1}}<br />{{/if}}<p><strong>{{$gimme->section->name}}</strong></p>{{/if}}
11.       <p><a href="{{uri}}">{{$gimme->article->name}}</a></p>
12.       {{assign var="section" value=$gimme->section->name}}
13.     {{/list_articles}}
14.     {{if $gimme->prev_list_empty}}<p>Nothing was found.</p>{{/if}}
15.     </dd>
16.
17.     {{list_issues length="1" constraints="number smaller $issn" order="byPublishDate desc"}}
18.     <dt class="ui-accordion-header ui-state-default">Previous Issue</dt>
19.     <dd class="ui-accordion-content">
20.     {{list_articles constraints="type is article section smaller_equal 120"}}
21.       {{if $gimme->section->name != $section}}{{if $gimme->current_list->index>1}}<br />{{/if}}<p><strong>{{$gimme->section->name}}</strong></p>{{/if}}
22.       <p><a href="{{uri}}">{{$gimme->article->name}}</a></p>
23.       {{assign var="section" value=$gimme->section->name}}
24.     {{/list_articles}}
25.     {{if $gimme->prev_list_empty}}<p>Nothing was found.</p>{{/if}}
26.     </dd>
27.     {{/list_issues}}
```

In this example we are also listing the latest PDF issues and Polls:

```
1.    <dt class="ui-accordion-header ui-state-default">Latest PDF versions<
      /dt>
2.    <dd class="ui-accordion-content">
3.      {{list_articles constraints="type is pdf issue greater 1 section is
       300 publish_date greater 1900-01-01"}}{{list_article_attachments length=
      "1"}}
4.        <a href="http://{{$gimme->publication->site}}{{uri options="artic
      leAttachment"}}">PDF #{{$gimme->issue->name}}({{$gimme->issue->number}})
      / {{$gimme->issue->publish_date|camp_date_format:"%d.%m.%Y"}}</a><br />
5.      {{/list_article_attachments}}{{/list_articles}}
6.    </dd>
7.
8.    <dt class="ui-accordion-header ui-state-default">Latest Polls</dt>
9.    <dd class="ui-accordion-content">
10.     {{list_polls length="3" order="bynumber desc"}}
11.       {{if $gimme->current_list->index>1}}<br /><br />{{/if}}<p><b>{{$g
      imme->poll->question}}</b></p><br />
12.       {{list_poll_answers}}<p>
13.       <div>{{$gimme->pollanswer->answer}}
14.         <div class="poll-result ui-state-highlight" style="width:{{$gim
      me->pollanswer->percentage|string_format:"%d"}}%;">{{$gimme->pollanswer->
      percentage|string_format:"%d"}}%</div>
15.       </div></p>
16.       {{/list_poll_answers}}
17.     {{/list_polls}}
18.   </dd>
19.
20.  </dl>
21. {{/if}}
```



And that's pretty much it!

Definitely, you should think in advance about what type of archive you will need. After ten years of work, you would still like to have everything in good order.

TEMPLATE REFERENCE

55. LANGUAGE ELEMENTS AND CONVENTIONS
56. BASIC SYNTAX

55. LANGUAGE ELEMENTS AND CONVENTIONS

In the following chapters describing the template language the following conventions are used:

- identifiers in between '<' and '>' must be replaced in the template according to their description
- spaces must be used as in the language description
- identifiers that are not in between '<' and '>' are language keywords and must be written as in the language description
- identifiers enclosed by '[' and ']' characters are not mandatory in the statement
- sequences of identifiers separated by '|' character describe a situation where all the identifiers are valid but only one can be used at a time

The template language is composed of:

- statements: keywords with a special meaning that define actions taken by the template parser
- parameters: keywords describing statement features; they are used to specify statement constraints
- parameter values: keywords that describe the parameter that must be taken

Any value must be put in between double quotes ("). The double quote must be escaped any time it is used in an identifier. For example, escaping a value that contains a quote:

```
{{ if $gimme->article->name == "Lucas \"the beast\"" }}
```

In some statements (usually lists) there is a special parameter named "constraints". This parameter describes conditions that cannot be specified in the usual way:

```
parameter="value"
```

Instead, constraints are specified in the following way:

```
constraints="<constraints_list>"
```

Constraints are built from the following expressions:

- comparison expressions: <attribute> <operator> <value>
- attributes without type: <attribute>

Attributes may have no type, or one of the following types:

- integer: signed, 10 digits number
- string of characters: may contain any character except control characters - these will be removed automatically
- switch: has two values: "on" and "off"
- date: year, month, day; where the date value is specified it must be written in "yyyy-mm-dd" format
- time: hour, minute, second; where the time value is specified it must be written in "hh:mm:ss" format

- datetime: year, month, day, hour, minute, second; where the datetime value is specified it must be written in "yyyy-mm-dd hh:mm:ss" format
- topic: list of names defined by the Newscoop user for categorizing articles

Every type has a list of valid operators that can be used on attributes of that particular type. The operators list corresponding to defined types is:

- integer: <integer_operator> = is | not | greater | greater_equal | smaller | smaller_equal
- string of characters: <string_operator> = is | not | greater | greater_equal | smaller | smaller_equal
- switch: <switch_operator> = is | not
- date: <date_operator> = is | not | greater | greater_equal | smaller | smaller_equal
- time: <time_operator> = is | not | greater | greater_equal | smaller | smaller_equal
- datetime: <datetime_operator> = is | not | greater | greater_equal | smaller | smaller_equal
- topic: <topic_operator> = is | not

Spaces in values must be escaped with backslash, for example:

```
constraints="topic is Global\ Warming:en"
```

In this case, "Global Warming" is the topic name.

56. BASIC SYNTAX

The Newscoop template engine was built on Smarty, so the Newscoop template language is actually an extension of the Smarty template language. For details of the Smarty template language please read: http://smarty.net/manual/en/smarty.for.designers.php

All template tags are enclosed within delimiters. By default in Smarty these are the single curly brackets { and } but they can be changed. In Newscoop we use the double curly brackets {{ and }} for template tag delimiters.

For the examples in this Cookbook, we will assume that you are using the default Newscoop delimiters. All content outside of delimiters is displayed as static content, or unchanged. When the template engine encounters template tags, it attempts to interpret them, and displays the appropriate output in their place.

The following sub-chapters, which were copied from the Smarty manual, will familiarize you with the Smarty syntax.

ATTRIBUTES

Most of the functions take attributes that specify or modify their behaviour. Attributes to Smarty functions are much like HTML attributes. Static values don't have to be enclosed in quotes, but it is recommended for literal strings. Variables may also be used, and should not be in quotes.

Some attributes require boolean values (TRUE or FALSE). These can be specified as either unquoted true, on, and yes; or false, off, and no.

```
1.  {{include file='header.tpl'}}
2.  {{include file='header.tpl' attrib_name='attrib value'}}
3.  {{include file=$includeFile}}
4.  {{include file=#includeFile# title='Smarty is cool'}}
5.
6.   {{html_select_date display_days=yes}}
7.    {{mailto address='smarty@example.com'}}
8.     <select name='company_id'>
9.      {{html_options options=$companies selected=$company_id}}
10.    </select>
```

COMMENTS

Template comments are surrounded by asterisks, and then surrounded by the delimiter tags like this:

```
1.  {{* this is a comment *}}
2.  Comments are NOT displayed in the final output of the template, unlike <!
    -- HTML comments -->. These are useful for making internal notes in the t
    emplates which no one will see.
3.  {{* I am a template comment, I don't exist in the compiled output  *}}
4.  <html>
5.  <head>
6.  <title>{{$title}}</title>
7.  </head>
8.  <body>
9.  {{* another single line comment  *}}
10. <!-- HTML comment that is sent to the browser -->
```

```
11. {{* this multiline
12.    comment is
13.    not sent to browser
14. *}}
15. {{*********************************************************
16. Multi line comment block with credits block
17.   @ author:         bg@example.com
18.   @ maintainer:     support@example.com
19.   @ para:           var that sets block style
20.   @ css:            the style output
21. *********************************************************}}
22. {{* The header file with the main logo and stuff  *}}
23. {{include file='header.tpl'}}
24. {{* Dev note:  the $includeFile var is assigned in foo.php script  *}}
25. <!-- Displays main content block -->
26. {{include file=$includeFile}}
27. {{* this <select> block is redundant *}}
28. {{*
29. <select name="company">
30.   {{html_options options=$vals selected=$selected_id}}
31. </select>
32. *}}
33. <!-- Show header from affiliate is disabled -->
34. {{* $affiliate|upper *}}
35. {{* you cannot nest comments *}}
36. {{*
37. <select name="company">
38.   {{* <option value="0">-- none -- </option> *}}
39.   {{html_options options=$vals selected=$selected_id}}
40. </select>
41. *}}
42. {{* cvs tag for a template, below the 36 SHOULD be an american currency
43. . however its converted in cvs.. *}}
44. {{* &#36;Id: Exp &#36; *}}
45. {{* $Id: *}}
46. </body>
47. </html>
```

EMBEDDING VARS IN DOUBLE QUOTES

Smarty will recognize assigned variables embedded in "double quotes" so long as the variable name contains only numbers, letters, under_scores and square brackets []. See the PHP naming documentation at http://php.net/language.variables for more detail.

With any other characters, for example a .period or $object>reference, then the variable must be surrounded by `backticks`.

You cannot embed modifiers, they must always be applied outside of quotes.

```
1. {{func var="test $foo test"}}       <-- sees $foo
2. {{func var="test $foo_bar test"}}   <-- sees $foo_bar
3. {{func var="test $foo[0] test"}}    <-- sees $foo[0]
4. {{func var="test $foo[bar] test"}}  <-- sees $foo[bar]
5. {{func var="test $foo.bar test"}}   <-- sees $foo (not $foo.bar)
6. {{func var="test `$foo.bar` test"}} <-- sees $foo.bar
7. {{func var="test `$foo.bar` test"|escape}} <-- modifiers outside quotes!
```

Here are some practical examples:

```
1. {{* will replace $tpl_name with value *}}
2. {{include file="subdir/$tpl_name.tpl"}}
3.
4. {{* doesn't replace $tpl_name *}}
5. {{include file='subdir/$tpl_name.tpl'}}
6.
7. {{* must have backticks as it contains a . *}}
```

```
{{cycle values="one,two,`$smarty.config.myval`"}}

{{*  same as $module['contact'].'.tpl' in a php script *}}
{{include file="`$module.contact`.tpl"}}

{{*  same as $module[$view].'.tpl' in a php script *}}
{{include file="$module.$view.tpl"}}
```

ESCAPING SMARTY PARSING

It is sometimes desirable or even necessary to have Smarty ignore sections it would otherwise parse. A classic example is embedding Javascript or CSS code in a template. The problem arises as those languages use the { and } characters which are also the default delimiters for Smarty.

The simplest thing is to avoid the situation altogether by separating your Javascript and CSS code into their own files and then using standard HTML methods to access them.

Including literal content is possible using literal../literal blocks. Similar to HTML entity usage, you can use ldelim, rdelim or $smarty.ldelim to display the current delimiters.

FUNCTIONS

Every Smarty tag either prints a variable or invokes some sort of function. These are processed and displayed by enclosing the function and its attributes within delimiters like so: funcname attr1='val1' attr2='val2'.

```
{{config_load file='colors.conf'}}
{{include file='header.tpl'}}
{{insert file='banner_ads.tpl' title='Smarty is cool'}}
{{if $logged_in}}
    Welcome, <font >{{$name}}!</font>
{{else}}
    hi, {{$name}}
{{/if}}
{{include file='footer.tpl' ad=$random_id}}
```

- Both built-in functions and custom functions have the same syntax within templates.
- Built-in functions are the inner workings of Smarty, such as if, section and strip. There should be no need to change or modify them.
- Custom functions are additional functions implemented via plugins. They can be modified to your liking, or you can create new ones. html_options and popup are examples of custom functions.

MATH

Math can be applied directly to variable values.

```
{{$foo+1}}

{{$foo*$bar}}

{{* some more complicated examples *}}

{{$foo->bar-$bar[1]*$baz->foo->bar()-3*7}}

{{if ($foo+$bar.test%$baz*134232+10+$b+10)}}

{{$foo|truncate:"`$fooTruncCount/$barTruncFactor-1`"}}

```

```
13. {{assign var="foo" value="`$foo+$bar`"}}
```

VARIABLE MODIFIERS

Modifiers can be applied to variables, custom functions or strings. To apply a modifier, specify the value followed by a | (pipe) and the modifier name. A modifier may accept additional parameters that affect its behaviour. These parameters follow the modifier name and are separated by a : (colon). Also, all PHP functions can be used as modifiers implicitly (more below) and modifiers can be combined.

```
 1. {{* apply modifier to a variable *}}
 2. {{$title|upper}}
 3.
 4. {{* modifier with parameters *}}
 5. {{$title|truncate:40:'...'}}
 6.
 7. {{* apply modifier to a function parameter *}}
 8. {{html_table loop=$myvar|upper}}
 9.
10. {{* with parameters *}}
11. {{html_table loop=$myvar|truncate:40:'...'}}
12.
13. {{* apply modifier to literal string *}}
14. {{'foobar'|upper}}
15.
16. {{* using date_format to format the current date *}}
17. {{$smarty.now|date_format:"%Y/%m/%d"}}
18.
19. {{* apply modifier to a custom function *}}
20. {{mailto|upper address='smarty@example.com'}}
21.
22. {{* using  php's str_repeat *}}
23. {{'='|str_repeat:80}}
24.
25. {{* php's count *}}
26. {{$myArray|@count}}
27.
28. {{* php's shuffle on servers's ip *}}
29. {{$smarty.server.SERVER_ADDR|shuffle}}
30.
31. {{* this will uppercase and truncate the whole array *}}
32.  <select name="name_id">
33.   {{html_options output=$myArray|upper|truncate:20}}
34.  </select>
```

If you apply a modifier to an array variable instead of a single value variable, the modifier will be applied to every value in that array. If you really want the modifier to work on an entire array as a value, you must prepend the modifier name with an @ symbol.

For example:

```
{{$articleTitle|@count}}
```

will print out the number of elements in the $articleTitle array using the php count() function as a modifier.

Modifiers are autoloaded from the $plugins_dir or can be registered explicitly with the register_modifier() function. The later is useful for sharing a function between PHP scripts and Smarty templates.

All PHP functions can be used as modifiers implicitly, as demonstrated in the example above. However, using PHP functions as modifiers has two little pitfalls:

First - sometimes the order of the function-parameters is not the desirable one. Formatting $foo with {{"%2.f"|sprintf:$foo}} actually works, but asks for the more intuitive, like {{$foo|string_format:"%2.f"}} that is provided by the Smarty distribution.

Secondly - if $security is enabled, all PHP functions that are to be used as modifiers have to be declared trusted in the MODIFIER_FUNCS element of the $security_settings array

You can apply any number of modifiers to a variable. They will be applied in the order they are combined, from left to right. They must be separated with a | (pipe) character:

```
1. {{$articleTitle}}
2. {{$articleTitle|upper|spacify}}
3. {{$articleTitle|lower|spacify|truncate}}
4. {{$articleTitle|lower|truncate:30|spacify}}
5. {{$articleTitle|lower|spacify|truncate:30:". . ."}}
```

VARIABLES

Template variables start with the $dollar sign. They can contain numbers, letters and underscores, much like a PHP variable. You can reference arrays by index numerically or non-numerically. Also reference object properties and methods.

Config file variables are an exception to the $dollar syntax and are instead referenced with surrounding #hashmarks#, or via the $smarty.config variable.

```
1. {{$foo}}        <-- displaying a simple variable (non array/object)
2.
3. {{$foo[4]}}     <-- display the 5th element of a zero-indexed array
4.
5. {{$foo.bar}}    <-- display the "bar" key value of an array, similar to P
   HP $foo['bar']
6.
7. {{$foo.$bar}}   <-- display variable key value of an array, similar to PH
   P $foo[$bar]
8.
9. {{$foo->bar}}   <-- display the object property "bar"
10.
11. {{$foo->bar()}} <-- display the return value of object method "bar"
12.
13. {{#foo#}}       <-- display the config file variable "foo"
14.
15. {{$smarty.config.foo}} <-- synonym for {{#foo#}}
16.
17. {{$foo[bar]}}   <-- syntax only valid in a section loop, see {{section}}
18.
19. {{assign var=foo value='baa'}}{{$foo}} <--  displays "baa", see {{assign}
    }
```

Many other combinations are allowed:

```
1. {{$foo.bar.baz}}
2. {{$foo.$bar.$baz}}
3. {{$foo[4].baz}}
4. {{$foo[4].$baz}}
5. {{$foo.bar.baz[4]}}
6.
7. {{$foo->bar($baz,2,$bar)}} <-- passing parameters
8.
9. {{"foo"}}       <-- static values are allowed
10.
11. {{* display the server variable "SERVER_NAME" ($_SERVER['SERVER_NAME'])*}
    }
12. {{$smarty.server.SERVER_NAME}}
```

TEMPLATE REFERENCE - OBJECTS

57. INTRODUCTION TO NEWSCOOP OBJECTS
58. ARTICLE OBJECT AND ATTACHMENT, COMMENT, LOCATION
59. AUTHOR AND AUTHOR BIOGRAPHY
60. BROWSER
61. IMAGE
62. ISSUE
63. LANGUAGE
64. PUBLICATION
65. SECTION
66. SUBSCRIPTION
67. SUBTITLE (SUBHEADS IN LONG ARTICLES)
68. TEMPLATE
69. TOPIC
70. URL
71. USER
72. DEFAULT OBJECTS (LIST OF ALL AVAILABLE OBJECTS)

57. INTRODUCTION TO NEWSCOOP OBJECTS

The Newscoop template engine stores the template environment in an object named 'gimme'. All Newscoop properties and objects are attributes of the $gimme object.

The $gimme object has the following attributes which are not objects:

- version: the Newscoop version
- preview: true if the page was displayed in the preview window (by a Newscoop staff user)
- prev_list_empty: true if a list was displayed before querying this attribute and this list was empty

Any Newscoop object (e.g.: $gimme->language, or $gimme->article) has the following functions:

- has_property("property_name")

Returns true when the object has the given property. This function can be used for dynamic article properties too. For example, $gimme->article->has_property("name") will return true.

- same_as($gimme->other_object)

Returns true if the object variable given as a parameter points to the same data as the current object. For example, {{ if $gimme->article->same_as($gimme->default_article) }} will return true if the current article was the same as the article defined at the beginning of the template. If both were undefined this function returns true.

58. ARTICLE OBJECT AND ATTACHMENT, COMMENT, LOCATION

ARTICLE

The article object is set at the beginning of the main template based on the request URL. This object can be changed using the set_article function. The article object has the following properties:

Base properties/functions:

- name: article name
- number: article identifier in the Newscoop database
- author: returns an object corresponding to the first author in the list of authors, with the following properties:
    - name
    - first_name
    - last_name
    - email
    - defined
- authors: returns the complete list of authors for the current article - objects of type author (see above or "Author").
- keywords: text containing the article keywords separated by the Newscoop defined separator (default is comma [,])
- has_keyword (<keyword>): returns true if the given keyword existed in the article keywords list

Example:

```
{{ if $gimme->article->has_keyword (mp3podcast) }}<span class="audio"></span>{{
 /if }}
```

Important: the command matches parts of the keyword. In the above example keywords "mp3podcast10" and "mp3podcasttalk" both return true.

- type_name: article type name
- type_translated: article type name translated to the environment language
- <date_attribute>: article creation date field (year, month, day etc.)
- creation_date: article creation date; you can customize the date display format by using the filter camp_date_format
- publish_date: article publish date; you can customize the date display format by using the filter camp_date_format
- last_update: time of the last article edit
- template: the full path of the article template file
- type-> <article_type>-> <article_type_attribute>: returns the content of an article field specific to a certain article type (see the "Article Types" chapter in *Newscoop for Journalists and Editors*); read the chapter *Date and e-mail formatting* in this Cookbook for formatting the content of type date/time; see also the notes below
- url_name: the article name used in URL display (see the "Creating An Article" chapter in *Newscoop for Journalists and Editors*)

- translated_to(<language_code>): true if an article translation for the language with the given code existed
- subtitles_count(<field_name>): returns the number of the subtitles in the given content field; the article name is counted as the first subtitle so this function returns a number greater or equal to 1. If the specified field was not a content field it returns null.
- subtitle_url_id(<field_name>): returns the URL parameter which sets the number of the subtitle to be displayed from the field <field_name>
- current_subtitle_no(<field_name>): returns the number of the subtitle which will be displayed through the statement {{ $gimme->article-><field_name> }}; 0 for the default subtitle, 1 for the first subtitle defined in the article content field etc.
- publication: the publication to which this article belongs to - object of type publication (see the chapter "Publication")
- issue: the issue to which this article belongs to - object of type issue (see the chapter "Issue")
- section: the section to which this article belongs to - object of type section
- language: the article language - object of type language (see the chapter "Language")
- owner: the user who created the article - object of type user (see the chapter "User")
- defined: boolean value (true/false) - true if the article was set in the current environment; false otherwise

Comments related properties/functions:

- comments_enabled: true if comments were enabled for the article publication, article type and the current article
- comments_locked: true if comments were locked (meaning they can be listed but no new comments can be posted)
- comment_count: returns the number of the comments posted to the article and approved

Accessibility properties/functions:

- on_front_page: true if article front page flag was set
- on_section_page: true if article section page flag was set
- is_published: true if the article was published
- is_public: true if the article was accessible to the public
- is_indexed: true if the article was indexed by the search engine
- content_accessible: returns true if the content of the article is accessible for reading: either it is public or the reader subscribed to the publication (see also Editing An Article and Creating a Publication chapters in *Newscoop for Journalists and Editors*)

Attachment related properties/functions:

- has_attachments: true if the article had attached files
- image: returns the image object that was defined in the template environment; if the image was not defined in the environment, returns the first image of the article; if the article didn't have any image attached returns an unset image object
- image_index: returns the index of the current image inside the article; if the image wasn't defined in the environment or it doesn't belong to the article returns null.
- has_image(<image_index>): true if the article had an attached image and it's index was equal to the given index
- image(<image_index>): returns the attached image having the given index; if no such image existed, returns an unset image object. This is a function so the image index is given in between brackets. For example: $gimme->article->image(3)

- image<image_index>: returns the attached image having the given index; if no such image existed, returns an unset image object. This is a property, not a function. For example: $gimme->article->image3
- topics_count: returns the number of topics attached to this article
- has_topics: true if the article had attached topics
- has_topic(<topic_identifier>): true if the article had the specified topic attached to it

Statistics properties/functions:

- reads: returns the number of readers that viewed this article since it was published
- request_object_id: the identifier used in statistics gathering

<date_attribute> may be one of the following:

- year: year (four digits)
- mon: month as a number (1..12)
- mday: day of the month as a number (1..31)
- yday: day of the year (1..366)
- wday: day of the week as a number (0=Sunday..6=Saturday)
- hour: hour (0..23)
- min: minute (two digits)
- sec: seconds (two digits)
- mon_name: name of the month
- wday_name: day of the week

Note regarding the attribute type->[<article_type>->]<article_type_attribute>:

Attributes which are body fields (content) have the following properties:

- all_subtitles: returns the whole content of the body field, not just the current subtitle
- first_paragraph: returns the first paragraph of the current subtitle
- subtitles_count: returns the number of the subtitles in the body field
- subtitle_number: returns the number of the current subtitle: 0 for the default subtitle, 1 for the first subtitle defined in the article content field etc.
- subtitle_is_current: true if the subtitle that would be displayed through the statement {{ $gimme->article-><article_type_attribute> }} is the same as the subtitle defined in the template environment (see also the chapter "Subtitle (subheads in long articles)" and current_subtitle_no(<field_name>) above)
- has_previous_subtitles: true if the current subtitle from this field was not the first subtitle
- has_next_subtitles: true if the current subtitle from this field was not the last subtitle

Example: displaying the first paragraph of the dynamic field "content":

```
$gimme->article->content->first_paragraph
```

Note regarding the attribute type->[<article_type>->]<article_type_attribute>:

In Newscoop the table cell containing the image link in article body fields has the class *cs_img* and the cell containing the caption text has the class *caption*; this allows the web designer to change the layout of the article images by using a CSS file. The image link has the following structure:

```
1. <table border="0" cellspacing="0" cellpadding="0" class="cs_img" align=le
   ft>
2.   <tr>
3.     <td align="center">
4.       <img src="/get_img?NrArticle=143&NrImage=1" border="0"
5.           hspace="5" vspace="5">
```

```
6.      </td>
7.    </tr>
8.    <tr><td align="center" class="caption">Newscoop team</td></tr>
9.  </table>
```

Note regarding the attribute type->[<article_type>->]<article_type_attribute>:

The subtitle in the article body fields has the class *articlesubhead*; this allows the web designer to change the layout of the subtitles by using a CSS file. The subtitle anchor has the following structure:

```
1.  <span class="articlesubhead">
2.      <a name="a1.250_s1">Version 2.3.0 - 2.3.1</a>
3.  </span>
```

The structure of the anchor name is:

a<language_identifier>.<article_number>_s<subtitle_number>

ARTICLE ATTACHMENT

The article attachment object is usually initialized inside a list of article attachments. It is not initialized at the beginning of the template and can not be initialized by other Newscoop functions. The article attachment object has the following properties:

- identifier: the attachment identifier in the Newscoop database (integer value)
- file_name: the name of the attached document
- mime_type: the mime type of the attached document
- extension: the file extension of the attached document
- description: the user filled description field of the attached document in the current language
- size_b: the size of the attached document in bytes
- size_kb: the size of the attached document in kilobytes
- size_mb: the size of the attached document in megabytes
- defined: boolean value (true/false) - true if the attachment was set in the current environment; false otherwise

Examples

Taken from Template Pack "The Custodian" (date 2011-03-15) file "if-audio.tpl"

```
{{ if ($gimme->attachment->extension == mp3) || ($gimme->attachment->extension
== ogg) }}
```

Taken from Template Pack "The Custodian" (date 2011-03-15) file "if-audio.tpl"

```
<div class="audio-attachment-description">{{ $gimme->attachment->description }}
</div>
```

Taken from Template Pack "The Custodian" (date 2011-03-15) file "if-audio.tpl"

```
1.  <audio controls>
2.      <source src="http://{{ $gimme->publication->site }}{{ uri options="ar
    ticleattachment" }}" type="{{ $gimme->attachment->mime_type }}">
3.  </audio>
```

ARTICLE COMMENT

The comment object is usually initialized inside a list of article comments. It can be initialized at the beginning of the template from the URL request but can not be initialized by other Newscoop functions. The article comment object has the following properties:

- identifier: the numerical identifier of the article comment from the database
- real_name: the real name of the reader who posted the comment; the reader must be a registered Newscoop user; for anonymous readers this attribute will return an empty string
- nickname: the nickname of the reader who posted the comment
- reader_email: the email of the reader who posted the comment
- anonymous_author: whether the reader is anonymous or not
- submit_date: the date and time the comment was submitted
- subject: the subject of the article comment
- content: the content of the article comment
- content_real: the raw representation of the content
- level: the level of the current comment in the tree structure of the comments
- article: the article where this comment was submitted to - object of type article
- defined: true if the comment object had a valid value

Example

```
1.  {{ list_article_comments columns="2" order="bydate desc"}}
2.  {{ if $gimme->current_list->at_beginning }}
3.    <h2>{{ $gimme->article->comment_count }} Response(s) to &#8220;{{ $gimm
    e->article->name }}&#8221;</h2>
4.    <ol class="commentlist">
5.  {{ /if }}
6.     <li class="comment {{ if $gimme->current_list->column == "1" }}odd{{ e
    lse }}even{{ /if }}">
7.        <div class="comment-head cl">
8.           <div class="user-meta">
9.               <strong class="name">{{ $gimme->comment->nickname }}</strong
    > {{ $gimme->comment->submit_date|camp_date_format:"%e.%m.%Y at %H:%i" }}
10.           </div>
11.       </div>
12.       <div class="comment-entry">
13.           <p>{{ $gimme->comment->content }}</p>
14.       </div>
15.    </li>
16. {{ if $gimme->current_list->at_end }}
17.   </ol>
18. {{ /if }}
19. {{ /list_article_comments }}
```

ARTICLE LOCATION

Note: this works only in Newscoop 3.5.0 and newer versions.

Maps require jQuery: In order to use display maps you must include jQuery in the header of your document, with a link like this:

<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js">

The article "location" object is usually initialized inside a list of articles. It is not initialized at the beginning of the template and can not be initialized by other Newscoop functions. The article location object has the following properties:

- name: the location's name
- latitude: latitude in degrees
- longitude: longitude in degrees
- text: location's text description
- content: location's rich text description
- multimedia: list of multimedia items related to the location, see the "Location Multimedia" object.

Examples

```
{{ list_article_locations }}
    {{ if $gimme->location->enabled }}
        {{ $gimme->location->name }}{{ if $gimme->current_list->at_end }} {{ else }}, {{ /if }}
    {{ /if }}
{{ /list_article_locations }}
```

LISTING ALL INFORMATION FOR LOCATIONS IN AN ARTICLE

```
<h2>List article with geolocation information</h2>
<ul>
{{ list_articles}}
  {{ list_article_locations }}
    {{ if $gimme->location->enabled }}
      {{ if $gimme->current_list->at_beginning }}
        <ul>
          <li>LOCATION DATA FOR: {{ $gimme->article->name }}</li>
        <ul>
      {{ /if }}
            <li>list index: {{ $gimme->current_list->index }}</li>
            <li>name: {{ $gimme->location->name }}</li>
            <li>latitude: {{ $gimme->location->latitude }}</li>
            <li>longitude: {{ $gimme->location->longitude }}</li>
            <li>text: {{ $gimme->location->text }}</li>
            <li>content: {{ $gimme->location->content }}</li>
            <li>multimedia:</li>
              <ul>
                {{ foreach from=`$gimme->location->multimedia` item=multimediaitem }}
                  <li>src: {{ $multimediaitem->src }}</li>
                  <li>type: {{ $multimediaitem->type }}</li>
                  <li>spec: {{ $multimediaitem->spec }}</li>
                  <li>width: {{ $multimediaitem->width }}</li>
                  <li>height: {{ $multimediaitem->height }}</li>
                {{ /foreach }}
              </ul>
      {{ if $gimme->current_list->at_end }}
          </ul>
        </ul>
      {{ /if }}
    {{ /if }}
  {{ /list_article_locations }}
{{ /list_articles }}
</ul>
```

Ushahidi compatible KML format. You can use this as a feed to call as a layer in Ushahidi

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
<Document>
  <name>{{ $gimme->publication->name }}</name>
  <description><![CDATA[ .]]></description>
{{ list_articles}}
```

```
    {{ if $gimme->location->enabled }}
  <Style id="style{{ $gimme->article->number }}-{{ $gimme->current_list->index }}">
    <IconStyle>
      <Icon>
        <href>http://www.sourcefabric.org/javascript/geocoding/markers/marker-gold.png</href>
      </Icon>
    </IconStyle>
  </Style>
    {{ /if }}
  {{ /list_article_locations }}
{{ /list_articles }}
{{ list_articles}}
  {{ list_article_locations }}
    {{ if $gimme->location->enabled }}
  <Placemark>
    <name>{{ $gimme->location->name }} @ {{ $gimme->article->name }}</name>
    <description></description>
    <styleUrl>#style{{ $gimme->article->number }}-{{ $gimme->current_list->index }}</styleUrl>
    <Point>
      <coordinates>{{ $gimme->location->longitude }},{{ $gimme->location->latitude }},0.000000</coordinates>
    </Point>
  </Placemark>
    {{ /if }}
  {{ /list_article_locations }}
{{ /list_articles }}
</Document>
</kml>
```

59. AUTHOR AND AUTHOR BIOGRAPHY

AUTHOR

Note: this works only in Newscoop 3.5.0 and newer versions.

The article author object is usually initialized inside a list of article authors. It is also always possible to access this object for the current article in the environment (see "Article"). It is not initialized at the beginning of the template and can not be initialized by other Newscoop functions. The article author object has the following properties:

- first_name: the author's first name
- last_name: the author's last name
- name: the author's full name
- email: the author's e-mail address
- type: the author type
- biography: the author biography - see "Author Biography" object
- picture: the author picture - see "Image" object
- defined: boolean value (true/false) - true if the author was set in the current environment; false otherwise

Examples

```
1. {{ list_article_authors order="bylastname" }}
2.  <p class="author-head">
3.   {{ $gimme->author->type }}: {{ $gimme->author->name }}
4.  </p>
5.  <p class="author-bio">
6.   <img src="{{ $gimme->author->picture->imageurl }}" class="author-pic">
7.   {{ $gimme->author->biography->text }}
8.  </p>
9. {{ /list_article_authors }}
```

```
1. {{ if $gimme->article->defined }} by
2.  {{ $gimme->article->author->first_name }}
3.   {{ $gimme->article->author->last_name }}
4.    {{ $gimme->article->author->type }}
5. {{ /if }}
```

AUTHOR BIOGRAPHY

Note: this works only in Newscoop 3.5.0 and newer versions.

The author biography object is usually initialized within the "Author" object. It is not a main object as it can only be used through the "Author" object. Also, it is not initialized at the beginning of the template and cannot be initialized by other Newscoop functions. The author biography object has the following properties:

- first_name: the translated author's first name
- last_name: the translated author's last name
- name: the translated author's full name
- text: the translated author's biography
- defined: boolean value (true/false) - true if the author biography was set in the current environment; false otherwise

Example

```
1. {{ if $gimme->article->defined }} by
2.  {{ $gimme->article->author->biography->name }}
3.   ({{ $gimme->article->author->type }})<br />
4.    <p>{{ $gimme->article->author->biography->text }}</p>
5. {{ /if }}
```

60. BROWSER

(This chapter is based on the PHP browser detection script by Harald Hope, available under GNU GPL v3 from:
http://techpatterns.com/downloads/scripts/browser_detection_php_ar.txt)

The browser object is set at the beginning of the main template based on the request URL. The browser object can be called as follows:

```
{{ $gimme->browser }} or {{ $gimme->browser->browser_name }}
```

The above statement returns the full browser name string, if available, otherwise it returns ''. The statement:

```
{{ $gimme->browser->browser_number }}
```

returns the browser version number, if available, otherwise it returns ''. The statement:

```
{{ $gimme->browser->browser_working }}
```

returns the working shorthand browser name: ie, op, moz, konq, saf, ns4, webkit, and some others. If not shorthand, it will probably just return the full browser name, like lynx. The statement:

```
{{ $gimme->browser->dom }}
```

returns true/false if it is a basic dom browser, ie >= 5, opera >= 5, all new mozillas, safaris, or konquerors. The statement:

```
{{ $gimme->browser->ie_version }}
```

tests to see what general IE it is. Possible return values:

ie9x - all new msie 9 or greater - note that if in compat mode, 7,8,9 all show as 7
ie7x - all new msie 7 or greater
ie5x - msie 5 and 6, not mac
ieMac - msie 5.x mac release
ie4 - msie 4
old - pre msie 4

The statement:

```
{{ $gimme->browser->mobile_data }}
```

returns an array of data about mobiles. Note the browser/os number data is very unreliable so don't count on that. No Blackberry version handling is done explicitly. Make sure to test if this is an array, because if it's not mobile it will be null, not an array listed by array index number:

0 - $mobile_device
1 - $mobile_browser
2 - $mobile_browser_number
3 - $mobile_os
4 - $mobile_os_number
5 - $mobile_server
6 - $mobile_server_number
7 - $mobile_device_number

Note: $mobile_browser only returns if a specifically mobile browser is detected, like minimo. Same for mobile os, with the exception of GNU/Linux. Otherwise the standard script os/browser data is used. $mobile_server is a handheld service like docomo, novarro-vision, etc. Sometimes the string will contain no other usable data than this to determine if it's handheld or not.

```
1. {{ if $gimme->browser->ua_type != "mobile" }}
2.   {{ assign var="browserdetect_mobile_device" value="false" }}
3.   {{ assign var="browserdetect_mobile_os" value="false" }}
4.   {{ assign var="browserdetect_mobile_os_number" value="false" }}
5. {{ else }}
6.   {{ assign var="mobile_data" value=`$gimme->browser->mobile_data` }}
7.   {{ php }}
8.   $mobile_data = $this->get_template_vars('mobile_data');
9.   $this->assign('browserdetect_mobile_device', $mobile_data[0]);
10.  $this->assign('browserdetect_mobile_os', $mobile_data[3]);
11.  $this->assign('browserdetect_mobile_os_number', $mobile_data[4]);
12.  {{ /php }}
13. {{ /if }}
```

The statement:

```
{{ $gimme->browser->moz_data }}
```

returns array of mozilla / gecko information. Return Array listed by index number:

0 - $moz_type [moz version - the specific brand name that is, eg: firefox)
1 - $moz_number - the full version number of $moz_type (eg: for firefox: 3.6+2b)
2 - $moz_rv - the Mozilla rv version number, math comparison version. This tells you what gecko engine is running in the browser (eg rv: 1.8)
3 - $moz_rv_full - rv number (for full rv, including alpha and beta versions: 1.8.1-b3)
4 - $moz_release_date - release date of the browser

```
1. {{ if $gimme->browser->browser_working == "moz" }}
2.   {{ assign var="browser_data" value=`$gimme->browser->moz_data` }}
3.   {{ php }}
4.   $browser_data = $this->get_template_vars('browser_data');
5.   $this->assign('browserdetect_name', $browser_data[0]);
6.   $this->assign('browserdetect_engine', "gecko");
7.   $this->assign('browserdetect_engineversion', $browser_data[2]);
8.   $this->assign('browserdetect_version', $browser_data[1]);
9.   {{ /php }}
10. {{ /if }}
```

The statement:

```
{{ $gimme->browser->os }}
```

returns which os is being used - win, nt, mac, OR iphone, blackberry, palmos, palmsource, symbian, beos, os2, amiga, webtv, linux, unix. The statement:

```
{{ $gimme->browser->os_number }}
```

returns windows versions, 95, 98, ce, me, nt: 4; 5 [windows 2000]; 5.1 [windows xp]; 5.2 [Server 2003]; 6.0 [Windows Vista], 6.1 [Windows 7]. Only win, nt, mac, iphone return os numbers (mac/iphone return 10 if OS X.) OR returns GNU/Linux distro/unix release name, otherwise returns null.

```
{{ $gimme->browser->run_time }}
```

The time it takes this script to execute from start to point of returning value. Requires PHP 5 or greater. Returns time in seconds to 8 decimal places: 0.00245687. Run time does not count the time used by PHP to include/parse the file initially. That total time is about 5-10x longer. Because subsequent script run-throughs go very fast, you will see the seconds go from something like 0.00115204 for the first time, to something like 0.00004005 for second and more runs. The statement:

```
{{ $gimme->browser->safe }}
```

returns true/false, you can determine what makes the browser be safe lower down, currently it's set for ns4 and pre version 1 mozillas not being safe, plus all older browsers. The statement:

```
{{ $gimme->browser->true_ie_number }}
```

returns the true version of msie running, ignoring the compat mode version.

Note that php will turn 7.0 to 8 when adding 1, so keep that in mind in your tests. 7.1 will become 8.1 as expected, however. This test currently only tests for 7.x -> 8.x

FYI: in PHP, 7.0 == 7 is true but 7.0 === 7 is NOT true. If this is null but set, then it is NOT running in compatibility mode. The statement:

```
{{ $gimme->browser->ua_type }}
```

returns one of the following:

bot (web bot)
bro (normal browser)
bbro (simple browser)
mobile (handheld)
dow (downloading agent)
lib (http library)

The statement:

```
{{ $gimme->browser->webkit_data }}
```

returns array of webkit data. The Return Array is listed by index number:

0 - $webkit_type [webkit version name (Eg. chrome)]
1 - $webkit_type_number [webkit version number (Eg. Chrome's: 1.2)]
2 - $browser_number [the actual webkit version number (Eg. Webkit's: 436)]

```
1. {{ if $gimme->browser->browser_working == "webkit" }}
2.   {{ assign var="browser_data" value=`$gimme->browser->webkit_data` }}
3.   {{ assign var="browserdetect_engineversion" value=`$gimme->browser->bro
   wser_number` }}
4.   {{ php }}
5.   $browser_data = $this->get_template_vars('browser_data');
6.   $this->assign('browserdetect_name', $browser_data[0]);
7.   $this->assign('browserdetect_engine', "webkit");
8.   $this->assign('browserdetect_version', $browser_data[1]);
9.   {{ /php }}
```

```
  10. {{ /if }}
```

61. IMAGE

The image object is usually initialized inside a list of article images or a list of images. It is not initialized at the beginning of the template and can not be initialized by another Newscoop function. The image object has the following properties:

- number: the image identifier in the images archive
- photographer: the name of the photographer that took the picture
- place: a short text containing the place where the picture was taken
- description: short description of the picture
- caption: displays the image caption field
- <date_attribute>: image creation date field (year, month, day etc.)
- date: image creation date; you can customize the date display format by using the filter camp_date_format (see the chapter "Date and e-mail formatting")
- last_update: displays the time the image was updated
- article_index: returns the index of the current image inside the article defined in the environment; if the image wasn't defined in the environment, the article was not defined or the image didn't belong to the article, returns null
- imageurl: returns the URL of the current image
- thumbnailurl: returns the URL of the current image's thumbnail
- type: the subtype from the mimetype field; e.g.: png, jpeg, gif etc.
- is_local: 1 if the image was stored locally, 0 if external
- defined: boolean value (true/false) - true if the image was set in the current environment; false otherwise

<date_attribute> may be one of the following:

- year: year as a number (four digits)
- mon: month as a number (1..12)
- mday: day of the month as a number (1..31)
- yday: day of the year (1..366)
- wday: day of the week as a number (0=Sunday..6=Saturday)
- hour: hour (0..23)
- min: minute (two digits)
- sec: seconds (two digits)
- mon_name: name of the month
- wday_name: day of the week

62. ISSUE

The issue object is set at the beginning of the main template, based on the request URL. This object can be changed using the set_issue function. The issue object has the following properties:

- name: issue name
- number: issue identifier in the Newscoop database (integer value)
- <date_attribute>: issue publish date field (year, month, day etc.)
- date: the issue publish date; you can customize the date display format by using the filter camp_date_format (see the chapter "Date and e-mail formatting")
- publish_date: alias of date
- template: the full path of the issue template file
- publication: the publication to which this issue belongs (see the chapter "Publication")
- language: the issue language (see the chapter "Language")
- url_name: the issue name used in URL display (see the chapter "Creating An Issue" in *Newscoop for Journalists and Editors*)
- defined: boolean value (true/false) - true if the issue was set in the current environment; false otherwise
- is_current: true if the issue set in the environment was the latest published issue
- is_published: true if the issue has been and is currently published; false otherwise

<date_attribute> may be one of the following:

- year: year as a number (four digits)
- mon: month as a number (1..12)
- mday: day of the month as a number (1..31)
- yday: day of the year (1..366)
- wday: day of the week as a number (0=Sunday..6=Saturday)
- hour: hour (0..23)
- min: minute (two digits)
- sec: seconds (two digits)
- mon_name: name of the month
- wday_name: day of the week

63. LANGUAGE

The language object is set at the beginning of the main template, based on the request URL. This object can be changed using the set_language function. The language object has the following properties:

- name: language name
- number: language identifier in the Newscoop database (integer value)
- english_name: language name in English
- code: language international code
- defined: boolean value (true/false) - true if the language was set in the current environment; false otherwise

If you want to verify whether the current environment language is the same as the initial language, compare it to the default_language object. For example:

```
1. {{ if $gimme->language == $gimme->default_language }}
2. ...
3. {{ /if }}
```

If you want to display parts of your templates in a different language according to the language selected by the user, you can use something similar to the following code. This snippet was used for a bilingual publication.

```
{{ if $gimme->language->code == "en" }}
 archive
  {{ else }}
   [trans]archive
{{ /if }}
```

64. PUBLICATION

The publication object is set at the beginning of the main template, based on the request URL. This object can be changed using the set_publication function. The publication object has the following properties:

- name: publication name
- identifier: publication identifier in the Newscoop database (integer value)
- default_language: the publication default language - object of type Language
- site: publication site
- defined: boolean value (true/false) - true if the publication was set in the current environment; false otherwise
- public_comments: true if the public (anonymous readers) are allowed to post comments
- moderated_comments: true if the comments posted by the current reader will be moderated
- captcha_enabled: true if CAPTCHA will be used to detect spam
- subscription_currency: returns the currency used for subscription payments
- subscription_time_unit: returns the time unit (day, week, month, year) used to set the subscription length
- subscription_trial_time: returns the default time of the trial subscription in time units
- subscription_paid_time: returns the default time of the paid subscription in time units
- subscription_time: returns the default time of the subscription in time units; the subscription type should be defined through request parameters in the user form (see the chapter "Subscription form")
- subscription_unit_cost: returns the cost of the time unit for the paid subscription, for a single translation of the publication
- subscription_unit_cost_all_lang: returns the cost of the time unit for the paid subscription for all translations of the publication

For more details on these attributes see also the chapter "Creating a Publication" in *Newscoop for Journalists and Editors*.

65. SECTION

The *section* object is set at the beginning of the main template, based on the request URL. This object can be changed using the set_section function. The section object has the following properties:

- name: section name
- number: section identifier in the Newscoop database
- description: section description text
- url_name: the section name used in URL display (see the *"Creating a section"* chapter in *Newscoop for Journalists and Editors*)
- template: the full path of the issue template file
- publication: the publication to which this section belongs to - object of type Publication
- issue: the issue to which this section belongs to - object of type Issue
- language: the section language - object of type Language
- defined: boolean value (true/false) - true if the section was set in the current environment; false otherwise

66. SUBSCRIPTION

The user subscription object is set at the beginning of the main template based on the session cookies, or if the Login action took place. It can not be initialized by other Newscoop functions. The subscription object has the following properties:

- identifier: the subscription identifier in the Newscoop database
- currency: the currency identifier
- type: one of the following values: "trial", "paid"
- start_date: returns the start date of the subscription
- expiration_date: the expiration date in the format "yyyy-mm-dd hh:mm:ss"
- is_active: true if the subscription was active
- is_valid: true if the subscription was active and did not expire
- publication: returns the publication to which the subscription was made
- has_section(<section_number>): returns true if the subscription included the given section
- defined: true if the subscription object had a valid value

67. SUBTITLE (SUBHEADS IN LONG ARTICLES)

The subtitle object is usually initialized inside a list of subtitles. It can be initialized at the beginning of the template from the URL request but can not be initialized by other Newscoop functions. The subtitle object has the following properties:

- number: the order number of the subtitle (starts from 0)
- name: subtitle name without the HTML formatting
- field_name: the article field name to which the subtitle belongs
- formatted_name: the subtitle name with HTML formatting
- content: the subtitle content
- count: the number of subtitles in the field content
- has_previous_subtitles: true if previous subtitles exist
- has_next_subtitles: true if subtitles exist after the current subtitle

68. TEMPLATE

The template object is set at the beginning of the main template based on the request URL. This object cannot be changed using Newscoop functions. The template object has the following properties:

- name: the template file name
- identifier: the template identifier in the Newscoop database
- type: returns one of the following values: issue, section, article, default, nontpl
- defined: true if the template object had a valid value

69. TOPIC

The topic object is usually initialized inside a list of article topics or a list of subtopics. It can be initialized at the beginning of the template from the URL request, or by using the set_topic function in Newscoop. The topic object has the following properties:

- name: returns the topic name in the current language defined in the template environment
- value: returns the topic value in the following format: <topic_name_lang_code> = <topic_name>:<language_code>
- identifier: the topic identifier in the Newscoop database
- defined: true if the topic object had a valid value

70. URL

The URL object follows the changes in the template environment, meaning that every time an object in the environment changes the URL object is updated. It has the following properties:

- is_valid: returns true if the URL was valid, false otherwise. On invalid URLs Newscoop returns a "404 not found" HTTP response. This option only works in Newscoop 3.4.0 and newer versions. Here is example code which could be used in a 404 page:

```
1. {{ if !$gimme->url->is_valid }}
2.  <h3>The requested page was not found.</h3>
3. {{ set_language name=`$gimme->publication->default_language->english_name
   ` }}
4.   {{ set_current_issue }}
5. {{ else }}
6.     <!-- display content -->
7. {{ /if }}
```

- uri: returns the complete link URI, and is equivalent to:

```
{{ $gimme->url->uri_path }}?{{ $gimme->url->url_parameters }}
```

- uri_path: returns only the path part of the URI, the part before the parameters list. For example, if /en/1/2/3?param1=text was the full URI, uri_path is /en/1/2/3
- url: returns the complete URL in the form:

```
http://<default_site_alias><uri>
```

- url_parameters: returns a string containing the runtime environment parameters in URL format
- form_parameters: the runtime environment parameters in HTML form format:

```
<input type="hidden" name="<param_name>" value="<param_value>">
```

- base: returns the URL base in the form:

```
http[s]://<server_name>[:<port>]
```

The port is not displayed if it's value was the default value (80 for HTTP, 443 for HTTPS)

- path: equivalent to uri_path
- query: equivalent to url_parameters
- type: returns the identifier of the URL type set in the publication (see also "Creating A Publication" in *Newscoop for Journalists and Editors*)
- request_uri: equivalent to uri
- scheme: one of the following values: http, https
- host: the host name from the URL
- port: the port to which the request was made
- language: returns the language object corresponding to the language set in the URL; this value is always the same as the language in the environment (see the chapter "Language" for more details)

- publication: returns an object corresponding to the publication identified by the <server_name>; this value is always the same as the publication in the environment (see the chapter "Publication" for more details)
- issue: returns an object corresponding to the issue specified in the URL (unset if the issue was not specified); this value is always the same as the issue in the environment (see the chapter "Issue" for more details)
- section: returns an object corresponding to the section specified in the URL (unset if the section was not specified); this value is always the same as the section in the environment (see the chapter "Section" for more details)
- article: returns an object corresponding to the article specified in the URL (unset if the article was not specified); this value is always the same as the article in the environment (see the chapter "Article object and attachment, comment, location" for more details)

The URL object has the following functions:

- get_parameter(<parameter_name>): returns the value of the given parameter, null if not set
- set_parameter(<parameter_name>, <parameter_value>): set the given parameter to the given value
- reset_parameter(<parameter_name>): unset the given parameter

71. USER

The user object is set at the beginning of the main template based on the session cookies, or if the Login action or Edit user action took place. It can not be initialized by other Newscoop functions. The user object has the following properties:

- identifier: the user identifier in the Newscoop database
- name: the user's full name
- uname: the user's login name
- gender: "M" or "F"
- email
- city
- str_address: street address
- state
- phone
- fax
- country: the country name
- country_code: the country code
- contact
- second_phone
- postal_code
- employer
- position
- interests
- how
- languages
- improvements
- field1
- field2
- field3
- field4
- field5
- text1
- text2
- text3
- pref1
- pref2
- pref3
- pref4
- title
- age
- defined: true if the user object had a valid value
- logged_in: true if the user was defined and authenticated
- blocked_from_comments: true if the user was blocked from posting comments
- subscription: returns the first subscription assigned to this user; unset if the user is a staff member
- is_admin: true if the user is a staff member, and can therefore log in to the Newscoop administration interface
- has_permission(<permission_name>): true if the user is a staff member and has been given a specific permission in the Newscoop administration interface

72. DEFAULT OBJECTS (LIST OF ALL AVAILABLE OBJECTS)

The default objects contain the information of a specific object at the beginning of the main template.

DEFAULT ARTICLE

The default_article object is set at the beginning of the main template based on the request URL and cannot be modified. It has the same attributes as the Article object.

DEFAULT ISSUE

The default_issue object is set at the beginning of the main template based on the request URL and cannot be modified. It has the same attributes as the Issue object.

DEFAULT LANGUAGE

The default_language object is set at the beginning of the main template based on the request URL and cannot be modified. It has the same attributes as the Language object.

DEFAULT PUBLICATION

The default_publication object is set at the beginning of the main template based on the request URL and cannot be modified. It has the same attributes as the Publication object.

DEFAULT SECTION

The default_section object is set at the beginning of the main template based on the request URL and cannot be modified. It has the same attributes as the Section object.

DEFAULT TEMPLATE

The default_template object is set at the beginning of the main template based on the request URL and cannot be modified. It has the same attributes as the Template object.

DEFAULT TOPIC

The default_topic object is set at the beginning of the main template based on the request URL and cannot be modified. It has the same attributes as the Topic object.

DEFAULT URL

The default_url object is set at the beginning of the main template based on the request URL and cannot be modified. It has the same attributes as the URL object.

TEMPLATE REFERENCE - CONTROLLING OBJECTS

73. SET OBJECTS
74. UNSET OBJECTS
75. LOCAL - TEMPORARY VARIABLE ENVIRONMENT

73. SET OBJECTS

SET ARTICLE

Purpose:

Sets the runtime environment article to the one selected by the statement constraint. If the statement constraint was not valid, the section is not changed.

Syntax:

```
{{ set_article name="<article_name>" }}
```

Select the article having the specified name. If the supplied name was not valid, this parameter is not modified.

```
{{ set_article number="<article_number>" }}
```

Select the article having the specified number. If the supplied number was not valid, this parameter is not modified.

Constraints:

Cannot be used inside "list_articles", "list_article_attachments", "list_article_comments", "list_article_topics", "list_article_audio_attachments", "list_search_results" and "list_subtitles" statements.

See also "Set Default Article", and "Unset Article".

SET CURRENT ISSUE

Purpose:

Sets the runtime environment issue to the last published issue.

Syntax:

```
{{ set_current_issue }}
```

Constraints:

Cannot be used inside any list statement.

See also "Set Issue", "Set Default Issue", and "Unset Issue".

SET DEFAULT ARTICLE

Purpose:

Sets the runtime environment article to the "default_article".

Syntax:

```
{{ set_default_article }}
```

Constraints:

Cannot be used inside "list_articles", "list_article_attachments", "list_article_comments", "list_article_topics", "list_article_audio_attachments", "list_search_results" and "list_subtitles" statements.

See also "Set Article", and "Unset Article".

SET DEFAULT ISSUE

Purpose:

Sets the runtime environment issue to the "default_issue".

Syntax:

```
{{ set_default_issue }}
```

Constraints:

Cannot be used inside any list statement.

See also "Set Issue", "Set Current Issue", and "Unset Issue".

SET DEFAULT LANGUAGE

Purpose:

Sets the runtime environment language to the "default_language".

Syntax:

```
{{ set_default_language }}
```

Constraints:

Cannot be used inside any list statement.

See also "Set Language", and "Unset Language".

SET DEFAULT PUBLICATION

Purpose:

Sets the runtime environment publication to the "default_publication".

Syntax:

```
{{ set_default_publication }}
```

Constraints:

Cannot be used inside any list statement.

See also "Set Publication", and "Unset Publication".

SET DEFAULT SECTION

Purpose:

Sets the runtime environment section to the "default_section".

Syntax:

```
{{ set_default_section }}
```

Constraints:

Cannot be used inside "list_sections", "list_articles", "list_article_attachments", "list_article_comments", "list_article_topics", "list_article_audio_attachments", "list_search_results" and "list_subtitles" statements.

See also "Set Section", and "Unset Section".

SET DEFAULT TOPIC

Purpose:

Sets the runtime environment topic to the "default_topic".

Syntax:

```
{{ set_default_topic }}
```

Constraints:

Cannot be used inside "list_articles" and "list_article_topics" statements.

See also "Set Topic", and "Unset Topic".

SET ISSUE

Purpose:

Sets the runtime environment issue to the one selected by the statement constraint. If the statement constraint was not valid, the issue is not changed.

Syntax:

```
{{ set_issue number="<issue_number>" }}
```

Select the issue having the specified number. If the number supplied was not valid, this parameter is not changed.

Constraints:

Cannot be used inside any list statement.

See also "Set Default Issue", "Set Current Issue", and "Unset Issue".

SET LANGUAGE

Purpose:

Sets the runtime environment language to the one selected by its English name. From this statement on, the language set is the newly chosen one. If the language name supplied was not valid, this variable is not modified.

Syntax:

```
{{ set_language name="<language_name>" }}
```

Filters:

<language_name> is the English name of the selected language.

Constraints:

Can not be used inside any list statement.

See also "Set Default Language", and "Unset Language".

SET PUBLICATION

Purpose:

Sets the runtime environment publication to the one selected by the statement constraint. If the statement constraint was not valid, the publication is not changed.

Syntax:

```
{{ set_publication name="<publication_name>" }}
```

Select the publication having the specified name. If the name supplied was not valid, this parameter is not modified.

```
{{ set_publication identifier="<publication_identifier>" }}
```

Select the publication having the specified identifier. The publication identifier is a unique number associated to the publication and is supplied by the Newscoop administration interface.

Constraints:

Cannot be used inside any list statement.

See also "Set Default Publication", and "Unset Publication".

SET SECTION

Purpose:

Sets the runtime environment section to the one selected by the statement constraint. If the statement constraint was not valid, the section is not changed.

Syntax:

```
{{ set_section name="<section_name>" }}
```

Select the section having the specified name; this has to be written in the language of the context. If the name supplied was not valid, this parameter is not modified.

```
{{ set_section number="<section_number>" }}
```

Select the section having the specified number; this is not dependent on context language. If the number supplied was not valid, this parameter is not modified.

Constraints:

Cannot be used inside "list_sections", "list_articles", "list_article_attachments", "list_article_comments", "list_article_topics", "list_article_audio_attachments", "list_search_results" and "list_subtitles" statements.

See also "Set Default Section", and "Unset Section".

SET TOPIC

Purpose:

Sets the runtime environment topic to the one selected by the statement constraint. If the statement constraint was not valid, the topic is not changed. Setting the environment topic will change the behaviour of an article list: only articles having that topic will be listed. For example:

```
1. {{ set_topic name="test:en" }}
2.  {{ list_articles }}
3.  ...
4.  {{ /list_articles }}
```

The example above will list only articles having the topic 'test'. The topic is automatically appended to the URL parameters, so you don't have to set the topic in the current page:

```
1. {{ set_topic name"test:en" }}
2. <a href="{{ uri }}">text</a>
```

Syntax:

```
{{ set_topic name="<topic_name_lang_code>" }}
```

```
{{ set_topic identifier="<integer_value>" }}
```

Select the topic having the specified name:language pair or identifier. If the supplied value was not valid, this variable is not modified. The name:language pair must be written in the following format:

```
<topic_name_lang_code> = <topic_name>:<language_code>
```

Examples:

sport:en, music:en where 'sport' or 'music' is the topic name and 'en' is the English language code.

Constraints:

Cannot be used inside "list_articles" and "list_article_topics" statements.

See also "Set Default Topic", and "Unset Topic".

74. UNSET OBJECTS

UNSET ARTICLE

Purpose:

Unset the runtime environment article. After this statement the article object will not be defined any more.

Syntax:

```
{{ unset_article }}
```

Constraints:

Cannot be used inside "list_articles", "list_article_attachments", "list_article_comments", "list_article_topics", "list_article_audio_attachments", "list_search_results" and "list_subtitles" statements.

See also "Set Default Article", and "Set Article".

UNSET COMMENT

Purpose:

Unset the runtime environment comment. After this statement the comment object will not be defined any more.

Syntax:

```
{{ unset_comment }}
```

Constraints:

Cannot be used inside "list_article_comments" statements.

UNSET ISSUE

Purpose:

Unset the runtime environment issue. After this statement the issue object will not be defined any more.

Syntax:

```
{{ unset_issue }}
```

Constraints:

Cannot be used inside any list statements.

See also "Set Issue", "Set Current Issue", and "Set Default Issue".

UNSET LANGUAGE

Purpose:

Unset the runtime environment language. After this statement the language object will not be defined any more.

Syntax:

```
{{ unset_language }}
```

Constraints:

Cannot be used inside "list_articles", "list_article_attachments", "list_article_comments", "list_article_topics", "list_article_audio_attachments", "list_search_results" and "list_subtitles" statements.

See also "Set Language", and "Set Default Language".

UNSET PUBLICATION

Purpose:

Unset the runtime environment publication. After this statement the publication object will not be defined any more.

Syntax:

```
{{ unset_publication }}
```

Constraints:

Cannot be used inside any list statement.

See also "Set Publication", and "Set Default Publication".

UNSET SECTION

Purpose:

Unset the runtime environment section. After this statement the section object will not be defined any more.

Syntax:

```
{{ unset_section }}
```

Constraints:

Cannot be used inside "list_sections", "list_articles", "list_article_attachments", "list_article_comments", "list_article_topics", "list_article_audio_attachments", "list_search_results" and "list_subtitles" statements.

See also "Set Section", and "Set Default Section".

UNSET TOPIC

Purpose:

Unset the runtime environment topic. After this statement the topic object will not be defined any more.

Syntax:

```
{{ unset_topic }}
```

Constraints:

Cannot be used inside "list_articles" and "list_article_topics" statements.

See also "Set Default Topic", and "Set Topic".

75. LOCAL - TEMPORARY VARIABLE ENVIRONMENT

LOCAL

Purpose:

Creates a temporary environment; when leaving the local block the previous template environment (before entering local) is restored.

Syntax:

```
1. {{ local }}
2.  <list_of_instructions>
3. {{ /local }}
```

The list of instructions may contain any instruction allowed in the current context. If it is used inside a list it must respect the constraints of the list.

Constraints:

None.

Note: {{ local }} creates a new object. If you use {{ local }} excessively you might run into performance issues because it is resource-hungry.

TEMPLATE REFERENCE - LISTS

76. LIST ARTICLES AND ARTICLE RELATED CONTENT
77. LIST ARTICLE TOPICS AND SUBTOPICS
78. LIST OF ISSUES AND SECTIONS
79. LIST SUBTITLES (PAGINATION OF LONG ARTICLES)
80. LIST IMAGES
81. LIST LANGUAGES
82. LIST SEARCH RESULTS

76. LIST ARTICLES AND ARTICLE RELATED CONTENT

LIST OF ARTICLES

Purpose:

Select the list of articles according to the given constraints and current environmental variables. The publication, language, issue, section and article variables may not be defined outside the list_articles statement; inside the statement however, all these variables are defined. The code between "{{ list_articles }}" statement "{{ /list_articles }}" is repeated for every article in the list.

Syntax:

```
{{ list_articles [length="<integer_value>"]
        [columns="<integer_value>"]
        [constraints="<list_of_article_constraints>"]
        [ignore_publication="true"]
        [ignore_issue="true"]
        [ignore_section="true"]
        [ignore_language="true"]
        [location="<longitude, latitude>]
        [order="<order_condition>"] }}
    <list_of_instructions>
{{ /list_articles }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current list

```
{{ list_articles length="10" columns="2" order="byPublishDate desc"}}
{{ if $gimme->current_list->column == "1" }}
left column
{{ /if }}
{{ /list_articles }}
```

- ignore_publication: list articles from all publications, not only from the environment publication
- ignore_issue: list articles from all issues, not only from the environment issue; if the section was defined it will list only articles from the environment section
- ignore_section: list articles from all sections, not only from the environment section
- ignore_language: list articles in all languages; if the issue and section were defined in the environment it lists only articles belonging to the environment issue/section
- location: longitude and latitude values for the location

<list_of_instructions> may contain any statement except those listed at the end of the page.

- <list_of_article_constraints>=
  - <article_constraint> <list_of_article_constraints>
  - <article_constraint>
- <article_constraint>=
- name <string_operator> <string_value>
- number <integer_operator> <integer_value>
- keyword <string_value>
- OnFrontPage <switch_operator> <switch_value>
- OnSection [<switch_operator> <switch_value>]
- upload_date <date_operator> <date_value>
- publish_date <date_operator> <date_value>
- public <switch_operator> <switch_value>
- type <switch_operator> <article_type>
- <article_type> <article_type_attribute> <attribute_type_operator> <attribute_type_value>
- matchAllTopics
- matchAnyTopic
- topic is|not <string_value>
- reads <integer_operator> <integer_value>
- author <string_operator> <string_value>
- issue <integer_operator> <integer_value>
- section <integer_operator> <integer_value>

Examples:

Taken from Template Pack "The Custodian" (date 2011-03-16) file "article.tpl"

```
1. {{ list_articles name="topic_articles"
2. constraints="number not `$gimme->article->number`
3. `$topic_cond` matchAnyTopic"  ignore_issue=true length=3 }}
4.   {{ include file="classic/tpl/teaserframe_articlelistright.tpl"
5. }}
6.   {{ include file="classic/tpl/pagination.tpl" }}
7.   {{ assign var="number_cond" value="`$number_cond`
8. number not `$gimme->article->number` " }}
9. {{ /list_articles }}
```

Examples of constraints:

constraints="issue greater 10 issue smaller 20"

constraints="section greater_equal 40 section smaller 60"

- name, number, upload_date, publish_date are self-explanatory article attributes
- keyword: all articles containing the specified keyword (and respecting all the other constraints) will be in the list
- OnFrontPage: articles having "Show article on front page" flag in <switch_operator> relation with <switch_value> will be selected; for details see Creating articles within a section
- OnSection: articles having "Show article on section page" flag in <switch_operator> relation with <switch_value> will be selected; for details see Creating articles within a section
- public: articles having "Allow users without subscription..." flag in <switch_operator> relation with <switch_value> will be selected

- ... <article_type_attribute> ...: articles being of <article_type> and having <article_type_attribute> in <attribute_type_operator> relation with <attribute_type_value> will be selected; for details see "Article Types"
- matchAllTopics/matchAnyTopic: defines the behaviour of the list when matching articles topics against the list of topics given as parameters:
    - matchAllTopics will force the selection of articles that have all the given topics
    - matchAnyTopic (default) will select articles that have at least one topic from the given topic list
- topic: if "is" operator is used, articles having specified topic in their list of topics will be selected; if "not" operator is uses articles not having specified topic in their list of topics will be selected
- reads: you can set constraints based on the number of readers that viewed this article since it was published
- author: you can list articles that have or don't have a certain author; e.g.: "author is John\ Doe" (the backslash is needed to escape the space character - see Language Elements; Conventions at the bottom of the page). The author attribute has a predefined value "__current"; when using this value the author will be filled in from the currently defined article. For example: "author is __current"

Note on author attribute: this works only in Newscoop 3.2.1 and newer versions.

Note on topics: the topic name must be written in the following format: <topic_name>:<language_code>

Examples of valid topic names:

sports:en

health:en

Note on topics: If a certain topic was defined in the template environment by use of the "set_topic" statement or URL parameter the list will change the behaviour of the articles list. Only articles having that topic will be listed. For example:

```
1. {{ set_topic name="test:en" }}
2. {{ list_articles }}
3. ...
4. {{ /list_articles }}
```

This will list only articles having the topic 'test'. The topic is automatically appended to the URL parameters so you don't have to set the topic in the current page.

- <order_condition>=
    - byNumber desc|asc
    - byName desc|asc
    - byDate desc|asc
    - byCreationDate desc|asc
    - byPublishDate desc|asc
    - byLastUpdate desc|asc
    - byPopularity desc|asc
    - byPublication desc|asc
    - byIssue desc|asc
    - bySection desc|asc
    - byLanguage desc|asc
    - bySectionOrder desc|asc
    - byComments desc|asc
    - byLastComment desc|asc

- byComments instructs the Newscoop to list articles by the number of comments filed to each article
- byLastComment will list articles ordered by the last article comment time
- byDate is an alias of byCreationDate
- The other order conditions are self-explanatory
- The default order of articles in the list (if no order condition was set) is: first they are ordered by the issue number descending, then by the section number ascending and by the article order in the section ascending

Note: byComments and byLastComment only work in Newscoop 3.2.2 and newer. byLastUpdate is implemented in Newscoop 3.5 and newer.

Inside the list the data context is defined by the constraints applied to the current article for every processed line. The data context is restored after the list processing.

Constraints:

Inside list_articles the following statements are forbidden:

- list_languages
- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article

list_articles statement cannot be used inside list_subtitles, or list_search_results statements.

LIST OF ARTICLE ATTACHMENTS

Purpose:

Create a list of documents attached to the article currently defined in the template environment. If the article was not set the list is empty. The code between "{{ list_article_attachments }}" statement and "{{ /list_article_attachments }} is repeated for every attachment in the list.

Syntax:

```
1. {{ list_article_attachments [length="<integer_value>"]
2.                             [columns="<integer_value>"]
3.                             [language="current"] }}
4.    <list_of_instructions>
5. {{ /list_article_attachments }}
```

Filters:

- all_languages: if true (default) the list will contain all article attachments independent of their language; if false, the list will contain only article attachments that have the language currently defined by the template environment
- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list
- language="current": list only attachments that were set as available for all translations and available for the language currently set in the template environment

Inside list_articles the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article
- list_article_attachments

Inside the list, the current article attachment is set to the current element of the list. The environment context is restored after the list processing.

Example:

```
1. {{ list_article_attachments }}
2.  {{ if $gimme->current_list->at_beginning }}
3.   <h4>Downloads:</h4>
4.  {{ /if }}
5.   <a href="/attachment/{{ $gimme->attachment->identifier }}">
6.    {{ $gimme->attachment->file_name }}</a>
7.   ({{ $gimme->attachment->size_kb }}kb)<br/>
8. {{ /list_article_attachments }}
```

Here is an example of how to use the attachment list to play a number of MP3 files:

```
1. {{ list_article_attachments }}
2.  {{ if $gimme->attachment->extension == "mp3" }}
3. <object type="application/x-shockwave-flash"
4. data="/templates/radioactive/apps/player_mp3_maxi.swf" width="200" height
   ="20">
5. <param name="movie" value="/templates/radioactive/apps/player_mp3_maxi.sw
   f" />
6. <param name="bgcolor" value="#444444"/>
7. <param name="FlashVars" value="mp3={{ uri options="articleattachment" }}"
    />
8. <!-- player home: http://flash-mp3-player.net/ -->
9. </object>
10.  {{ /if }}
11. {{ /list_article_attachments }}
```

LIST OF ARTICLE AUTHORS

Note: this works only in Newscoop 3.5.0 and newer versions.

Purpose:

Create the list of authors to the article currently defined in the template environment. If the article was not defined, the list is empty. The code between the "{{ list_article_authors }}" statement and "{{ /list_article_authors }} is repeated for every author in the list. Inside the list every author is represented by an author object, (see "Author").

Syntax:

```
1. {{ list_article_authors [length="<integer_value>"]
2.                         [columns="<integer_value>"]
3.                         [order="<author_order>"] }}
4.    <list_of_instructions>
5. {{ /list_article_authors }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.
- order="<author_order>"=
    - byFirstName: order by the author's first name
    - byLastName: order by the author's last name

Example:

Taken from Template Pack "The Journal" (date 2011-03-15) file "article-author-popup.tpl"

```
1.  {{ list_article_authors }}
2.   <div id="hidden{{ $gimme->current_list->index }}Content"  class="teammem
    berinfo" style="display:none">
3.    <img style="width: 150px; float: left; margin: 0 10px 10px 0"
4.    src="{{ $gimme->author->picture->imageurl }}" />
5.     <h2>{{ $gimme->author->name }}</h2>
6.      <div class="text">
7.      {{ $gimme->author->biography->text }}
8.      </div>
9.  </div>
10. {{ /list_article_authors }}
```

Inside list_articles the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article
- list_article_images

Inside the list, the current author is set to the current element of the list. The environment context is restored after the list processing.

LIST OF ARTICLE COMMENTS

Purpose:

Create a list of comments attached to the article currently defined in the template environment. If the article was not defined the comments list is empty. The code between the "{{ list_article_comments }}" statement and "{{ /list_article_commnets }}" is repeated for every comment in the list.

Syntax:

```
1. {{ list_article_comments [ignore_language="true|false"]
2.                          [ignore_article="true|false"]
3.                          [length="<integer_value>"]
4.                          [columns="<integer_value>"]
5.                          [order="<order_condition>"] }}
6.    <list_of_instructions>
7. {{ /list_article_comments }}
```

Filters:

- ignore_language: list comments regardless of the comment language
- ignore_article: list comments for all articles, not only for the current article
- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object.
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.

Note on ignore_article: this option will force the list order to date ordering. ignore_language and ignore_article only work in Newscoop 3.2.2 and newer.

- <order_condition>=
  - byDate desc|asc
  - default desc|asc

The default order of the comments in the list (if no order condition was specified) is based on the tree structure of the comments as in the following example:

- root comment 1
  - reply 1 (parent is root comment 1)
    - reply 1_1 (parent is reply 1)
- root comment 2
- ...

The first element in the list is the first comment that was submitted, second is it's first reply (if it exists), first reply of the first reply .. and so on down the tree structure, until it finds no other reply, second reply to the root element etc.

If the order by date condition was specified the comments are displayed strictly by their submission date, regardless of the relation they had to the other comments.

Example:

```
1. {{ list_article_comments order="byDate desc" }}
2.     Subject: {{ $gimme->comment->subject }}<br/>
3.     Posted {{ $gimme->comment->submit_date }}
4.     by <b>{{ $gimme->comment->reader_email }}</b>
5.     <br/>
6.     {{ $gimme->comment->content }}
7.     <br/>
8. {{ /list_article_comments }}
```

The following example is taken from the Template Pack "The Custodian" (2011-03-16) file "comments.tlp"

```
1. {{ list_article_comments }}
2. {{ if $gimme->current_list->at_beginning }}
3.  <a name="commentlist">
4.   <h4>
5.   {{ if $gimme->language->name == "English" }}Previous comments
6.   {{ else }}Los comentarios anteriores
7.   {{ /if }}
8.   </h4>
9.  </a>
10. {{ /if }}
11.  <div class="comment" {{ if $gimme->current_list->at_end }}
12.   id="everlast"{{ /if }}>
13.    <p><strong>{{ $gimme->comment->nickname }}</strong><br>
14.    {{ $gimme->comment->content }}</p>
15.     <p>
16.      <em>
17.       {{ $gimme->comment->subject }} |
18.       <span>{{ $gimme->comment->submit_date|camp_date_format:"%M
19.       %e, %Y" }}</span>
20.      </em>
21.     </p>
22.  </div>
23. <!-- /.comment -->
24. {{ /list_article_comments }}
```

Inside list_articles the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article
- list_article_comments

LIST OF ARTICLE IMAGES

Purpose:

Create a list of images attached to the article currently defined in the template environment. If the article was not defined, the list is empty. The code between the "{{ list_article_images }}" statement and "{{ /list_article_images }} is repeated for every image in the list.

Syntax:

```
{{ list_article_images [length="<integer_value>"]
                       [columns="<integer_value>"]
                       [order="<image_order>"] }}
   <list_of_instructions>
{{ /list_article_images }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.
- order="<image_order>"=
- default, number: order by the number assigned to the image when attached to an article
- byDescription: order by the image description
- byPhotographer: order by the photographer name
- byDate: order by the image date
- byLastUpdate: order by the image last update time

Example:

Taken from Template Pack "The Custodian" (date 2011-03-15) file "article-gallery.tpl"

```
{{ list_article_images }}
 {{ if $gimme->current_list->count gt 1 }}
  {{ if $gimme->current_list->at_beginning }}
   <div id="article-gallery">
    <h4>{{ if $gimme->language->name == "English" }}Article gallery
        {{ else }}Mini galera{{ /if }}:
    </h4>
  {{ /if }}
     <div class="gallery-item">
      <a class="grouped_elements"
       href="{{ $gimme->article->image->imageurl }}" rel="group">
       <img alt="{{ $gimme->article->image->description }}"
       src="{{ $gimme->article->image->thumbnailurl }}" />
      </a>
     </div><!-- /.gallery-item -->
   {{ if $gimme->current_list->at_end }}
   </div><!-- /#article-gallery -->
   {{ /if }}
 {{ /if }}
{{ /list_article_images }}
```

Inside list_articles the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article
- list_article_images

Inside the list, the current image is set to the current element of the list. The environment context is restored after the list processing.

LIST OF ARTICLE LOCATIONS

Note: this works only in Newscoop 3.5.0 and newer versions.

Maps require jQuery: In order to use display maps you must include jQuery in the header of your document, with a link like this:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/
1/jquery.min.js">
```

Purpose:

Create a list of map locations related to the article currently defined in the template environment. If the article was not defined, the list is empty. The code between the "{{ list_article_locations }}" statement and "{{ /list_article_locations }} is repeated for every location in the list. Inside the list every location is represented by a map location object, (see "Article Location").

Syntax:

```
1. {{ list_article_locations [length="<integer_value>"]
2.                           [columns="<integer_value>"] }}
3.    <list_of_instructions>
4. {{ /list_article_images }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the intervals of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.

Example:

Ushahidi compatible KML format. You can use this as a feed to call as a layer in Ushahidi:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <kml xmlns="http://earth.google.com/kml/2.2">
3.  <Document>
4.    <name>{{ $gimme->publication->name }}</name>
5.    <description><![CDATA[ .]]></description>
6.  {{ list_articles}}
7.    {{ list_article_locations }}
8.      {{ if $gimme->location->enabled }}
9.    <Style id="style{{ $gimme->article->number }}-{{ $gimme->current_list->
    index }}">
10.     <IconStyle>
11.       <Icon>
12.        <href>http://www.sourcefabric.org/javascript/geocoding/markers/mar
    ker-gold.png</href>
13.       </Icon>
14.     </IconStyle>
15.   </Style>
16.     {{ /if }}
17.   {{ /list_article_locations }}
18. {{ /list_articles }}
19.
20. {{ list_articles}}
21.   {{ list_article_locations }}
22.     {{ if $gimme->location->enabled }}
23.   <Placemark>
24.     <name>{{ $gimme->location->name }} @ {{
25. $gimme->article->name }}</name>
26.     <description></description>
27.     <styleUrl>#style{{ $gimme->article->number }}-{{
28. $gimme->current_list->index }}</styleUrl>
29.     <Point>
30.       <coordinates>{{ $gimme->location->longitude
31. }},{{ $gimme->location->latitude }},0.000000</coordinates>
32.     </Point>
33.   </Placemark>
34.     {{ /if }}
35.   {{ /list_article_locations }}
36. {{ /list_articles }}
37. </Document>
38. </kml>
```

Taken from Template Pack "The Journal" (date 2011-03-15) file "article-cont.tpl"

```
1.  <p>
2.  Location(s):
3.  {{ list_article_locations }}
4.   {{ if $gimme->location->enabled }}
5.    {{ $gimme->location->name }}
6.     {{ if $gimme->current_list->at_end }}{{ else }}, {{ /if }}
```

```
7.  {{ /if }}
8. {{ /list_article_locations }}
9. </p>
```

Inside list_article_locations the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article
- list_article_images

Inside the list, the current location is set to the current element of the list. The environment context is restored after the list processing.

77. LIST ARTICLE TOPICS AND SUBTOPICS

LIST OF ARTICLE TOPICS

Purpose:

Create a list of topics attached to the article currently defined in the template environment. If the article was not set, the list is empty. The topic parameter may not be defined outside the list_article_topics statement; inside the statement however, this parameter is defined. The code between "{{ list_article_topics }}" statement and "{{ /list_article_topics }}" is repeated for every topic in the list.

Syntax:

```
1. {{ list_article_topics [length="<integer_value>"]
2.                        [columns="<integer_value>"] }}
3.   <list_of_instructions>
4. {{ /list_article_topics }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object.
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.

Examples:

Taken from Template Pack "The Custodian" (date 2011-03-16) file "article.tpl"

```
1. {{ list_article_topics }}
2.    {{ assign var="topic_cond" value="`$topic_cond` topic is `$gimme->topi
   c->identifier` " }}
3. {{ /list_article_topics }}
```

Taken from Template Pack "The Custodian" (date 2011-03-16) file "topic-list.tpl"

```
 1. {{ list_article_topics }}
 2.  {{ if $gimme->current_list->at_beginning }}
 3.   {{ if $gimme->language->name == "English" }}Related topics
 4.   {{ else }}Temas relacionados
 5.   {{ /if }}
 6.  {{ /if }}
 7.  : {{$gimme->topic->name }}
 8.  {{ if $gimme->current_list->at_end }}
 9.  {{ /if }}
10. {{ /list_article_topics }}
```

Inside list_articles the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article
- list_article_topics

Inside the list, the current topic is set to the current element of the list. The environment context is restored after the list processing.

LIST OF SUBTOPICS

Purpose:

Create a list of subtopics of the topic currently set in the template environment. If the topic was not set it will generate the list of root topics. The topic parameter may not be defined outside the list statement; inside the statement however, this parameter is defined. The code between "{{ list_subtopics }}" statement and "{{ /list_subtopics }}" is repeated for every topic in the list.

Syntax:

```
1. {{ list_subtopics [length="<integer_value>"]
2.                   [columns="<integer_value>"]
3.                   [order="<order_condition>"] }}
4.    <list_of_instructions>
5. {{ /list_subtopics }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object.
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.

<list_of_instructions> may contain any statement except those listed at the end of the page.

- <order_condition>=
    - byNumber desc|asc
    - byName desc|asc

Inside the list, the data context is defined by the constraints applied to the current topic for every processed line. The data context is restored after the list processing.

Constraints:

Inside list_subtopics the following statements are forbidden:

- list_subtopics
- set_topic
- set_default_topic
- unset_topic

78. LIST OF ISSUES AND SECTIONS

LIST OF ISSUES

Purpose:

Select the list of issues according to the given constraints and current environmental variables. The publication, language and issue variables may not be defined outside the list_issues statement; inside the statement however, all these variables are defined. The code between "{{ list_issues }}" statement and "{{ /list_issues }}" is repeated for every issue in the list.

Syntax:

```
{{ list_issues [length="<integer_value>"]
               [columns="<integer_value>"]
               [constraints="<list_of_issue_constraints>"]
               [order="<order_condition>"] }}
   <list_of_instructions>
{{ /list_issues }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object.
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.

<list_of_instructions> may contain any statement except: "set_language", "set_publication", "list_issues", "set_issue".

- <list_of_issue_constraints>=
  - <issue_constraint> <list_of_issue_constraints>
  - <issue_constraint>
- <issue_constraint>=
  - number<integer_operator> <integer_value>
  - name <string_operator> <string_value>
  - publish_date <date_operator> <date_value>
  - publish_year <integer_operator> <integer_value>
  - publish_month <integer_operator> <integer_value>
  - publish_mday <integer_operator> <integer_value>
  - <date_constraint>
- <date_constraint>=
  - year <integer_operator> <integer_value>
  - mon_nr <integer_operator> <integer_value>
  - mday <integer_operator> <integer_value>
  - yday <integer_operator> <integer_value>

- wday <integer_operator> <integer_value>
- hour <integer_operator> <integer_value>
- min <integer_operator> <integer_value>
- sec <integer_operator> <integer_value>

where year stands for year, mon_nr for month number (1..12), mday for month day (1..31), yday for year day (1..365), wday for week day (1..7), hour for hour, min for minute and sec for second.

Any parameter used in <list_of_issue_constraints> can only be used once.

- <order_condition>=
  - byNumber desc|asc
  - byName desc|asc
  - byDate desc|asc
  - byCreationDate desc|asc
  - byPublishDate desc|asc

Order conditions are self-explanatory; byDate and byCreationDate are aliases of byPublishDate.

Inside the List the following environment variables are modified:

- language: if not defined before list start
- publication: if not defined before list start
- issue

The environment is restored after the list ends.

Examples:

Taken from Template Pack "The Custodian" (date 2011-03-16) file "archive.tpl"

```
1.  {{ list_issues length="10" order="byNumber desc" }}
2.   <h2>
3.    <a href="{{ url options="template classic/archive.tpl" }}">
4.    {{ $gimme->issue->name }}</a>
5.   </h2>
6.    {{ if $gimme->language->name == "English" }}Issue
7.    {{ else }}Edicin{{ /if }}
8.     #{{ $gimme->issue->number }} /
9.    ({{ if $gimme->language->name == "English" }}published
10.   {{ else }}publicado{{ /if }}
11.  {{ $gimme->issue->publish_date|camp_date_format:'%M %D, %Y %h:%i:%s' }})
12.  {{ include file="skins/greenpiece/includes/pagination.tpl" }}
13. {{ /list_issues }}
```

Constraints:

Inside list_issues the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue

- set_default_issue
- unset_issue

list_issues statement can not be used inside any other list statements.

LIST OF SECTIONS

Purpose:

Select the list of sections according to the given constraints and current environmental variables. The publication, language, issue and section variables may not be defined outside the list_sections statement; inside the statement however, all these variables are defined. The code between "{{ list_sections }}" statement and "{{ /list_sections }}" is repeated for every section in the list.

Syntax:

```
1. {{ list_sections [length="<integer_value>"]
2.                  [columns="<integer_value>"]
3.                  [constraints="<list_of_section_constraints>"] }}
4.    <list_of_instructions>
5. {{ /list_sections }}
```

Filters:

length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object.

columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.

<list_of_instructions> may contain any statement except: "set_language", "set_publication", "list_issues", "set_issue", "list_section", "set_section".

- <list_of_section_constraints>=
  - <section_constraint> <list_of_section_constraints>
  - <section_constraint>
- <section_constraint>=
  - name <string_operator> <string_value>
  - number <integer_operator> <integer_value>

Any parameter used in <list_of_section_constraints> can only be used once.

Inside list_sections, the data context is defined by the constraints applied to the current section for every processed line. The data context is restored after the list processing.

Example:

Taken from the Template Pack "The Custodian" (date 2011-03-16) file "archive.tpl"

```
1. {{ list_sections order="bynumber asc" }}
```

```
  <a href="{{ uri }}" class="linksection-{{ $gimme->section->number }}">
  {{ $gimme->section->name }}</a>
  <!--{{ $gimme->section->number }}-->
 </h3>
 <ul>
  {{ list_articles }}
  <li id="list-article">
   <h4>
    <a href="{{ uri }}" class="linksection-{{ $gimme->section->number }}"
>
    {{ $gimme->article->name }}</a>
   </h4>
  <div class="list-article-published">
    {{ if $gimme->language->name == "English" }}posted
    {{ else }}publicado el
    {{ /if }}
   {{ $gimme->article->publish_date|camp_date_format:'%M %D, %Y %h:%i:%s'
 }}
  </div>
  {{ include file="classic/tpl/topic-list.tpl" }}
  </li>
  {{ /list_articles }}
 </ul>
{{ /list_sections }}
```

Constraints:

Inside list_sections the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- unset_issue
- list_section
- set_section
- set_default_section
- unset_section

The list_sections statement cannot be used inside list_articles, list_subtitles, or list_search_results statements.

79. LIST SUBTITLES (PAGINATION OF LONG ARTICLES)

LIST OF SUBTITLES

Purpose:

Create a list of subtitles for the content of the article currently defined in the template environment. The article, if not specified somewhere else, is treated as group of paragraphs. The markup for a new paragraph is the subtitle. The code between the "{{ list_subtitles }}" statement and "{{ /list_subtitles }}" is repeated for every subtitle in the list.

Syntax:

```
1. {{ list_subtitles [length="<integer_value>"]
2.                   [columns="<integer_value>"] }}
3.    <list_of_instructions>
4. {{ /list_subtitles }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list

<list_of_instructions> may contain any statement except: "set_language", "set_publication", "list_issues", "set_issue", "list_sections", "set_section", "list_articles", or "set_article".

Inside the list, the data context is defined by the constraints applied to the current article for every processed line. The data context is restored after the list processing.

Constraints:

Inside list_articles the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue

- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article

80. LIST IMAGES

LIST OF IMAGES

Purpose:

Create a list of images. The list is built from the Media Archive independently, whether the images are related to articles or not. The code between the "{{ list_images }}" statement and "{{ /list_images }}" is repeated for every image in the list. Inside the list every image is represented by an image object, (see "Image").

Syntax:

```
1. {{ list_images [length="<integer_value>"]
2.                [columns="<integer_value>"]
3.                [<filter>="<filter_value>"]}}
4.    <list_of_instructions>
5. {{ /list_images }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object.
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.
- order="<order_field> <direction>" where <direction> may be "asc" (ascending) or "desc" (descending) and the <order_field> can be one of:
    - default: order by image identifier
    - byDescription: order by the description field
    - byPhotographer: order by the photographer field
    - byDate: order by image date field
    - byLastUpdate: order by the time the image was updated
- description="<value>": select images whose description matches exactly the given value
- photographer="<value>": select images whose photographer matches exactly the given value
- place="<value>": select images whose place matches exactly the given value
- caption="<value>": select images whose caption matches exactly the given value
- date="<value>": select images whose date matches exactly the given value
- type="<value>": select images whose type matches exactly the given value; valid type values are: png, jpeg, gif etc.
- description_like="<value>": select images whose description matches partially the given value
- photographer_like="<value>": select images whose photographer matches partially the given value
- place_like="<value>": select images whose place matches partially the given value
- caption_like="<value>": select images whose caption matches partially the given value
- start_date="<value>": select images whose date matches or is greater than the given value
- end_date="<value>": select images whose date matches or is smaller than the given value
- search="<value>": select images whose description, photographer, place and caption matches partially the given value
- local="true" | "false": select only local images if true, remote images if false

Inside list_images the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections

- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article
- list_article_attachments

Inside the list, the current image is set to the current element of the list. The environment context is restored after the list processing.

Example:

```
1. {{ list_images photographer="John\ Doe" order="byLastUpdate" }}
2.     <img src="{{ $gimme->image->thumbnailurl }}" /><br />
3.     <p>{{ $gimme->image->description }}</p>
4. {{ /list_images }}
```

81. LIST LANGUAGES

LIST OF LANGUAGES

Note: this works only in Newscoop 3.2.1 and newer versions.

Purpose:

Select the list of languages according to the given constraints and current environmental variables. The language variable may not be defined outside the list_languages statement; inside the statement however, this variable is defined. The code between the {{list_languages}} statement and {{/list_languages}} is repeated for every language in the list.

Syntax:

```
{{ list_languages [length="<integer_value>"]
                  [columns="<integer_value>"]
                  [of_publication="true|false"]
                  [of_issue="true|false"]
                  [of_article="true|false"]
                  [exclude_current="true|false"]
                  [order="<order_condition>"] }}
   <list_of_instructions>
{{ /list_languages }}
```

Filters:

- length="<integer_value>": <integer_value> specifies the list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object.
- columns="<integer_value>":<integer_value> specifies the list columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.
- of_publication: if true, will list languages in which publication issues were translated
- of_issue: if true, will list languages in which the current issue was translated
- of_article: if true, will list languages in which the current article was translated
- exclude_current: if true, will not include the current language in the list

When none of the three attributes (of_publication, of_issue, of_section) was specified it will list all available languages in Newscoop.

<list_of_instructions> may contain any statement except those listed at the end of the page.

- <order_condition>=
  - byNumber desc|asc
  - byName desc|asc
  - byEnglish_Name desc|asc
  - byCode desc|asc

Order conditions are self-explanatory.

Inside the list, the following environment variable is modified:

- language: if not defined before list start

The environment is restored after the list ends.

Constraints:

Inside list_issues the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- list_languages

82. LIST SEARCH RESULTS

LIST OF SEARCH RESULTS

Purpose:

Create a list of articles that match the search keywords entered by the reader. The publication, language, issue, section, and article variables may not be defined outside the list_search_results statement; inside the statement however, all these variables are defined. The code between the "{{ list_search_result }}" statement and "{{ /list_search_result }}" is repeated for every article in the list.

Syntax:

```
1. {{ list_search_results [length="<integer_value>"]
2.                        [columns="<integer_value>"]
3.                        [order="<order_condition>"] }}
4.    <list_of_instructions>
5. {{ /list_search_result }}
```

Filters:

- length="<integer_value>": <integer_value> specifies list_length and forces the list to have at most list_length items. If the list contains more items than list_length items the interval of elements to be displayed can be switched using has_previous_elements and has_next_elements from the current_list object.
- columns="<integer_value>": <integer_value> specifies columns_number and sets an environment variable. This is incremented as if the items were placed in a table cell. The counting starts from one and the variable is incremented for every new element. When it reaches the maximum value it is reset to one. This is very useful in building tables of data. For details see current_list.

<list_of_instructions> may contain any statement except: "set_language", or "set_publication".

- <order_condition>=
  - byDate desc|asc
  - byCreationDate
  - byPublishDate
  - byNumber desc|asc
  - byName desc|asc

Order conditions are self-explanatory; byDate is an alias of byCreationDate. The default order of articles in the list (if no order condition was set) is: first they are ordered by the publication identifier ascending, then by the issue number descending, then by the section number ascending, and by the article order in the section ascending.

Inside the list, the data context is defined by the constraints applied to the current article for every processed line. The data context is restored after the list processing.

Constraints:

Inside list_articles the following statements are forbidden:

- set_language
- set_default_language
- unset_language
- set_publication
- set_default_publication
- unset_publication
- list_issues
- set_issue
- set_default_issue
- set_current_issue
- unset_issue
- list_sections
- set_section
- set_default_section
- unset_section
- list_articles
- set_article
- set_default_article
- unset_article

TEMPLATE REFERENCE - FORMS

83. SEARCH FORM
84. LOGIN AND REGISTRATION FORM
85. SUBSCRIPTION FORM
86. COMMENT FORM
87. ACTION CHECK ON FORM SUBMISSION
88. GENERAL FORM ELEMENTS AND FUNCTIONS

83. SEARCH FORM

FORM SEARCH

Purpose:

Generate the search form and data fields for searching keywords in published articles. By default the search action is performed in the current publication only.

Syntax:

```
1. {{ search_form [template="<template_name>"] submit_button="<button_name>"
2.                [html_code="<html_code>"]
3.                [button_html_code="<html_code>"] }}
4.   <list_of_instructions>
5. {{ /search_form }}
```

- <template_name> is the name of the next template to be requested from the search form
- <button_name> is the name of the button for submitting the form
- html_code: you can insert whatever HTML code you want in the <form> statement; for example: {{ user_form .. html_code="id=\"my_id\"" }}
- button_html_code: you can insert whatever HTML code you want in the button input field statement; such as: {{ user_form .. button_html_code="id=\"my_submit_id\"" }}

The list of instructions may contain any instruction allowed in the current context.

Setting the search scope: whether to search in all publications, in the current publication, in the current issue or in the current section. Insert the following field in the search form:

```
{{ camp_select object="search" attribute="level" }}
```

Constraints:

Can not be used within itself (e.g. search in search).

EDIT SEARCH

Purpose:

Generates a text input field so that a reader can search for articles on your site. This statement should be used inside the search form.

Syntax:

```
{{ camp_edit object="search" attribute="<attribute>"
                             [html_code="<HTML_code>"]
                             [size ="<field_length>"] }}
```

Filters:

<attribute> being one of the following:

- keywords: allows search keywords input
- start_date: select only articles published starting on the selected date
- end_date: select only articles published up to the selected date

Constraints:

The search edit field can be used only inside the search form.

SELECT SEARCH

Purpose:

Generates a check box or a pop-up list for selecting the search mode or the search level respectively. This statement should be used inside the search form.

Syntax:

```
{{ camp_select object="search" attribute="<attribute>"
                               [html_code="<HTML_code>"] }}
```

Filters:

<attribute> being one of the following:

- mode: if checked, the search will return articles that contain all entered keywords; if not checked, it will return articles that contain at least one keyword
- level: allows the reader to select the scope of the search: multiple publications, publication, issue, section
- section: select only articles that are in the given section
- issue: select only articles that are in the given issue

Constraints:

The select search field can only be used inside the search form.

84. LOGIN AND REGISTRATION FORM

FORM - LOGIN

Purpose:

Generate the form and data fields for logging in a user.

Syntax:

```
{{ login_form [template="<template_name>"] submit_button="<button_name>"
              [html_code="<html_code>"]
              [button_html_code="<html_code>"] }}
  <list_of_instructions>
{{ /login_form }}
```

- <template_name> is the name of the next template to be requested from the login form
- <button_name> is the name of the button for submitting the form
- html_code: you can insert whatever HTML code you want in the <form> statement; for example: {{ user_form .. html_code="id=\"my_id\"" }}
- button_html_code: you can insert whatever HTML code you want in the button input field statement; such as: {{ user_form .. button_html_code="id=\"my_submit_id\"" }}

Constraints:

Cannot be used inside subscription and user forms. Cannot be used within itself (e.g. login in login).

A simple implementation of a login form for the public website of your publication:

```
{{ if ! $gimme->user->logged_in }}
 <p>Login</p>
  {{ if $gimme->login_action->is_error }}
   <p>There was an error logging in:
   {{ $gimme->login_action->error_message }}</p>
  {{ /if }}
 {{ login_form submit_button="Login" button_html_code="class=\"submitbutt
on\"" }}
  <p>User ID: {{ camp_edit object="login" attribute="uname" }}</p>
  <p>Password: {{camp_edit object="login" attribute="password" }}</p>
 {{ /login_form }}
{{ else }}
 <p>Welcome {{ $gimme->user->name }}</p>
 <p><a href="?logout=true">Logout</a></p>
{{ /if }}
```

The logout requires this code in the head of every page:

```
{{ if $gimme->url->get_parameter('logout') == 'true' }}
 <META HTTP-EQUIV="Set-Cookie" CONTENT="LoginUserId=; path=/">
 <META HTTP-EQUIV="Set-Cookie" CONTENT="LoginUserKey=; path=/">
 {{ $gimme->url->reset_parameter('logout') }}
 <META HTTP-EQUIV="Refresh" content="0;url={{ uri }}">
{{ /if }}
```

EDIT LOGIN

Purpose:

Generates a text input field for entering a subscriber's login user name or password. Use this to allow a subscriber to login to your site. These statements should be used inside the login form.

Syntax:

```
{{ camp_edit object="login" attribute="<attribute>"
                            [html_code="<HTML_code>"]
                            [size ="<field_length>"] }}
```

Filters:

<attribute> being one of the following:

- uname: allows login name input
- password: allows password input

Constraints:

The login edit fields can only be used inside the login form.

SELECT LOGIN

Purpose:

Generates a check box; if checked the user session will be remembered for a period of two weeks so the user will not have to login again. This statement should be used inside the login form.

Syntax:

```
{{ camp_select object="login" attribute="rememberuser"
                              [html_code="<HTML_code>"] }}
```

Constraints:

The select login field can only be used inside the login form.

LOGOUT

There is no template keyword to logout a subscriber. Instead, just put these two lines in your logout.tpl file:

```
<META HTTP-EQUIV="Set-Cookie" CONTENT="LoginUserId=; path=/">
<META HTTP-EQUIV="Set-Cookie" CONTENT="LoginUserKey=; path=/">
```

FORM - USER

Purpose:

Generate the form and data fields for adding a new user, or editing an existing user's data.

Syntax:

```
1. {{ user_form [template="<template_name>"] submit_button="<button_name>"
2.              [html_code="<html_code>"]
3.              [button_html_code="<html_code>"] }}
4.   <list_of_instructions>
5. {{ /user_form }}
```

- <template_name> is the name of the next template to be requested from the user form
- <button_name> is the name of the button for submitting the form
- html_code: you can insert whatever HTML code you want in the <form> statement; for example: {{ user_form .. html_code="id=\"my_id\"" }}
- button_html_code: you can insert whatever HTML code you want in the button input field statement; such as: {{ user_form .. button_html_code="id=\"my_submit_id\"" }}

Constraints:

Cannot be used inside subscription and login forms. Cannot be used within itself (for example: user in user).

EDIT USER

Purpose:

Generates a text input field for editing the selected user attribute. This statement should be used inside the user form.

Syntax:

```
1. {{ camp_edit object="search" attribute="<attribute>"
2.                              [html_code="<HTML_code>"]
3.                              [size ="<field_length>"]
4.                              [columns="<max_columns>"]
5.                              [rows="<max_rows>"] }}
```

Filters:

<attribute> being one of the following:

- name
- uname
- email
- city
- str_address
- state
- phone
- fax
- contact
- second_phone
- postal_code
- employer
- position

- how
- languages
- field1
- field2
- field3
- field5
- interests
- improvements
- text1
- text2
- text3
- password
- passwordagain

The parameters (name, uname, email, city...) are fields describing the user's data; field1-field5, text1-text3 are extra fields for storing extra information of your choice.

Note: the columns and rows parameters were implemented starting with Newscoop version 3.2.2.

Constraints:

The user edit field can only be used inside the user form.

SELECT USER

Purpose:

Generates a drop-down list or radio buttons for selecting values for the given fields. This statement should be used inside the user form.

Syntax:

```
{{ camp_select object="user" attribute="<attribute>"
                             [html_code="<HTML_code>"] }}
```

Filters:

<attribute> being one of the following:

- gender
- title
- country
- age
- employertype
- pref1
- pref2
- pref3
- pref4

The parameters (country, title, gender...) are fields describing the user information; pref1-pref4 are extra fields for storing extra information (the publication administrator sets their meaning).

Constraints:

The select user field can only be used inside the user form.

85. SUBSCRIPTION FORM

FORM - SUBSCRIPTION

Purpose:

Generate the form and default data fields for subscription. Edit and Select statements can be used to generate fields for every section in an issue.

Syntax:

```
1. {{ subscription_form type="<subscription_type>"
2.                          [template="<template_name>"]
3.                          submit_button="<button_name>"
4.                          [total="<total_field_name>"]
5.                          [html_code="<html_code>"]
6.                          [button_html_code="<html_code>"] }}
7.    <list_of_instructions>
8. {{ /subscription_form }}
```

- <subscription_type> has two values: by_publication and by_section
- <template_name> is the name of the next template to be requested from the subscription form
- <button_name> is the name of the button for submitting the form
- <total_field_name> is the name of the field where the total payment is written
- <evaluate_button_name> is the name of the evaluate button (when pressing this button, the total payment is updated in the total field
- html_code: you can insert whatever HTML code you want in the <form> statement; for example: {{ user_form .. html_code="id=\"my_id\"" }}
- button_html_code: you can insert whatever HTML code you want in the button input field statement; such as: {{ user_form .. button_html_code="id=\"my_submit_id\"" }}

Constraints:

Cannot be used inside user and login forms. Cannot be used within itself (for example: subscription in subscription).

EDIT SUBSCRIPTION

Purpose:

Generates an input field for editing the subscription time for a certain section. The section must be defined in the template environment. This statement should be used inside the subscription form.

Syntax:

```
{{ camp_edit object="search" attribute="time"
                             [html_code="<HTML_code>"]
                             [size ="<field_length>"] }}
```

Constraints:

The subscription edit field can only be used inside the subscription form.

SELECT SUBSCRIPTION

Purpose:

Special-purpose input for subscription data.

Syntax:

```
{{ camp_select object="subscription" attribute="<attribute>"
                                     [html_code="<HTML_code>"] }}
```

Filters:

<attribute> being one of the following:

- section: generates a checkbox for selecting the current section in the subscription form; use together with "List of Sections" (see also "Creating reader subscriptions through the template language")
- allLanguages: generates a checkbox allowing the user to choose a subscription to all available languages
- languages: generates a multiple selection list allowing the user to select individual languages; this selection list is automatically deactivated if the user chose to subscribe to all available languages.

Constraints:

The select subscription field can only be used inside the subscription form.

86. COMMENT FORM

FORM - COMMENT

Purpose:

Generate the form and default data fields for article comment submission. Inside the form camp_edit statements can be used to generate article comment input fields.

Syntax:

```
{{ comment_form [template="<template_name>"]
                submit_button="<submit_button_name>"
                [preview_button="<preview_button_name>"]
                [html_code="<html_code>"]
                [button_html_code="<html_code>"] }}
   <list_of_instructions>
{{ /comment_form }}
```

- <template_name>: the name of the next template to be requested when submitting the comment
- <submit_button_name>: the name of the button for submitting the comment
- <preview_button_name>: the name of the button for previewing the comment
- html_code: you can insert whatever HTML code you want in the <form> statement; for example: {{ user_form .. html_code="id=\"my_id\"" }}
- button_html_code: you can insert whatever HTML code you want in the button input field statement; such as: {{ user_form .. button_html_code="id=\"my_submit_id\"" }}

EDIT COMMENT

Purpose:

Generates text fields so that a user can enter a comment. This statement should be used inside the comment form.

Syntax:

```
{{ camp_edit object="comment" attribute="<attribute>"
                              [html_code="<HTML_code>"]
                              [size ="<field_length>"]
                              [columns="<max_columns>"]
                              [rows="<max_rows>"] }}
```

Filters:

<attribute> being one of the following:

- nickname: a place for the user to type in a nickname
- reader_email: a place for the user to type in their email address; this field is optional for logged in readers because the email can be read from the subscribers database
- subject: a place for the user to type in a subject line for their comment
- content: a place for the user to type in their comment.

The attributes reader_email, subject and content are mandatory in each Edit Comment form.

Note: the columns and rows parameters were implemented starting with Newscoop version 3.2.2.

Constraints:

The comment edit field can only be used inside the comment form.

EDIT CAPTCHA

Purpose:

Generates a text field allowing the user to input the CAPTCHA code for spam control. This statement should be used inside the comment form.

Syntax:

```
{{ camp_edit object="captcha" attribute="code"
                              [html_code="<HTML_code>"]
                              [size ="<field_length>"] }}
```

CAPTCHA IMAGELINK

Purpose:

"captcha_image_link" will insert the link for the CAPTCHA image. Use inside an "img" HTML tag as follows:

```
<img src="{{ captcha_image_link }}">
```

Syntax:

```
{{ captcha_image_link }}
```

87. ACTION CHECK ON FORM SUBMISSION

EDIT SUBSCRIPTION ACTION

The edit_subscription_action object is initialized when a subscription form was submitted. It has the following properties:

- defined: true if a subscription submit action took place
- is_error: true if a subscription submit action took place and the result was an error
- error_code: error code of the subscription submit action; null if no subscription submit action took place
- error_message: error message of the subscription submit action; null if no subscription submit action took place
- ok: true if a subscription submit action took place and the result was success
- is_trial: true if the submitted subscription type was trial
- is_paid: true if the submitted subscription type was paid subscription

EDIT USER ACTION

The edit_user_action object is initialized when a user add/edit form was submitted. It has the following properties:

- defined: true if a user data submit action took place
- is_error: true if a user data submit action took place and the result was an error
- error_code: error code of the user data submit action; null if no user data submit action took place
- error_message: error message of the user data submit action; null if no user data submit action took place
- ok: true if a user data submit action took place and the result was success
- type: can take one of the following two values: "add" for user add submit, "edit" for an existing user data edit submit
- name: the user full name as filled in the form
- uname: the user login name as filled in the form
- password: the account password as filled in the form
- passwordagain: the password confirmation
- email: the user email as filled in the form
- city
- str_address
- state
- phone
- fax
- contact
- second_phone
- postal_code
- employer
- position
- interests
- how
- languages
- improvements
- field1

- field2
- field3
- field4
- field5
- text1
- text2
- text3
- country
- title
- gender
- age
- employertype
- pref1
- pref2
- pref3
- pref4

LOGIN ACTION

The login_action object is defined when a login action takes place. It has the following properties:

- defined: true if a login action took place
- is_error: true if a login action took place and the result was an error
- error_code: error code of the login action; null if no login action took place
- error_message: error message of the login action; null if no login action took place
- ok: true if a login action took place and the result was successful
- user_name: the login name of the user that attempted to log in
- remember_user: true if the remember user flag was set in the login form

PREVIEW COMMENT ACTION

The preview_comment_action object is initialized when the preview button is clicked on a comment form. It has the following properties:

- defined: true if a comment preview action took place
- is_error: true if a comment preview action took place and the result was an error
- error_code: error code of the comment preview action; null if no comment preview action took place
- error_message: error message of the comment preview action; null if no comment preview action took place
- ok: true if a comment preview action took place and the result was successful
- subject: the comment subject as filled in the form
- content: the comment content as filled in the form
- reader_email: the comment reader email as filled in the form

SEARCH ARTICLES ACTION

The search_articles_action object is initialized when a search action takes place. It has the following properties:

- defined: true if a search action took place
- is_error: true if a search action took place and the result was an error
- error_code: error code of the search action; null if no search action took place

- error_message: error message of the search action; null if no search action took place
- ok: true if a search action took place and the result was successful
- search_phrase: the phrase for which the search was performed
- search_keywords: an array of keywords for which the search was performed
- match_all: true if the match all flag was set
- search_level: 0 for multiple publication search, 1 for current publication search, 2 for current issue search, 3 for current section search
- submit_button: the submit button text
- template: the template used on search form submission

SUBMIT COMMENT ACTION

The submit_comment_action object is initialized when a comment form is submitted. It has the following properties:

- defined: true if a comment submit action took place
- is_error: true if a comment submit action took place and the result was an error
- error_code: error code of the comment submit action; null if no comment submit action took place
- error_message: error message of the comment submit action; null if no comment submit action took place
- ok: true if a comment submit action took place and the result was successful
- subject: the comment subject as filled in the form
- content: the comment content as filled in the form
- reader_email: the comment reader email as filled in the form

88. GENERAL FORM ELEMENTS AND FUNCTIONS

FORM PARAMETERS

"formparameters" prints the runtime environment parameters in HTML form format. It prints only data context parameters and can be used to create links.

Syntax:

```
{{ formparameters [options="<options_list>"] }}
```

- <options_list> =
  - <option>
  - <option> <options_list>

- <option> =
  - fromStart
  - articleComment

Filters:

- fromstart: prints the parameters received at the start of the template, not the current ones (useful when building site maps)
- articleComment: inserts the article comment identifier in the parameters list; if no article comment was defined in the template environment, no parameter is inserted. This attribute is needed if you want to pass the current comment id to another page for display purposes.

OPTION INPUT FIELDS

Purpose:

Generate an option input field; these fields are used in forms to allow the reader to enter data. These must be used in conjunction with the form statements.

Syntax:

```
1. {{ camp_select object="<object_name>"
2.                attribute="<attribute_name>"
3.                [html_code="<HTML_code>"] }}
```

If the html_code parameter was set, the HTML code will be inserted inside the input field.

Example:

```
{{ camp_select object="user"
               attribute="country"
               html_code="id=\"countryId\"" }}
```

The following code will output a pop-up list of available countries:

```
1. <select name="f_user_country" id="countryId">
2. ...
3. </select>
```

TEXT INPUT FIELDS

Purpose:

Generate an input text field; these fields are used in forms to allow the reader to enter data. These must be used in conjunction with the form statements.

Syntax:

```
1. {{ camp_edit object="<object_name>"
2.              attribute="<attribute_name>"
3.              [html_code="<HTML_code>"]
4.              [size ="<field_length>"]
5.              [columns="<max_columns>"]
6.              [rows="<max_rows>"] }}
```

For text input fields, if the size parameter was set, the input field size will be set to that value. For text box fields, if the columns/rows parameter was set, the box will have the given number of columns/rows.

Note: columns and rows parameters were implemented starting with Newscoop version 3.2.2.

If the html_code parameter was set, the HTML code will be inserted inside the input field. For example, the following code:

```
{{ camp_edit object="user"
             attribute="name"
             html_code="id=\"userNameInput\"" }}
```

will output:

```
<input type="text" name="f_user_name" size="50" maxlength="255" id="userNameInp
ut">
```

TEMPLATE REFERENCE - MAPS AND GEOLOCATION

89. MAP
90. DISPLAYING A MAP
91. LOCATION MULTIMEDIA
92. GEOLOCATION SEARCH

89. MAP

Note: this works only in Newscoop 3.5.0 and later versions.

Requires jQuery: In order to use display maps you must include jQuery in the header of your document, with a link like this:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/
1/jquery.min.js">
```

The map_ object is usually initialized by the current article in the environment (see "Article"). It is not initialized at the beginning of the template and can not be initialized by other Newscoop functions. The map_ object has the following properties:

- name: the name given to the map
- provider: the map provider
- locations: a list of map locations, see the "Article Location" object
- is_enabled: TRUE if the map is enabled, FALSE otherwise

Examples

```
1. {{ if $gimme->map->is_enabled }}
2.     <p>Map Name: {{ $gimme->map->name }}</p>
3.     <p>Map Provider: {{ $gimme->map->provider }}</p>
4. {{ /if }}
```

90. DISPLAYING A MAP

Note: this works only in Newscoop 3.5.0 and later versions.

Maps require jQuery: In order to use display maps you must include jQuery in the header of your document, with a link like this:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/
1/jquery.min.js">
```

Purpose:

"map" prints a geolocation map and the list of locations. The article containing the map must have been previously initialized, otherwise this function will display nothing.

Syntax:

```
{{ map [width="<width_value>"]
       [height="<height_value>" ]
       [show_locations_list="<boolean_value>" ]
       [show_reset_link="<string_value>|<boolean_value>" ] }}
```

Filters:

- width: <width_value> Width size for the Map frame to be displayed
- height: <height_value> Height size for the Map frame to be displayed
- show_locations_list: <boolean_value> Whether the list of map locations is displayed or not. Default value FALSE
- show_reset_link: <string value> | <boolean value> Whether the link to reset the Map to the initial view is displayed or not. Default value TRUE. If a string is provided instead of TRUE or true, then the string will be used as the link text instead of the default text.

STYLING THE MAP VIEW

Below you can see an example of the code generated by Newscoop to display a Map. You can style the way it looks like by defining CSS styles for the classes.

```
 <!-- Map Container START //-->
<div class="geomap_container">
  <!-- Map general info and
locations Container START //-->
  <div class="geomap_locations">
    <!-- Map General Info (Title)
START //-->
    <div class="geomap_info">
      <dl class="geomap_map_name">
        <dt
class="geomap_map_name_label">Map:</dt>
        <dd
class="geomap_map_name_value">My Map</dd>
      </dl>
    </div>
    <!-- Map General Info (Title)
END //-->
    <!-- Map Locations List START
//-->
```

```
20.     <div id="side_info"
21. class="geo_side_info">
22.       <!-- Map Location Nr. 1 START
23. //-->
24.       <div id="poi_seq_0">
25.         <a class="geomap_poi_name"
26. href="#"
27. onClick="geo_hook_on_map_feature_select(geo_object_61_1, 0);
28. return false;">POI no. 1</a>
29.         <div
30. class="geomap_poi_perex"></div>
31.         <div
32. class="geomap_poi_center">
33.           <a href="#"
34. onClick="geo_object_61_1.center_lonlat(14.753722843736,
35. 48.948841006863); return false;">Center</a>
36.         </div>
37.         <div
38. class="geomap_poi_spacer"> </div>
39.       </div>
40.       <!-- Map Location Nr. 1 END
41. //-->
42.       <!-- Map Location Nr. 2 START
43. //-->
44.       <div id="poi_seq_1">
45.         <a class="geomap_poi_name"
46. href="#"
47. onClick="geo_hook_on_map_feature_select(geo_object_61_1, 1);
48. return false;">POI Name</a>
49.         <div
50. class="geomap_poi_perex">Any text</div>
51.         <div
52. class="geomap_poi_center">          <a href="#"
53. onClick="geo_object_61_1.center_lonlat(1.240539250526,
54. 47.067502513872); return false;">Center</a>
55.         </div>
56.         <div
57. class="geomap_poi_spacer"> </div>
58.       </div>
59.       <!-- Map Location Nr. 2 END
60. //-->
61.     </div>
62.     <!-- Map Locations List END
63. //-->
64.   </div>
65.   <!-- Map general info and
66. locations Container START //-->
67.   <!-- Map Menu (Show Reset Link)
68. START //-->
69.   <div class="geomap_menu">
70.     <a href="#"
71. onClick="geo_object_61_1.map_showview(); return false;">Gimme
72. my map</a>
73.   </div>
74.   <!-- Map Menu (Show Reset Link)
75. END //-->
76.   <!-- Map Render Area START //-->
77.   <div class="geomap_map">
78.     <div
79. id="geo_map_mapcanvas_61_1"></div>
80.   </div>
81.   <!-- Map Render Area END //-->
82. </div>
83. <div style="clear:both"
84. ></div>
```

Sourcefabric provides some styling definitions as part of the sample templates.

91. LOCATION MULTIMEDIA

Note: this works only in Newscoop 3.5.0 and later versions.

Maps require jQuery : In order to use display maps you must include jQuery in the header of your document, with a link like this:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/
1/jquery.min.js">
```

The location multimedia object is usually initialized inside an Article Location object. It is not initialized at the beginning of the template and cannot be initialized by other Newscoop functions. The location multimedia object has the following properties:

- src: the media URL (for images), identifier (for YouTube, Vimeo videos), or file name (for flash, flv videos)
- type: can be "image" or "video"
- spec: for video type it can return the following: "youtube", "vimeo", "flash", "flv"; for image the empty string
- width: integer specifying the media frame width in pixels; the media is resized to this width (if non-zero)
- height: integer specifying the media frame height in pixels; the media is resized to this height (if non-zero)

92. GEOLOCATION SEARCH

The javascript/geocoding/map_search.js file has implemented a class for a map with the capability for the user to specify a search area. The area is selected via a half-transparent box. The output should be sent to the server where processed, as described below.

The search map can be used via the GetMapSearchHeader, GetMapSearchBody, and GetMapSearchCenter static methods of the Geo_Map class. They work in a similar way to the preview/tag functions. You can provide width and height values for the search map and optionally a list of four ids for automatic storage of top-left and bottom-right longitude/latitude values. The values can be taken from the class by calling the get_top_left and get_bottom_right methods of an instantiated object.

An example of the visual search interface is in the admin-files/articles/locations/search.php file.

The Geo_Map::GetGeoSearchSQLQuery takes two coordinates - opposite corners of the search area - and it returns a SQL statement for the database search. The two corners should go from West to East, otherwise it would not be known whether the box goes over the 180/-180 meridian or not. It is as shown below:

```
1.  // going east to west over the 180/-180, and south to north
2.  $p_coordinates = array();
3.  $p_coordinates[] = array("longitude" => "150", "latitude" => "20");
4.  $p_coordinates[] = array("longitude" => "40", "latitude" => "60");
5.
6.  // going directly west to east, and north to south
7.  $p_coordinates = array();
8.  $p_coordinates[] = array("longitude" => "-10", "latitude" => "60");
9.  $p_coordinates[] = array("longitude" => "40", "latitude" => "-20");
10.
11. $query = Geo_Map::GetGeoSearchSQLQuery($p_coordinates);
12. echo $query;
```

TEMPLATE REFERENCE - MODIFIERS

93. DATE AND E-MAIL FORMATTING
94. FILE SIZE FORMATTING
95. TRUNCATING UTF8 STRINGS
96. URL DISPLAY AND MODIFYING

93. DATE AND E-MAIL FORMATTING

DATE FORMATTING

The "camp_date_format" modifier formats a date string as specified:

```
{{ <string>|camp_date_format:"<date_attribute>" }}
```

```
{{ <string>|camp_date_format:"<date_format>" }}
```

Filters:

<date_attribute> may be one of the following:

- year: year (four digits)
- mon: month as a number (1..12)
- mday: day of the month as a number (1..31)
- yday: day of the year (1..366)
- wday: day of the week as a number (0=Sunday..6=Saturday)
- hour: hour (0..23)
- min: minute (two digits)
- sec: seconds (two digits)
- mon_name: name of the month
- wday_name: day of the week

<date_format> may contain any printable character; escape " (double quotes) in the date formatting with \ (backslash). For example: "%M %e, %Y, \"%W\"" will display the the date like this: July 5, 2008, "Saturday".

The following groups of characters have special meaning:

- %M - month name
- %W - weekday name
- %Y - year (4 digits)
- %y - year (2 digits)
- %m - month (01..12)
- %c - month (1..12)
- %w - day of the week (0=Sunday..6=Saturday)
- %d - day of the month (00..31)
- %e - day of the month (0..31)
- %j - day of the year (001..366)
- %D - day of the month with English suffix (1st, 2nd, 3rd etc.)
- %H - hour in format 00..23
- %h - hour in format 01..12
- %I - hour in format 1..12
- %i - minutes in format 00..59
- %S - seconds in format 00..59
- %s - is an alias of %S
- %p - AM or PM

Constraints:

None

EMAIL OBFUSCATION

The "obfuscate_email" modifier obfuscates a string; the email string is obfuscated to prevent spambot web crawlers from finding it.

```
{{ <email_string>|obfuscate_email }}
```

94. FILE SIZE FORMATTING

The "camp_filesize_format" modifier formats an integer in Byte specific multiples:

```
{{ <integer>|camp_filesize_format:"<filesize_format>" }}
```

<filesize_format> may take one of the following values:

- TB: for displaying the size in terabytes
- GB: for displaying the size in gigabytes
- MB: for displaying the size in megabytes
- KB: for displaying the size in kilobytes
- B: for displaying the size in bytes

95. TRUNCATING UTF8 STRINGS

The "truncate_utf8" modifier behaves exactly like the Smarty truncate modifier, except that it works properly on UTF8 strings. See more details on truncate here:

http://www.smarty.net/manual/en/language.modifier.truncate.php

96. URL DISPLAY AND MODIFYING

DISPLAYING THE URL

Purpose:

"uripath" prints only the path part of the URI, the part before the parameters list. If "/en/1/2/3?param1=text" was the full URI, the URI path is "/en/1/2/3".

"urlparameters" prints the runtime environment parameters in URL format.

"uri" prints the complete link URI and it is equivalent to:

```
{{ uripath }}?{{ urlparameters }}
```

"url" prints the complete URL in the form:

```
http://[site_alias][uri]
```

It is equivalent to:

```
http://{{ $gimme->publication->site }}{{ uri }}
```

Note:

- All four statements work for both template path and short URL types. Depending on the URL type, the statement displays the proper link
- For publications with short URL type the only way to create links is to use these three statements: either url, uri, uripath or urlparameters. It is not possible to build the URI manually

Syntax:

```
1. {{ url [options="<options_list>"] }}
2. {{ uri [options="<options_list>"] }}
3. {{ uripath [options="<options_list>"] }}
4. {{ urlparameters [options="<options_list>"] }}
```

Filters:

- <options_list>=
  - <option> <list_of_options>
  - <option>
- <option>=
  - fromstart
  - reset_issue_list
  - reset_section_list
  - reset_article_list
  - reset_searchresult_list
  - reset_subtitle_list
  - image <image_number> <image_options>
  - root_level
  - language
  - publication
  - issue
  - section
  - article
  - template <template_name>
  - articleAttachment
  - articleComment
  - audioAttachment
  - previous_subtitle <field_name>
  - next_subtitle <field_name>
  - all_subtitles <field_name>
  - previous_items
  - next_items

<image_number> is an <integer_value> representing a valid number of article images

<template_name> is a <string value> representing the full path of a valid template

- fromstart: prints the url corresponding to the received request, before any changes were made to the template runtime environment (useful when building site maps)
- reset_issue_list: reset list counters for the issues list
- reset_section_list: reset list counters for the sections list
- reset_article_list: reset list counters for the articles list
- reset_searchresult_list: reset list counters for the search results list
- reset_subtitle_list: reset list counters for the subtitles list
- image: print image parameters in order to show an image in a template; if you use this statement inside the "List of Article Images" statement the image number value is not mandatory. The image options are different depending on the Newscoop version

For Newscoop versions 3.0-3.3:

<image_options>=<image_ratio>.

The "image_ratio" parameter is an integer between 1 and 100. The image will be scaled from 1% of it's size to 100% based on the image ratio parameter.

For Newscoop versions 3.4 and newer:

<image_options>=<image_ratio>|width <width>|height <height>.

The "image_ratio" parameter is the same as above. You can specify an image to be resized to the given width or height.

E.g.: {{ uri options="image 1 width 100 height 100 }}

- root_level: this link will have all parameters reset: language, publication, issue, section, article, subtitle, lists, indexes etc.
- language: this link will have all parameters above language reset: publication, issue, section, article, subtitle, lists, indexes etc.; the language parameter remains unchanged
- publication: this link will have all parameters above publication reset: issue, section, article, subtitle, lists, indexes etc.; the language and publication parameters remain unchanged
- issue: this link will have all parameters above issue reset: section, article, subtitle, lists, indexes etc.; language, publication and issue parameters remain unchanged
- section: this link will have all parameters above section reset: article, subtitle, lists, indexes etc.; language, publication, issue and section parameters remain unchanged
- article: this link will have all parameters above article reset: article, subtitle, lists, indexes etc.; language, publication, issue, section and article parameters remain unchanged
- template <template_name>: this link will set the template in the link to <template_name>
- articleAttachment: displays the link for the current article attached document
- articleComment: inserts the article comment identifier in the parameters list; if no article comment was defined in the template environment, no parameter is inserted. This attribute is needed if you want to pass on the current comment id to another page for display purposes
- audioAttachment: displays the link for the current audio file attached to the article
- previous_subtitle <field_name>: add a parameter to the URI to select the previous subtitle of the given <field_name> for display
- next_subtitle <field_name>: add a parameter to the URI to select the next subtitle of the given <field_name> for display
- all_subtitles <field_name>: add a parameter to the URI to select all subtitles of the given <field_name> for display
- previous_items: add a parameter to the URI to select the previous interval of the current list items for display
- next_items: add a parameter to the URI to select the next interval of the current list items for display

Constraints:

None.

Examples:

For the canonical tag often used for SEO (search engine optimization) you should use...

- on article level

```
<link rel="canonical" href="{{ url options="article" }}" />
```

- on section level

```
<link rel="canonical" href="{{ url options="section" }}" />
```

- on front page / index level

```
<link rel="canonical" href="http://{{ $gimme->publication->site }}" />
```

APPENDIX

97. CREDITS AND LICENSE

97. CREDITS AND LICENSE

Newscoop is being developed by an international community of developers and designers led by Sourcefabric.

http://newscoop.sourcefabric.org

LICENSE

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS