TABLE OF CONTENTS

INTRODUCTION

1. READY TO COOK?

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

Welcome to the Newscoop 4 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, and social media integration. The powerful template engine of Newscoop allows you to create any text formats from your database - HTML, XML, Javascript, you name it. This makes it very attractive for contemporary web developers, because it goes beyond "making HTML" and is by nature HTML5 ready. You can find out more about Newscoop 4 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 publication up and running with Newscoop.

The Newscoop 4 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 by using one of the sample themes from the Sourcefabric website. It's possible to adjust these pre-made themes to your needs. In the longer 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/en/community/news/

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. ABOUT THIS BOOK

Updated to Newscoop 4.2.0

The Newscoop 4 Cookbook is a work in progress and open to your input. Dedicated to Open Source software development, Sourcefabric makes both the software and its documentation 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 wider use of software.

If you experience the Newscoop 4 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 4 Cookbook online at http://manuals.sourcefabric.org, 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 Booktype, the content of this book is continuously evolving. Chances are, the latest PDF, ePub or the online version on the site http://manuals.sourcefabric.org contains more details than the printed copy.

All work, no fun at the Book Sprint in 2011 :)

Sourcefabric first started this publication for Newscoop 3 with a one-week Book Sprint in Schloss Neuhausen, Germany in 2011, 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.

Trying to squeeze in the documentation work during office hours in 2013

In 2013 we upgraded the Newscoop 4 Cookbook in Berlin. A lot of examples had to be re-written and re-tested since Newscoop version 4 was a new and improved beast. Present were (in reverse alphabetical order) Tomasz Rondio, Ljuba Rankovic, Paweł Mikołajczuk, Pete Haughie, and Micz Flor with the remote support of Daniel James.

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

3. 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.

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

The Sourcefabric team at our yearly "Sourcecamp" in 2011

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, and Cluj.

The Sourcefabric founders 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.

WHY 'SOURCEFABRIC'?

Both the journalist's 'source' (the origin of information) and the programmer's 'source' (the code on which software is built) are vital to the work we do.

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.

PLANNING YOUR PUBLICATION

4. 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 server, 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

Which 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 a contractor handling 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 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 labour involved.

5. 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 *Publishing Articles* in the manual Newscoop 4 for Journalists and Editors.

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 news 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 Newscoop publication is like a living organism; you'll have to look after it.

6. 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 theme "The New Custodian" which reflects magazine structure. It 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 New Custodian" can be seen at http://newscoop-demo.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

7. $GIMME ALL YOUR LOVIN'

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 4 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:

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

{{ $gimme->article->comment->subject }}

{{ $gimme->publication->default_language }}

{{ $gimme->author->name }}

{{ $gimme->author->biography->text }}

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

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 <em> 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, colour 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:

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

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

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

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

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

```
<div class="wrapper">
    <h2 class="title section{{ $gimme->section->number }}">
       {{ $gimme->article->name }}</h2>
         <span>{{ $gimme->article->subtitle }}</span>
           <div class="section">{{ $gimme->section->name }}</div>
</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 publication 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 towards 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.

```
<ul>
{{ list_article_authors }}
    <li>
```

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 }}" />
```

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

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

When you are done, close this list. Thanks.

```
    </li>
{{ /list_article_authors }}
</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:

```
Location(s):
{{ list_article_locations }}
    {{ if $gimme->location->enabled }}
        <p>
        {{ $gimme->location->name }}<br />
        {{ $gimme->location->longitude }}, {{ $gimme->location->latitude }}
        </p>
    {{ /if }}
{{ /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 each 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.

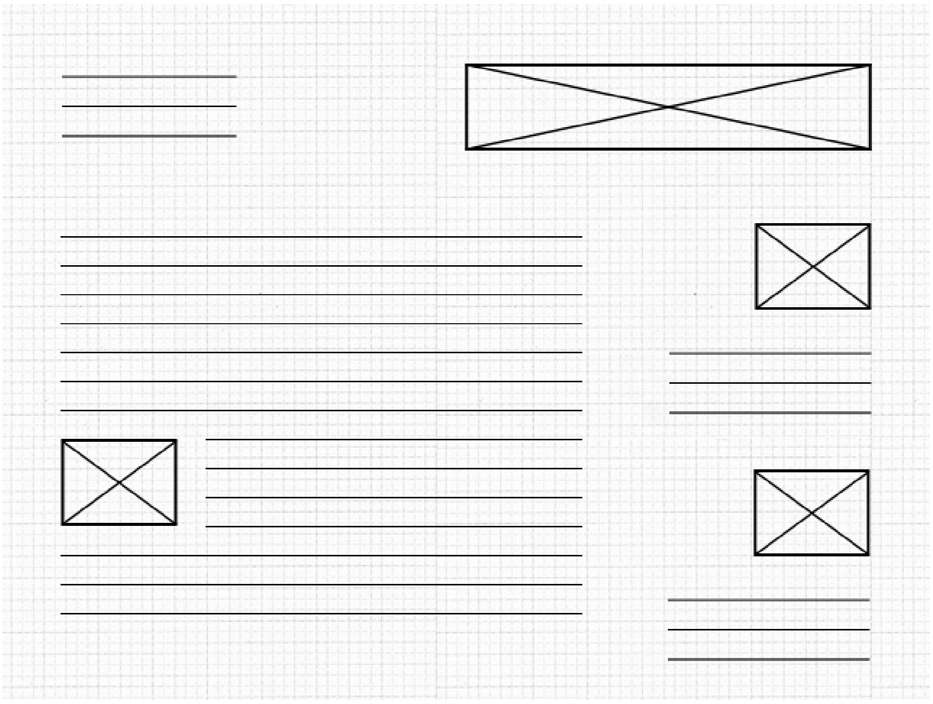
Once you understand the power of $gimme, you will start getting a sense of the freedom it offers you when building websites. Enjoy!

8. 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 definition terms, 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 publication'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 the conversation of the community behind your publication. 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.

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.

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. This ratio between line height and font size is close to the golden ratio.

Using too many font faces might have major impact on the professionalism a publication conveys, whereas 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.

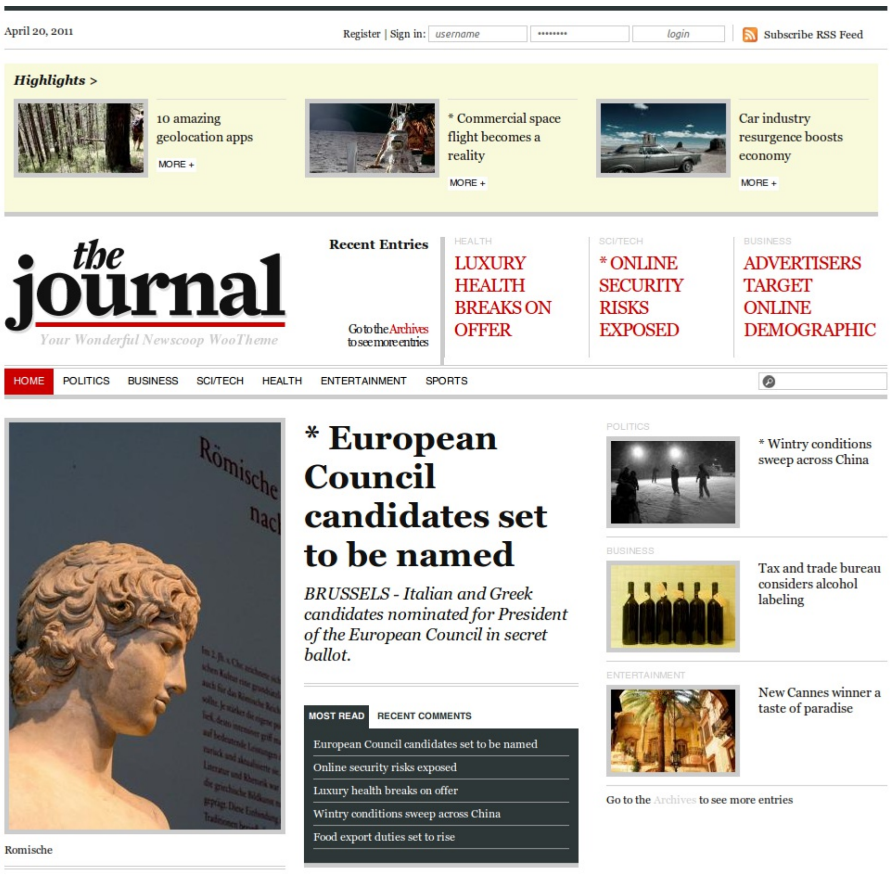
```
<link href="http://fonts.googleapis.com/css?family=Droid+Sans" rel="stylesheet"
type="text/css" />
```

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. Another very good resource for free and open source fonts is FontSquirrel.com

9. 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 the 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 Newscoop theme "The Journal"...



April 20, 2011

Register | Sign in: username •••••••• login Subscribe RSS Feed

Highlights >

10 amazing geolocation apps MORE +

* Commercial space flight becomes a reality MORE +

Car industry resurgence boosts economy MORE +

the journal

Your Wonderful Newscoop WooTheme

Recent Entries

Go to the Archives to see more entries

HEALTH LUXURY HEALTH BREAKS ON OFFER

SCI/TECH * ONLINE SECURITY RISKS EXPOSED

BUSINESS ADVERTISERS TARGET ONLINE DEMOGRAPHIC

HOME POLITICS BUSINESS SCI/TECH HEALTH ENTERTAINMENT SPORTS

Romische

* European Council candidates set to be named

BRUSSELS - Italian and Greek candidates nominated for President of the European Council in secret ballot.

MOST READ RECENT COMMENTS

European Council candidates set to be named

Online security risks exposed

Luxury health breaks on offer

Wintry conditions sweep across China

Food export duties set to rise

POLITICS * Wintry conditions sweep across China

BUSINESS Tax and trade bureau considers alcohol labeling

ENTERTAINMENT New Cannes winner a taste of paradise

Go to the Archives to see more entries

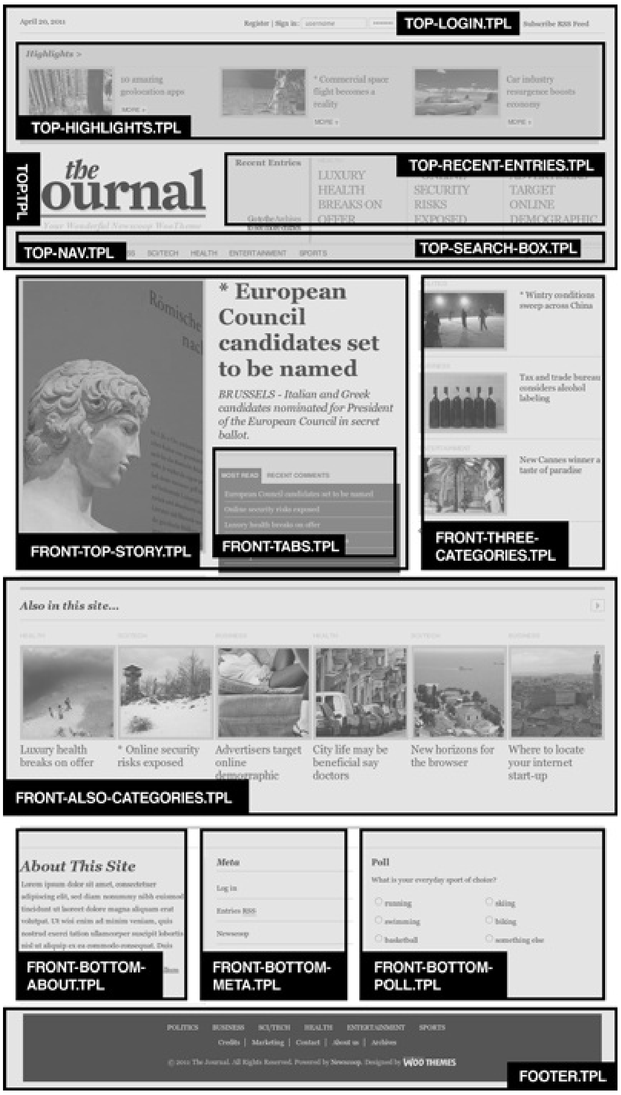
Front page of "The Journal".

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 <div> tags. A <div> 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 <div> tags that are present only once in your site pages (e.g. navigation, sidebar, footer, etc.). You should use classes for <div> 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:

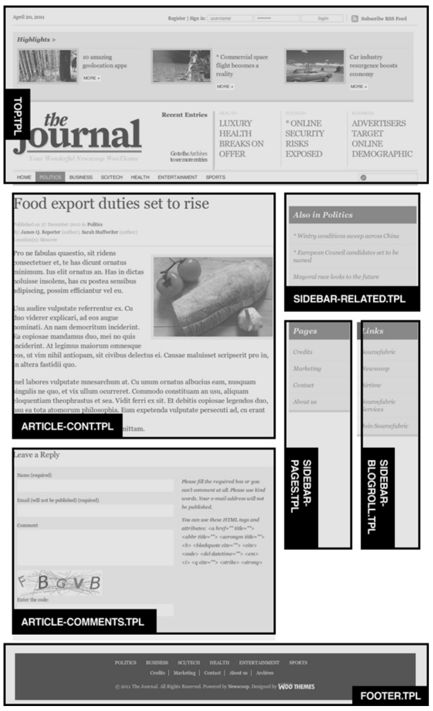


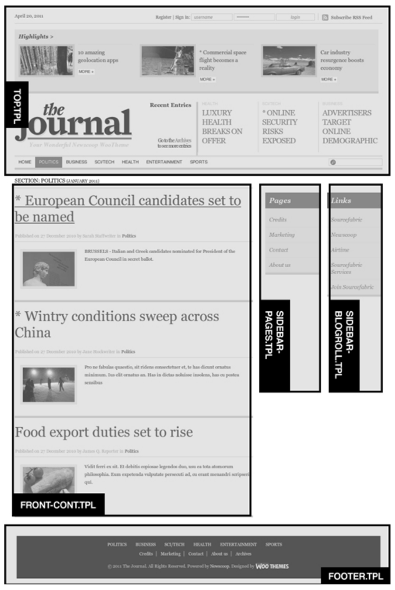
Your HTML files will end up being .tpl files, which are Newscoop template files. For "The Journal", the front page template **front.tpl** is a skeleton which includes and positions the following sub-templates:

- **_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.





10. ORGANISING TEMPLATE FILES

A template is a single file with the .tpl ending. Usually, a number of .tpl files plus Javascript, CSS and other files are combined into a theme. This Cookbook is intended to help you to create or modify templates and bundle them into a unique theme for your publication. Inside the theme, the individual template files are structured in folders and sub-folders. While there is a certain freedom about doing this, we encourage you to follow the "best practice" that we describe here.

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. The themes that 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 theme to another. Read on to find out what we recommend when it comes to naming templates and placing them in your file system.

Every theme available for download from Sourcefabric has four main templates:

- Front page
- Section page
- Article page
- 404 page

These are called - no surprise here - front.tpl, section.tpl, article.tpl and 404.tpl.

You could run a publication with these four 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="_tpl/article-comments.tpl" }}
```

As you can see in this example, the path to the template is relative. Inside a theme, all subfolders can be reached as relative paths. A typical part of a template might look like the following piece of code. This example is taken from the theme "Quetzal":

```
<section id="content">

  <div>

  {{ if $gimme->article->type_name == "debate" }}
    {{ include file="_tpl/article-debate.tpl" }}
  {{ else }}
    {{ include file="_tpl/article-cont.tpl" }}
  {{ /if }}

  {{ include file="_tpl/article-aside.tpl" }}

  </div> <!--end div class="row"-->

  {{ include file="_tpl/tablet-more-tabs.tpl" }}

</section> <!-- end section id=content -->
```

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. Includes that are used on many pages can have short names, like header.tpl, main-nav.tpl, footer.tpl
3. 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.

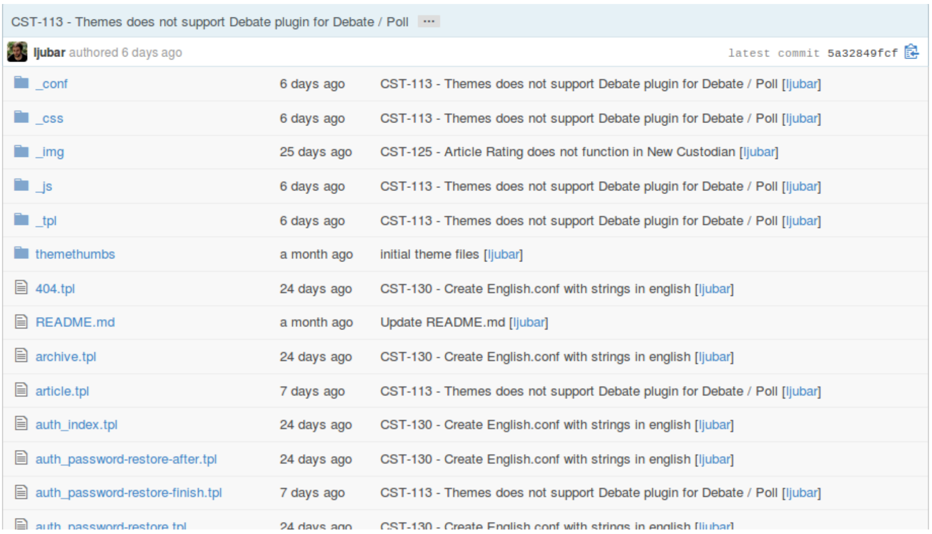
4. Special templates that are not directly used to output content on the web page but to define context (for example the _html-head.tpl which defines the HEAD section of web pages, or some RSS or sitemap generators) can have the '_' (underscore) sign as the first character; this way, all such templates are grouped together at the top of the template listing page.

FOLDER STRUCTURE

Let's take a look at a live Newscoop theme "The New Custodian" developed by Ljuba Rancovic:

https://github.com/newscoop/theme-NewCustodian

Inside this theme, files are structured inside folders. On the top level, you can only see a smaller number of template files, which call more detailed code from template files inside the _tpl folder.



CST-113 - Themes does not support Debate plugin for Debate / Poll ···

ljubar authored 6 days ago — latest commit 5a32849fcf

_conf	6 days ago	CST-113 - Themes does not support Debate plugin for Debate / Poll [ljubar]
_css	6 days ago	CST-113 - Themes does not support Debate plugin for Debate / Poll [ljubar]
_img	25 days ago	CST-125 - Article Rating does not function in New Custodian [ljubar]
_js	6 days ago	CST-113 - Themes does not support Debate plugin for Debate / Poll [ljubar]
_tpl	6 days ago	CST-113 - Themes does not support Debate plugin for Debate / Poll [ljubar]
themethumbs	a month ago	initial theme files [ljubar]
404.tpl	24 days ago	CST-130 - Create English.conf with strings in english [ljubar]
README.md	a month ago	Update README.md [ljubar]
archive.tpl	24 days ago	CST-130 - Create English.conf with strings in english [ljubar]
article.tpl	7 days ago	CST-113 - Themes does not support Debate plugin for Debate / Poll [ljubar]
auth_index.tpl	24 days ago	CST-130 - Create English.conf with strings in english [ljubar]
auth_password-restore-after.tpl	24 days ago	CST-130 - Create English.conf with strings in english [ljubar]
auth_password-restore-finish.tpl	7 days ago	CST-113 - Themes does not support Debate plugin for Debate / Poll [ljubar]
auth_password-restore.tpl	24 days ago	CST-130 - Create English.conf with strings in english [ljubar]

From the sub-folders you see in this screenshot only one name is fixed: **_conf**. The others could be named any way you want. But to make things easier for other theme developers, better stick to the suggested structure. Here are the folders explained:

- **_conf**
  This optional folder can contain configuration files for the template engine. Here you can also place the conf files for multilingual sites, as described in the *Multilingual publications and templates* section of this Cookbook.
- **_css**
  Place all you .css files here. In some cases this folder might also contain .less or other markdown files which make it easier to customise existing themes. Feel free to place other folders in side this one, e.g. "bootstrap".
- **_img**
  This is for static images and sometimes other assets. Most important here: logo.png - changing this is the first step to customisation :)
- **_js**
  This is for javascript files and can also contain subfolders.
- **_tpl**
  Inside this folder you can place all the template files which can then be called with {{ include ... }}. Again, to structure your files, feel free to use subfolders in here.
- **themethumbs**
  This folder contains three thumbnail images which are displayed in the Configure -> Themes administration page of Newscoop. In this case they are called article.jpg, section.jpg and front.jpg. They serve to give the user an idea of what the theme looks like before the user applies the theme to a specific publication.

If you check out Sourcefabric 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.

INCLUDING TEMPLATES FROM TEMPLATES

In order to call a template from within a template, you use the **include** command. Here are some examples of including static files:

```
<link rel="shortcut icon" href="{{ url static_file='_img/favicon.ico' }}" />

<link rel="stylesheet" href="{{ url static_file='_css/ui-lightness/jquery-
ui.custom.css' }}" />

<script src="{{ url static_file='_js/libs/jquery.min.js' }}"></script>

<link href="{{ url static_file="_css/video-js.css" }}" rel="stylesheet" />
```

And here are some examples of how to include template files:

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

<aside class="fourcol last">{{ include file="_tpl/article-aside.tpl" }}</aside>
```

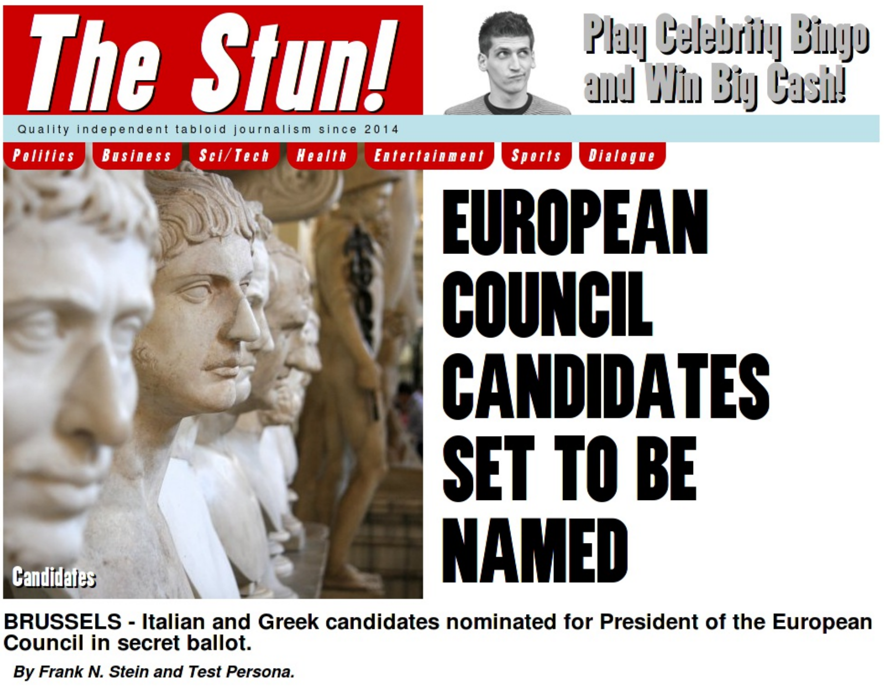
The <aside> tag is new in HTML5.

11. YOUR FIRST NEWSCOOP THEME

If you're a designer who is new to the Newscoop or Smarty way of making web pages, or you've been trained on GUI-based web development tools, the trend towards template-driven design might seem intimidating. Perhaps you don't have a computer science background, and now you're being expected to understand programming as well as design. Fear not, because Newscoop's template language is just as simple as HTML, and you'll find it a huge time-saver once you have grasped the basics. No longer will you be forced to adopt ready-made themes for your next content management system deployment, or worse still, have to dive into unfamiliar code to make the adaptations your client requires.

To prove just how simple Newscoop publication design can be, take a look at this theme, The Stun, available for free download under the GNU GPLv3 license from: https://github.com/newscoop/theme-TheStun/releases/download/v1.3/theme-TheStun.zip

This theme is also available on GitHub at https://github.com/newscoop/theme-TheStun for examination and forking. (Please note that the 'Download this repository as a zip file' button on GitHub exports Newscoop theme files within a folder, whereas a working Newscoop theme should have its main template files in the zip file root. See the Cookbook chapter 'Creating a Newscoop theme' for details of the proper file structure).


The Stun!
Play Celebrity Bingo and Win Big Cash!
Quality independent tabloid journalism since 2014
Politics
Business
Sci/Tech
Health
Entertainment
Sports
Dialogue
Candidates

EUROPEAN COUNCIL CANDIDATES SET TO BE NAMED

BRUSSELS - Italian and Greek candidates nominated for President of the European Council in secret ballot.

By Frank N. Stein and Test Persona.

The Stun theme contains just two CSS files (one for design, one for fonts) and the bare minimum of features needed to display the front page, sections and articles of your publication. The design of The Stun is by Pete Bradley, former News International page designer.

The idea behind this theme is to make it easier for new designers to learn how a Newscoop theme works, compared to looking at the code of full-feature themes like New Custodian or Rockstar. It also contains an example of how to use a web font from http://www.fontsquirrel.com in a Newscoop theme. Finally, news designers are freed from the stultifying tyranny of web-safe fonts!

Let's take a look at The Stun's front page template, front.tpl - just 33 lines of code plus whitespace, and the biggest block of that code is for creating the list of article authors with the proper punctuation (given that we can't guess how many authors there might be for a dynamically generated article).

This template starts by including _header.tpl from the _tpl folder of sub-templates (there are just three sub-templates in The Stun). Thanks to this sub-template we don't have to deal with the <HEAD> element of HTML, open a <BODY> tag, fiddle with the masthead or deal with navigation buttons. It's all taken care of, so we can focus on front page design.

```
{{ include file='_tpl/_header.tpl' }}
```

Next, we'll list articles from the "Front page" featured article playlist, a sequence of the day's most significant stories which has been created by the newspaper's editor in the Newscoop administration interface. All we have to do is tell Newscoop to list the article that the editor has made the top item in this particular playlist, by using a list length of "1":

```
{{ list_playlist_articles name="Front page" length="1" }}
```

The next part of the code is for the article headline, which we put in a <div> because we want to style it with CSS. The id of this div should make its function pretty obvious when we are editing the general.css file, found in the _css folder.

```
<div id="frontpage-headline">
  <a href="{{ uri options="article" }}"><h1>{{ $gimme->article->name
}}</h1></a>
</div>
```

See how we're using two sets of curly brackets on the line above? Within the element created by the <h1> and </h1> tags we are asking Newscoop for the article name to create our headline, which could be any line of text the editor wants to put there at that moment. We're also wrapping the <h1> element with an <a> element, but this time we're using the curly brackets to get the URI of this article. So we now have a styled headline, and a link to the article inside the publication.

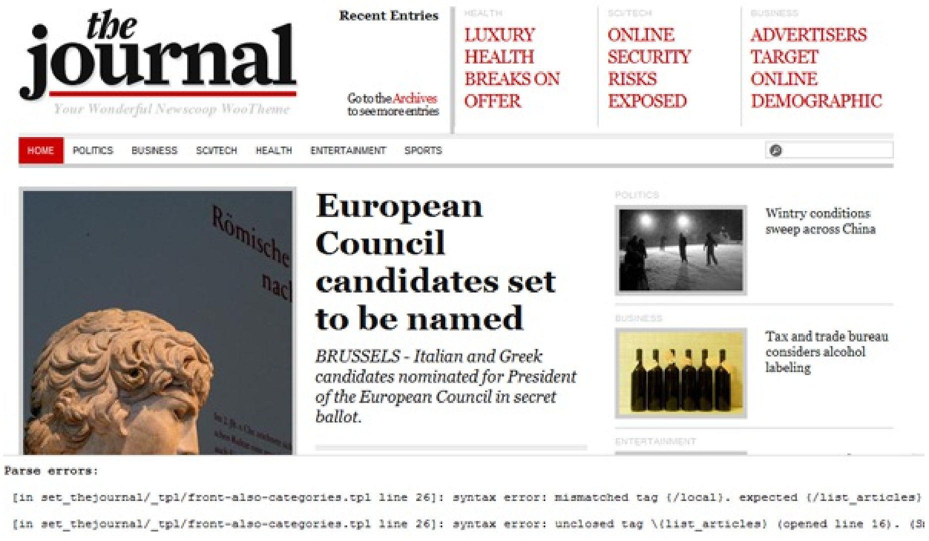
Typically we may wish to provide a shortened version of the story on the front page, and give the full article a different design treatment. This is very easy to do in Newscoop since we have a separate template for full articles, by default called article.tpl (although you can call this file anything you want). There is also a template for listing articles in the sections that the editor has created, such as Politics or Sports. The section template is called section.tpl by default, and just as with front.tpl or article.tpl you can re-use the header, footer or any other sub-template that you have created. You'll soon appreciate how much maintenance time is saved by the efficiency of this approach.

In The Stun, 404.tpl is an empty file. You can put any message to readers that you want in this template, but with an empty 404 template Newscoop will re-direct all requests for non-existent pages to the publication front page, which may be what you wanted anyway.

We hope you have fun playing with The Stun, and if you create new versions of it, please let us take a look at what you've come up with!

12. 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.


the journal
Your Wonderful Newscoop WooTheme
Recent Entries
Go to the Archives to see more entries
HEALTH
LUXURY HEALTH BREAKS ON OFFER
SCI/TECH
ONLINE SECURITY RISKS EXPOSED
BUSINESS
ADVERTISERS TARGET ONLINE DEMOGRAPHIC
HOME
POLITICS
BUSINESS
SCI/TECH
HEALTH
ENTERTAINMENT
SPORTS
European Council candidates set to be named
BRUSSELS - Italian and Greek candidates nominated for President of the European Council in secret ballot.
POLITICS
Wintry conditions sweep across China
BUSINESS
Tax and trade bureau considers alcohol labeling
ENTERTAINMENT

```
Parse errors:

 [in set_thejournal/_tpl/front-also-categories.tpl line 26]: syntax error: mismatched tag {/local}. expected {/list_articles}

 [in set_thejournal/_tpl/front-also-categories.tpl line 26]: syntax error: unclosed tag \{list_articles} (opened line 16). (S
```

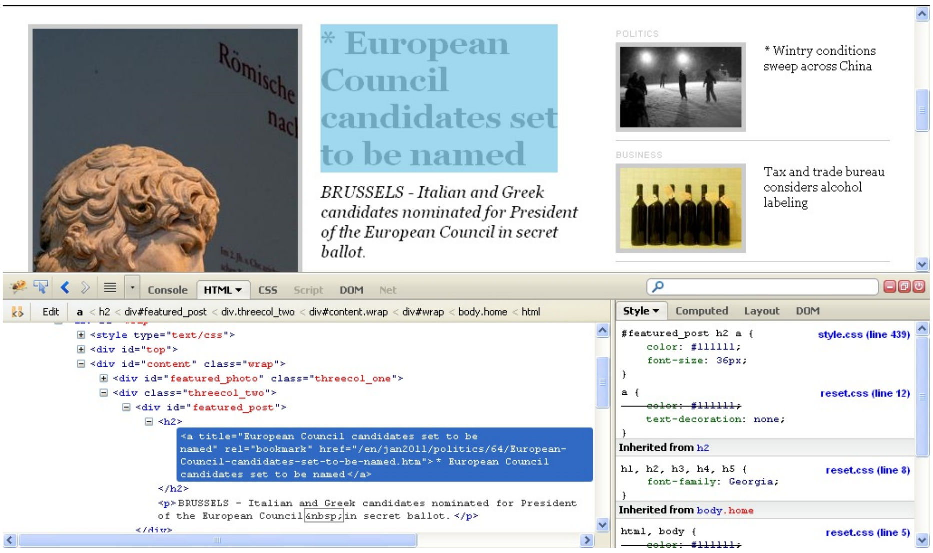
At the bottom of the preview window you can see error messages from the template parser.

The lower 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 <div> and <li> tags that it should but their content is empty, you can be sure that Newscoop isn't working like it should.

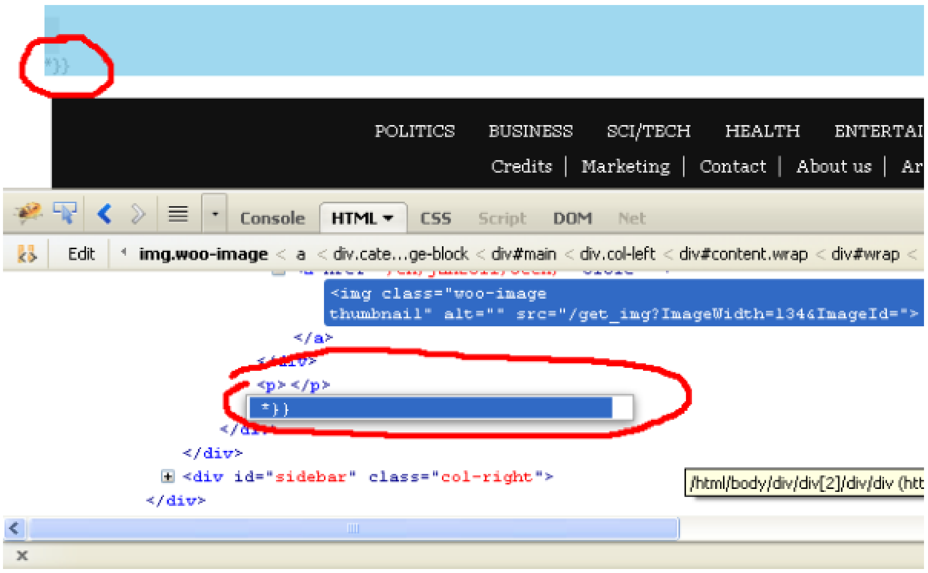
There are lots of very handy tools out there in browsers nowadays. You can use them to easily parse your HTML code and find an error. For example, Firebug (from http://getfirebug.com) is an extension to Firefox. All you need to do is search for it in the Firefox Add-ons Manager and click to install it. Chrome, Safari, Firefox, Opera, and even Internet Explorer all come with their own individual suites of powerful developer tools built in now.


* European Council candidates set to be named
BRUSSELS - Italian and Greek candidates nominated for President of the European Council in secret ballot.
POLITICS
* Wintry conditions sweep across China
BUSINESS
Tax and trade bureau considers alcohol labeling
Console
HTML
CSS
Script
DOM
Net
Edit
a < h2 < div#featured_post < div.threecol_two < div#content.wrap < div#wrap < body.home < html
Style
Computed
Layout
DOM

```
<style type="text/css">
<div id="top">
<div id="content" class="wrap">
  <div id="featured_photo" class="threecol_one">
  <div class="threecol_two">
    <div id="featured_post">
      <h2>
        <a title="European Council candidates set to be
        named" rel="bookmark" href="/en/jan2011/politics/64/European-
        Council-candidates-set-to-be-named.htm">* European Council
        candidates set to be named</a>
      </h2>
      <p>BRUSSELS - Italian and Greek candidates nominated for President
      of the European Council&nbsp;in secret ballot.</p>
    </div>
```

```
#featured_post h2 a {                 style.css (line 439)
    color: #111111;
    font-size: 36px;
}
a {                                   reset.css (line 12)
    color: #111111;
    text-decoration: none;
}
Inherited from h2
h1, h2, h3, h4, h5 {                  reset.css (line 8)
    font-family: Georgia;
}
Inherited from body.home
html, body {                          reset.css (line 5)
    color: #111111;
```

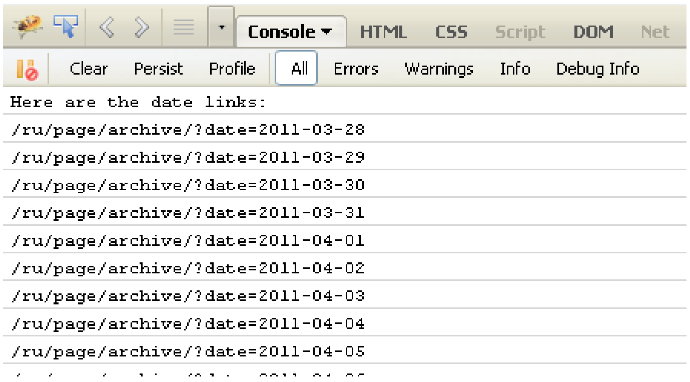
The usage is pretty easy - open, inspect element, find error:


*}}
POLITICS
BUSINESS
SCI/TECH
HEALTH
ENTERTAI
Credits
Marketing
Contact
About us
Ar
Console
HTML
CSS
Script
DOM
Net
Edit
img.woo-image < a < div.cate...ge-block < div#main < div.col-left < div#content.wrap < div#wrap <
/html/body/div/div[2]/div/div (htt

```
          <img class="woo-image
          thumbnail" alt="" src="/get_img?ImageWidth=134&ImageId=">
        </a>
      </div>
      <p></p>
      *}}
    </div>
  </div>
  <div id="sidebar" class="col-right">
</div>
```

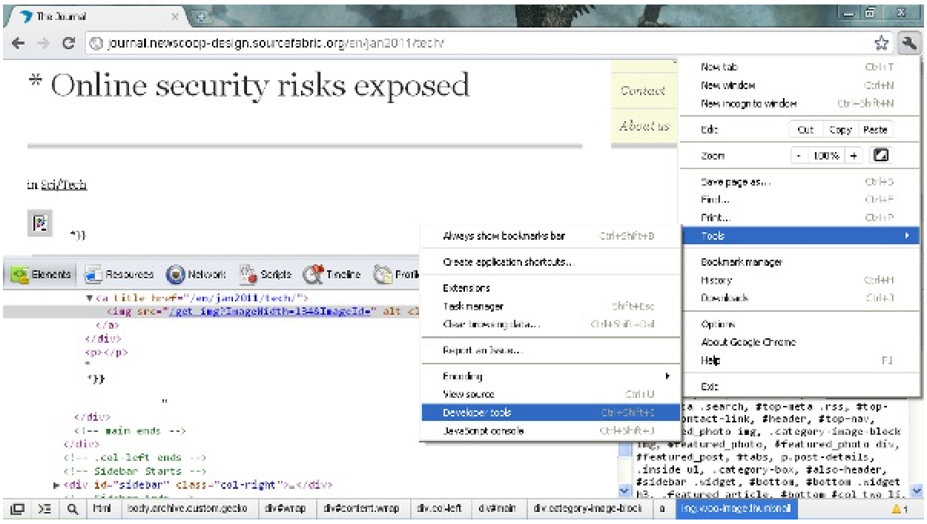
But what is really good is that you can debug your jQuery templates with it. Just adding a few lines of code to your JavaScript will provide Console feedback:

```
console.log ("Here are the date links:\r\n");
...
console.log (link + "\r\n");
```



```
Here are the date links:
/ru/page/archive/?date=2011-03-28
/ru/page/archive/?date=2011-03-29
/ru/page/archive/?date=2011-03-30
/ru/page/archive/?date=2011-03-31
/ru/page/archive/?date=2011-04-01
/ru/page/archive/?date=2011-04-02
/ru/page/archive/?date=2011-04-03
/ru/page/archive/?date=2011-04-04
/ru/page/archive/?date=2011-04-05
```

There's almost the same tool in Chrome, which can be found in the menu under Tools -> Developer tools, or simply right-click on the selection and choose 'Inspect Element':



```
▼<a title href="/en/jan2011/tech/">
    <img src="/get_img?ImageWidth=134&ImageId=" alt <l
  </a>
</div>
<p></p>
"
*}}
            "
</div>
<!-- main ends -->
</div>
<!-- .col-left ends -->
<!-- Sidebar Starts -->
▶<div id="sidebar" class="col-right">…</div>
```

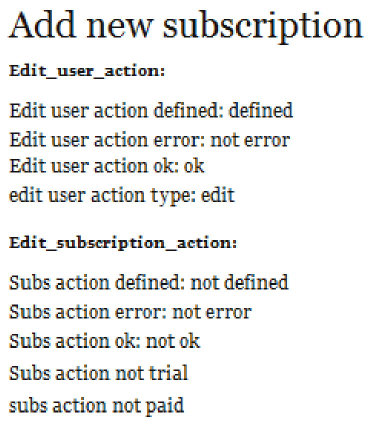
```
ta .search, #top-meta .rss, #top-
ontact-link, #header, #top-nav,
ed_photo img, .category-image-block
img, #featured_photo, #featured_photo div,
#featured_post, #tabs, p.post-details,
.inside ul, .category-box, #also-header,
#sidebar .widget, #bottom, #bottom .widget
h3, .featured article, #bottom #col two li,
```

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 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

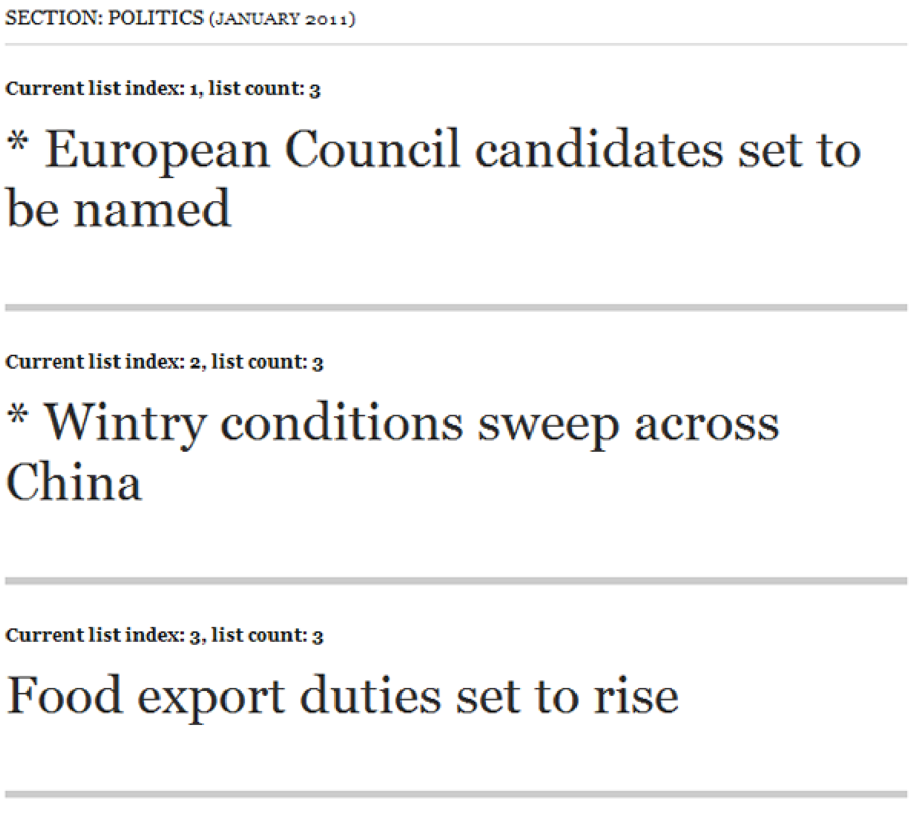
Edit_subscription_action:

Subs action defined: not defined
Subs action error: not error
Subs action ok: not ok
Subs action not trial
subs action not paid

Similarly, 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 iteration:-

```
{{ list_articles }}
    <div class="post wrap">
        <p>Current list index: {{ $gimme->current_list->index }}, list count:
{{ $gimme->current_list->count }}</p>
        <h2 class="post-title"><a href="{{ uri options="article" }}"
rel="bookmark" title="{{ $gimme->article->name }}">{{ if ! $gimme->article-
>content_accessible }}* {{ /if }}{{ $gimme->article->name }}</a></h2>
    </div>
{{ /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

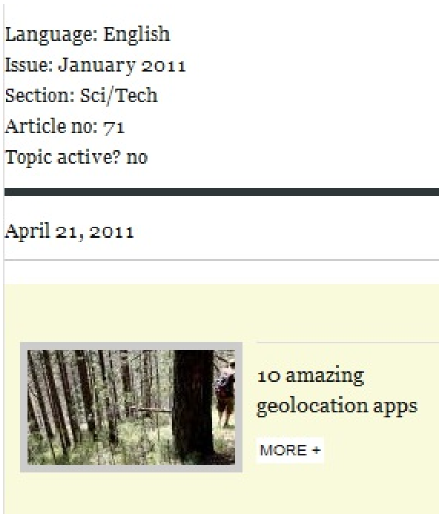
Current list index: 3, list count: 3

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:

```
<p>Language: {{ $gimme->language->name }}<br />
 Issue: {{ $gimme->issue->name }} <br />
 Section: {{ $gimme->section->name }}<br />
 Article no: {{ $gimme->article->number }}<br />
 Topic active? {{ if $gimme->topic->defined }}yes{{ else }}no{{ /if }}
</p>

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

The output displays the values of the parameters, so you can check if everything is OK:



Language: English
Issue: January 2011
Section: Sci/Tech
Article no: 71
Topic active? no

April 21, 2011

10 amazing geolocation apps

MORE +

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 nesting '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 value 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 clients. 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://www.sourcefabric.org/en/newscoop/newscooppro/

MAKING YOUR TEMPLATES

13. TIPS AND TRICKS TO SAVE YOU TIME

Besides the power of $gimme, at times you need some little extra help to display the information from your publication the way you want it. We compiled some answers to questions like "How do I get rid off the HTML?" or "I want to only display the first 40 characters, how do I chop them off?" or "How do I escape special characters so I can use the string inside Javascript?" and even "How do I get rid of line breaks, so my Javascript doesn't break?".

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 a path relative to the theme's zip file root:

```
{{ include file="_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 2013, 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 HTML content

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

Truncate string to specific length

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

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

```
{{ $gimme->article->full_text|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 &raquo; &laquo; &rdquo; and so on with &quot;

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

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

```
{{ $gimme->article->full_text|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 are 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 }}" >
```

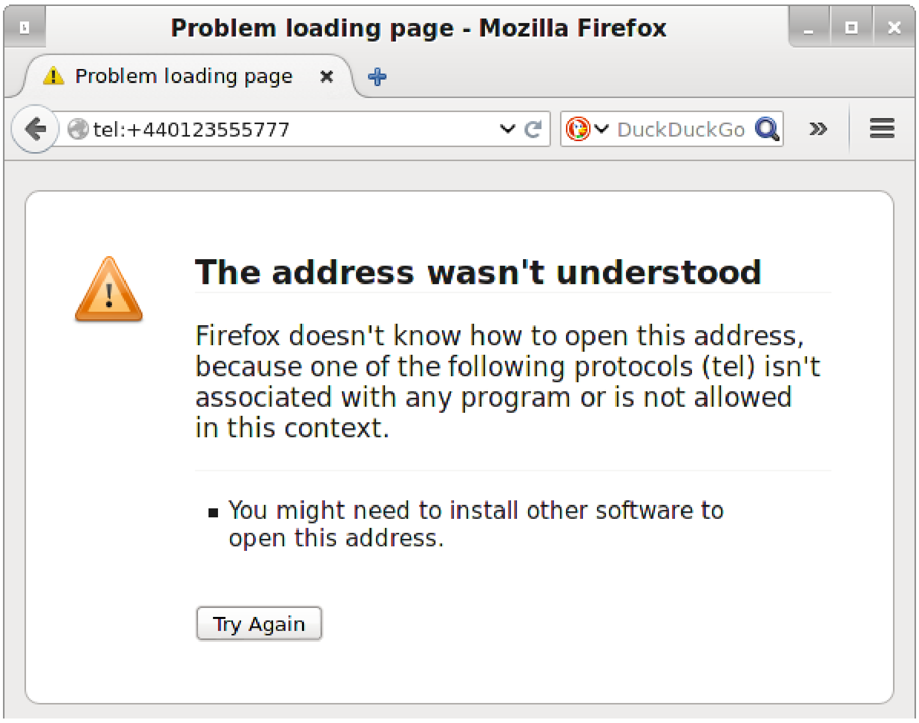
CREATING METATAGS FOR FACEBOOK SHARING

```
{{ if $gimme->article->defined }}{{* Open Graph protocol metatags for Facebook
sharing *}}

<meta property="og:title" content="{{$gimme->article-
>name|html_entity_decode|strip|escape:'html':'utf-
8'|regex_replace:'/&(.*?)quo;/':'&quot;'}}" />
<meta property="og:type" content="article" />
<meta property="og:url" content="http://{{ $gimme->publication->site }}{{ uri
}}" />
<meta property="og:site_name" content="{{ $gimme->publication->name }}" />

<meta property="og:description" content="{{$gimme->article-
>deck|strip_tags:false|trim|escape:'html':'utf-8' }}
{{if !$gimme->article->deck}}{{$gimme->article-
>full_text|strip_tags:false|trim|escape:'html':'utf-8' }}{{/if}}" />

{{ list_article_images }}
<meta property="og:image" content="{{ $gimme->article->image->imageurl }}" />
{{ /list_article_images }}

{{ /if }}
```

HANDLING PHONE NUMBERS IN LINKS

If phone numbers are formatted as links, spaces in the number between the <a> and </a> tags may enhance readability, but RFC 3966 says that spaces are not allowed in the URI which follows the *tel:* protocol. Newscoop can remove spaces, dashes and other non-allowed characters from the phone number URI, by using the *replace* or *regex_replace* variable modifiers.

Desktop browsers cannot be relied upon to have a phone application available or properly configured to handle the tel: protocol, which could give the appearance of a broken or malformed link.



The address wasn't understood

Firefox doesn't know how to open this address, because one of the following protocols (tel) isn't associated with any program or is not allowed in this context.

- You might need to install other software to open this address.

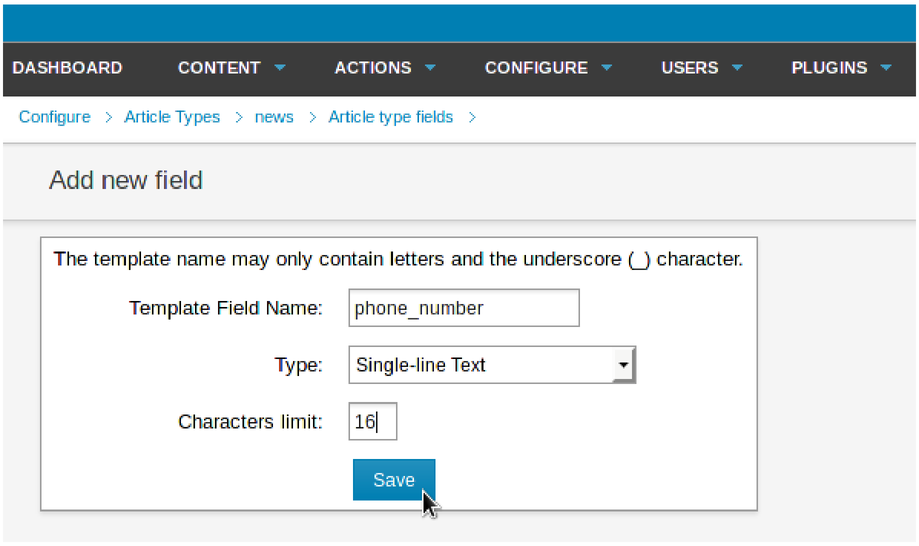
If we are targetting clickable phone numbers towards mobile browsers, we can serve a plain phone number, without a *tel:* protocol link, when the browser is detected as a desktop machine:

```
{{ if $gimme->browser->ua_type == "mobile" }}

  <a href="tel:{{ $gimme->article->phone_number|replace:' ':''|replace:'-':''
}}">
    {{ $gimme->article->phone_number }}
  </a>

{{ else }}

    {{ $gimme->article->phone_number }}

{{ /if }}
```

If a mobile browser was detected, the code above would render a phone number entered as *+44 0123 555-777* by the editor like so:

```
<a href="tel:+440123555777"> +44 0123 555-777 </a>
```

Another problem with phone URIs is that editors can attempt to put multiple phone numbers in one field. We can guard against that possibility in the Article Type by setting a character limit for a single-line text field. See 'Adding a new article type' in http://sourcefabric.booktype.pro/newscoop-42-for-journalists-and-editors/article-types/



Add new field

The template name may only contain letters and the underscore (_) character.

Template Field Name:	phone_number
Type:	Single-line Text
Characters limit:	16

14. PUBLICATIONS

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.

You can set a range of configuration parameters when creating or editing a publication in the Newscoop 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 }}
```

Default language

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

```
{{ $gimme->publication->default_language->name }} is the language name

{{ $gimme->publication->default_language->number }} is language identifier in
the Newscoop database (integer value)

{{ $gimme->publication->default_language->english_name }} is the language name
in English

{{ $gimme->publication->default_language->code }} is the language's
international code

{{ $gimme->publication->default_language->defined }} is a boolean value
(true/false) - true if the language was set in the current environment; false
otherwise
```

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 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->moderated_comments }}MODERATED{{ /if }}
```

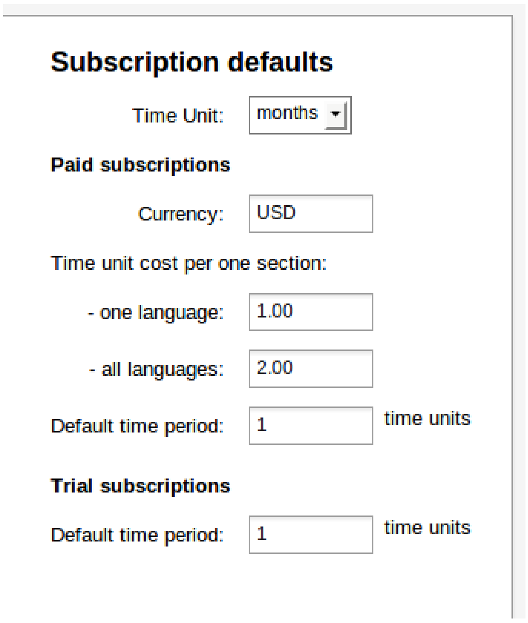
Use CAPTCHA to prevent spam? The reader must type in random letters or numbers shown before they can post a comment

```
{{ 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.



Subscription defaults

Time Unit: months

Paid subscriptions

Currency: USD

Time unit cost per one section:

- one language: 1.00

- all languages: 2.00

Default time period: 1 time units

Trial subscriptions

Default time period: 1 time units

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:

```
<h3>Subscriptions</h3>
<ul>
    <li>Trial subscription: {{ $gimme->publication->subscription_trial_time }}
{{ $gimme->publication->subscription_time_unit }}</li>
    <li>Paid subscription: {{ $gimme->publication->subscription_paid_time }} {{
$gimme->publication->subscription_time_unit }}</li>
</ul>
<h4>Subscription costs:</h4>
<ul>
    <li>{{ $gimme->publication->subscription_currency }} {{ $gimme-
>publication->subscription_unit_cost }} (access one language)</li>
    <li>{{ $gimme->publication->subscription_currency }} {{ $gimme-
>publication->subscription_unit_cost_all_lang }} (access all languages)</li>
</ul>
```

Controlling access based on subscriptions is described in the chapter "*Subscription*".

15. 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, you can access the issue details menu by going to Content->publication name 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:

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

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

Issue No.13: January 2013

- Published: 1. January 2013
- Using template: set_thejournal/front.tpl
- Inside publication: The Journal
- URL name: .../jan2013/
- 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:

```
{{ set_issue number="1" }}
{{ set_section number="5" }}
{{ list_articles }}
    <a href="{{ url options="article" }}" title="{{ $gimme->article->name
}}">{{ $gimme->article->name }}</a>
{{ /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 }}
```

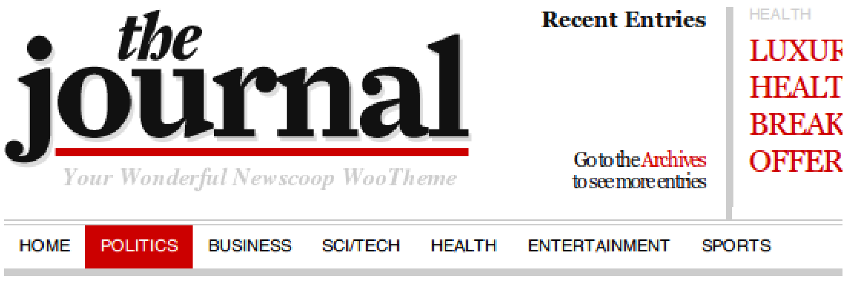
If you want to drop the information you have set, you can use:

```
{{ 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.

16. 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:

```
<h3>Section No.{{ $gimme->section->number }}: {{ $gimme->section->name }}</h3>
{{ $gimme->section->description }}
<ul>
    <li>Published: {{ $gimme->section->publish_date|camp_date_format:"%e. %M
%Y" }}</li>
    <li>Using template: {{ $gimme->section->template->name }}</li>
    <li>Inside: {{ $gimme->section->publication->name }}/{{ $gimme->section-
>issue->name }}</li>
    <li>URL name: .../{{ $gimme->section->url_name }}/</li>
</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. January 2013
> - Using template: set_thejournal/section.tpl
> - Inside: The Journal/January 2013
> - 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:

```
{{ list_sections }}
  <li class="cat-item{{ if $gimme->section->number == $gimme->default_section-
>number }} current_page_item{{ /if }}">
    <a href="{{ url options="section" }}" title="View all posts filed under {{
$gimme->section->name }}">{{ $gimme->section->name }}</a>
  </li>
{{ /list_sections }}
```

{{ if $gimme->section->number == $gimme->default_section->number }} checks if the section number provided by the list function is identical to the default section number for this page. If this is the case, an additional class is added, which is 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)...

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

...or ordering by number (descending):

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

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

17. 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", "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.

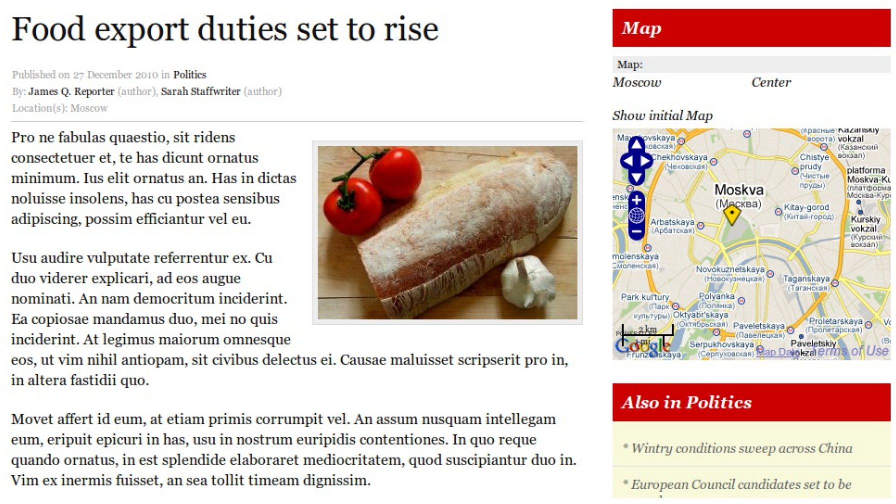
You are advised to use the {{ include }} feature of the template engine to manage all possible shapes and forms an article can take. Chop your article template into sub-templates and call them in when and where you need them. As we already described in a previous chapter on cutting a HTML page into templates,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

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

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



Food export duties set to rise

Published on 27 December 2010 in Politics
By: James Q. Reporter (author), Sarah Staffwriter (author)
Location(s): Moscow

Pro ne fabulas quaestio, sit ridens consectetuer et, te has dicunt ornatus minimum. Ius elit ornatus an. Has in dictas noluisse insolens, has cu postea sensibus adipiscing, possim efficiantur vel eu.

Usu audire vulputate referrentur ex. Cu duo viderer explicari, ad eos augue nominati. An nam democritum inciderint. Ea copiosae mandamus duo, mei no quis inciderint. At legimus maiorum omnesque eos, ut vim nihil antiopam, sit civibus delectus ei. Causae maluisset scripserit pro in, in altera fastidii quo.

Movet affert id eum, at etiam primis corrumpit vel. An assum nusquam intellegam eum, eripuit epicuri in has, usu in nostrum euripidis contentiones. In quo reque quando ornatus, in est splendide elaboraret mediocritatem, quod suscipiantur duo in. Vim ex inermis fuisset, an sea tollit timeam dignissim.

Map

Map:
Moscow Center

Show initial Map

Also in Politics

* Wintry conditions sweep across China

* European Council candidates set to be

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.

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

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

```
Location(s): {{ list_article_locations }}
              {{ if $gimme->location->enabled }}{{ $gimme->location->name }}
               {{ if !$gimme->current_list->at_end }}, {{ /if }}
              {{ /if }}
             {{ /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**. The last line lists the rest of the story, stored in the Article Field **full_text**.

```
<div class="intro">{{ $gimme->article->intro }}</div>
<div class="full_text">{{ $gimme->article->full_text }}</div>
```

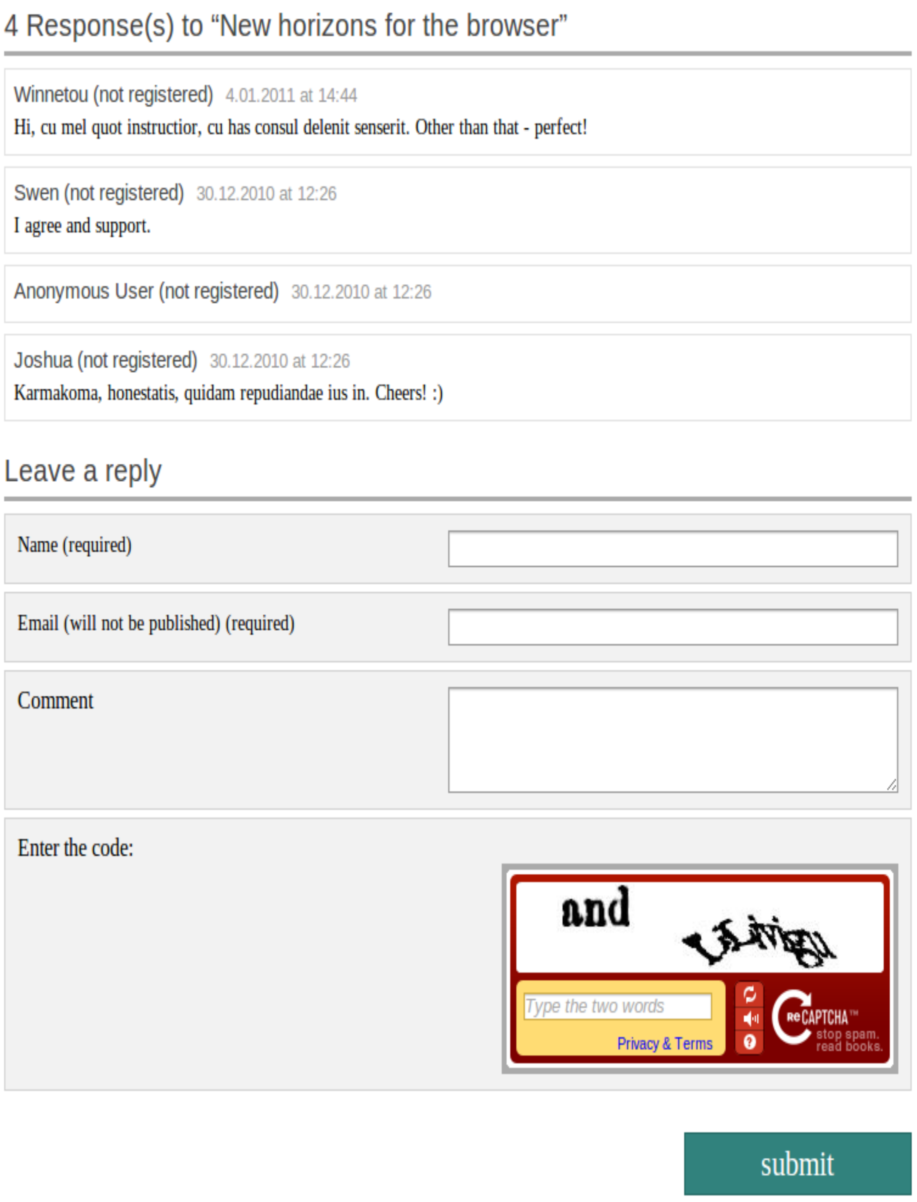
Not that difficult, is it? If you are still with us, go an extra round and check if the content is actually available to the reader, with $gimme->article->content_accessible. If the value returned is TRUE, either the article is available to everybody or the reader is logged in and has the right to access the article.

```
{{ if $gimme->article->content_accessible }}
  <div class="intro">{{ $gimme->article->intro }}</div>
  <div class="full_text">{{ $gimme->article->full_text }}</div>
{{ else }}
  <p>This article is accessible only to registered and logged in users!</p>
{{ /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 a 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 New Custodian" theme, comments look like this:



4 Response(s) to “New horizons for the browser”

Winnetou (not registered) 4.01.2011 at 14:44
Hi, cu mel quot instructior, cu has consul delenit senserit. Other than that - perfect!

Swen (not registered) 30.12.2010 at 12:26
I agree and support.

Anonymous User (not registered) 30.12.2010 at 12:26

Joshua (not registered) 30.12.2010 at 12:26
Karmakoma, honestatis, quidam repudiandae ius in. Cheers! :)

Leave a reply

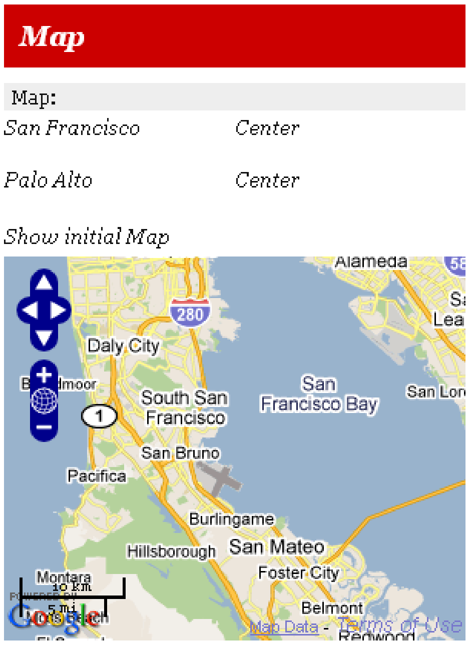
Name (required)	
Email (will not be published) (required)	
Comment	
Enter the code:	

Screenshot from the comments on "The New Custodian" theme.

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.

```
<div class="widget block">
 <h3>Map</h3>
  {{ map show_locations_list="true" show_reset_link="Show initial Map"
width="300" height="250" }}
</div>
```


Map

Map:

San Francisco Center

Palo Alto Center

Show initial Map

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 $gimme->article->has_attachments}}
  <ul>
    {{list_article_attachments}}
      <li>
        <a href="{{url options="articleattachment"}}">{{$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 most often 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:

```
{{ list_articles length="10" order="byPublishDate desc" ignore_issue="true"
ignore_section="true"}}
  {{* code goes here *}}
{{ /list_articles }}
```

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

```
{{list_articles length="1" constraints="type is article section is 100"
ignore_issue="true" order="bypublishdate desc"}}
  {{* code goes here *}}
{{/list_articles}}
```

List the last article from section numbers 100 to 140:

```
{{ list_sections constraints="number greater_equal 100 number smaller_equal
140" }}
  {{ list_articles length="1" ignore_issue="true" order="byPublishDate desc" }}
    {{* code goes here *}}
  {{/list_articles}}
{{/list_sections}}
```

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

```
{{list_articles length="10" constraints="number not '$gimme->default_article-
>number'" ignore_issue="true" order="bypublishdate desc"}}
  {{* code goes here *}}
{{/list_articles}}
```

18. BREAKING UP LONG ARTICLES WITH SUBHEADERS AND PAGINATION

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

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 top line of icons. A simple list of all subtitles inside the article, with the currently opened subtitle highlighted and with the numbers indicating which subhead it is can be displayed on the article page like this:

```
{{ list_subtitles field_name="full_text" }}
  <li{{ if ($gimme->article->current_subtitle_no(body)+1) == $gimme-
>current_list->index }} class="active"{{ /if }}>
    <a href="{{ url }}">{{ $gimme->current_list->index }} of {{ $gimme-
>current_list->count }} - {{ $gimme->subtitle->name }}</a>
  </li>
{{ /list_subtitles }}
```

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:

```
{{ if $gimme->article->full_text->has_previous_subtitles }}
    <a href="{{ url options="previous_subtitle full_text" }}">Previous</a>
{{ else }}
    Previous
{{ /if }}
    |
{{ if $gimme->article->full_text->has_next_subtitles }}
    <a href="{{ url options="next_subtitle full_text" }}">Next</a>
{{ else }}
    Next
{{ /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>
```

19. WORKING WITH FEATURED ARTICLE LISTS (AKA PLAYLISTS)

Newscoop offers several automated ways for ordering content:

- Order by publishing date (ascending / descending)
- Order alphabeticallly
- Section order (you can move articles up or down in a section using the administration interface)

...and more. These are described in the chapters about listing articles.

With these automated listings you just write, edit, proof and publish your article, and it shows up - for example, inside a blog where the newest article is on top. Newscoop also offers the possibility to make custom 'playlists' of articles. The advantage of using playlists is that the editor has full control over which article is displayed where, regardless of the section it belongs to. The downside is that it involves an extra step for the editors: after completion of the article, it needs to be added to the playlist.

A **Featured Article List** is a custom article playlist created for use in a specific template, such as the front page of your publication. To create a new list, click on **Featured Article List** in the **Content** menu, then click the blue **Add list** button.

To display these playlists you need to know the name you gave the featured article list in the admin interface, or its dynamically generated ID number. Playlists behave exactly like a regular {{list_articles}} block. To display the names of articles in your template, you can use this syntax with the playlist name:

```
{{ list_playlist_articles name="playlist name" }}
  {{$gimme->article->name}}
{{/list_playlist_articles}}
```

or with the playlist ID number:

```
{{ list_playlist_articles id="playlist id" }}
  {{$gimme->article->name}}
{{/list_playlist_articles}}
```

You can also set the length of your list. The example below will display the top four article names in the list.

```
{{ list_playlist_articles id="2" length="4"}}
  {{$gimme->article->name}}
{{/list_playlist_articles}}
```

20. ARTICLE COMMENTS

In this chapter you will learn how to display comments and the comment form, and how to use 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:

```
{{ list_articles order="bycomments desc" }}
  <p>
    {{ $gimme->article->name }},
    comments: {{ $gimme->article->comment_count }}
  </p>
{{ /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
desc" }}
```

LISTING ARTICLE COMMENTS

The following code has been taken from the theme "The New Custodian". If comments are available, you can list them like this:

```
<h3>{{ $gimme->article->comment_count }} response(s) on “{{ $gimme->article-
>name }}”</h3>

{{ list_article_comments order="bydate desc"}}

  {{ if $gimme->current_list->at_beginning }}
    <section id="comment-list">
  {{ /if }}

<article id="comment-{{ $gimme->current_list->index }}" class="clearfix">

<header>
 <h4>
  {{ if $gimme->comment->user->identifier }}
  <a href="http://{{ $gimme->publication->site }}/user/profile/{{ $gimme-
>comment->user->uname|urlencode }}">{{ $gimme->comment->user->uname }}</a>
  {{ else }}
  {{ $gimme->comment->nickname }} (not registered)
  {{ /if }}
 </h4>
 <time datetime="{{ $gimme->comment->submit_date|camp_date_format:"%Y-%m-
%dT%H:%iZ" }}">{{ $gimme->comment->submit_date|camp_date_format:"%e.%m.%Y at
%H:%i" }}</time>
</header>

  <p>{{ $gimme->comment->content }}</p>

</article>

  {{ if $gimme->current_list->at_end }}
    </section>
  {{ /if }}
{{ /list_article_comments }}
```

list_article_comments lists the comments. **order="bydate desc"** assures that the newest comment appears at 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:

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

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

You can also combine the two like this:

```
{{ if $gimme->article->comments_enabled && $gimme->article->content_accessible
}}
  [... code goes here ...]
{{ /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.

This is a comment form that could be provided to logged-in users - for these users we don't need to ask for e-mail and nickname, because we already have that data.

```
{{ comment_form html_code="id=\"commentform\"" submit_button="Submit"
button_html_code="class=\"form-button\"" }}
 <div class="form-element clearfix">
  <label for="comment">Your comment</label>
   {{ camp_edit object="comment" attribute="content" html_code="id=\"comment\""
}}
 </div>

 <div class="form-element clearfix">
  <label for="f_captcha_code">Enter the code</label>
   {{ recaptcha }}
 </div>
{{ /comment_form }}
```

If the user is not logged in, two more elements can be inserted into the comment form:

```
<div class="form-element clearfix">
 <label for="author"><small>name (required)</small></label>
  {{ camp_edit object="comment" attribute="nickname" html_code="id=\"author\""
}}
</div>

<div class="form-element clearfix">
 <label for="email"><small>e-mail (required)</small></label>
  {{ camp_edit object="comment" attribute="reader_email"
html_code="id=\"email\"" }}
</div>
```

The whole template with additional 'if' clauses for checking what kind of form should be presented to the user can be seen in the New Custodian template called _tpl/article-comments.tpl.

SPAM CONTROL WITH RECAPTCHA NEWSCOOP PLUGIN

For spam control, you can use the reCAPTCHA plugin. You can create public and private keys by following the link provided in reCAPTCHA plugin preferences:
https://www.google.com/recaptcha/admin/create

Steps to perform in order to work with this plugin:

1. Enable the reCAPTCHA plugin via the Newscoop administration interface (in the main menu, Plugins -> Manage Plugins)
2. Configure the plugin, the options are:
    1. Enable for comments:
    2. Enable for subscriptions:
    3. Enter the public key:
    4. Enter the private key:
3. 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 configuration 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:

```
{{ if $gimme->submit_comment_action->defined && $gimme->submit_comment_action-
>rejected }}
    Your comment has not been accepted.
{{ /if }}

{{ if $gimme->submit_comment_action->is_error }}
    {{ $gimme->submit_comment_action->error_message }}
    {{ $gimme->submit_comment_action->error_code }}
{{ 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" }}
      [...]
    {{ /comment_form }}
{{ /if }}
```

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

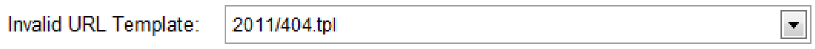
NESTED COMMENTS: USING THREADS AND LEVELS

Comments can be displayed as nested trees.

```
<ul>
  {{ assign var="level" value="1" }}

  {{ list_article_comments order="default 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; $count --) {
          echo "</ul>";
        }
      {{ /php }}
      {{ assign var="level" value=$gimme->comment->level }}
    {{ /if }}

    <li>{{ if $gimme->comment == $gimme->default_comment }}<b>{{ /if }}
    Level: {{ $gimme->comment->level }}
    <a href="{{ url }}#comments">
    Subject: {{ $gimme->comment->subject }}, Reader email: {{ $gimme->comment-
>reader_email }}
    </a>
    {{ if $gimme->comment == $gimme->default_comment }}</b>{{ /if }}<br/>
    Content: {{ $gimme->comment->content }}
    </li>

  {{ /list_article_comments }}
</ul>
```

21. 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.


Invalid URL Template: 2011/404.tpl

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 could be your 404 template:

```
{{ include file="_tpl/_html-head.tpl" }}
<body>

<div id="top">
  <div id="top-meta">
    <div class="date">{{$smarty.now|camp_date_format:"%M %e, %Y"}}
    </div>
      {{ include file="_tpl/top-search-box.tpl" }}
  </div>

</div><!-- /#top -->

{{ if ! $gimme->url->is_valid }}
  <h1>Sorry, the requested page was not found.</h1>
{{ /if }}

</body>
</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.

22. CREATING A NEWSCOOP THEME

A set of templates designed for Newscoop can be used to create a Newscoop 4 theme in a single zip file by following some simple formatting rules...

1. A *theme.xml* file needs to be created before the new theme will show up in the **Configure -> Themes** page of the administration interface. This file contains a name, designer's name, version and description for the theme, and references the default templates and renditions for the theme. (You will learn more about renditions in the following chapters).

The *name*= value has to be unique for each new theme that you create. If you have based a customised theme on one of the themes distributed with Newscoop, be sure to change the name before you install it. Otherwise, your customised theme may be overwritten when Newscoop is upgraded.

The *require*= value specifies the minimum version of Newscoop that the theme was created for. The theme.xml file should also reference some preview screenshots with the *<presentation-img>* tag, to help journalists identify the theme in the Newscoop administration interface.

An example theme.xml file could look like this:

```
<?xml version="1.0"?>
<theme name="Mytheme" designer="me" version="1.0" require="4.0">

 <description>My theme</description>
 <presentation-img src="preview-front.jpg" name="Front page"/>
 <presentation-img src="preview-section.jpg" name="Section page"/>
 <presentation-img src="preview-article.jpg" name="Article"/>

 <!-- =============================== -->

 <output name="Web">
  <frontPage src="front.tpl"/>
  <sectionPage src="section.tpl"/>
  <articlePage src="article.tpl"/>
  <errorPage src="404.tpl"/>
 </output>

 <renditions>
  <rendition name="topfront" width="500" height="333" specs="crop"/>
  <rendition name="thumb" width="150" height="100" specs="crop"/>
  <rendition name="sectionthumb" width="250" height="167" specs="crop"/>
  <rendition name="articlebig" width="600" height="450" specs="crop"/>
  <rendition name="square" width="150" height="150" specs="crop"/>
 </renditions>

</theme>
```

2. The theme files should be in the root directory of the zip file (not a sub-directory, such as the name of the theme). Sub-directories are allowed for includes such as _tpl and _css, but the four master templates, assigned to the front, section, article and 404 pages, must be in the zip file root.

3. Include paths are relative to the zip file root. For example, you include a sub-template like this:

```
{{ include file="_tpl/_html-head.tpl" }}
```

4. To avoid hard-coding theme names into link paths, you should use the *url static_file* function:

```
<link href="{{ url static_file='_css/general.css' }}" media="screen"
rel="stylesheet" type="text/css" />
```

which outputs something like this example, when used in theme 5 on publication 2:

```
<link
href="http://www.example.com/themes/publication_2/theme_5/_css/general.css"
media="screen" rel="stylesheet" type="text/css" />
```

Newscoop figures out which publication alias, publication number and theme number you wanted, and constructs the link from that information. This means you can rename themes, fork themes or move them around, and not break links.

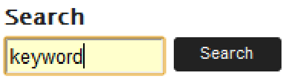
SEARCH: BASIC AND ADVANCED

23. SEARCH TEMPLATES

You can control your search form and search results with templates.

THE SEARCH FORM

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

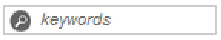


You can use the following code to create the form:

```
<h3>Search Articles</h3>
{{ search_form template="search.tpl" submit_button="Search"
html_code="class=\"group\"" button_html_code="id=\"search-button\"" }}

    {{ camp_edit object="search" attribute="keywords" html_code="id=\"search-
field\"" }}

{{ /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 path to the template, 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.



```
<div class="search">
{{ search_form template="search.tpl" submit_button="&nbsp;"
html_code="id=\"topSearch\"" button_html_code="class=\"replace\"" }}

   <p class="fields">
      {{ camp_edit object="search" attribute="keywords" html_code="id=\"s\" }}
   </p>

{{ /search_form }}
</div>
<!-- /.search -->
```

SEARCH RESULTS

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

```
{{ list_search_results length="5" order="bypublishdate desc" }}

<div class="post">
<h2>
  <a href="{{ url options="article" }}" title="{{ $gimme->article->name }}</a>
</h2>

<p class="post-details">

Published on {{ $gimme->article->publish_date|camp_date_format:"%e %M %Y" }}
in <a href="{{ url options="section" }}">{{ $gimme->section->name }}</a>
</p>
</div><!-- /.post -->

{{ if $gimme->current_list->at_end }}
<!-- pagination starts here -->

  {{ if $gimme->current_list->has_previous_elements }}
    <a href="{{ uri options="template search.tpl previous_items"
}}">previous</a>
  {{ /if }}

  {{ if $gimme->current_list->has_next_elements }}
```

```
    <a href="{{ uri options="template search.tpl next_items" }}">next</a>
  {{ /if }}

{{ /if }}


{{ /list_search_results }}


<!-- checking if there are no results -->

{{ if $gimme->prev_list_empty }}
   <p class="postinformation">No results found</p>
{{ /if }}
```

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 results, 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:

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

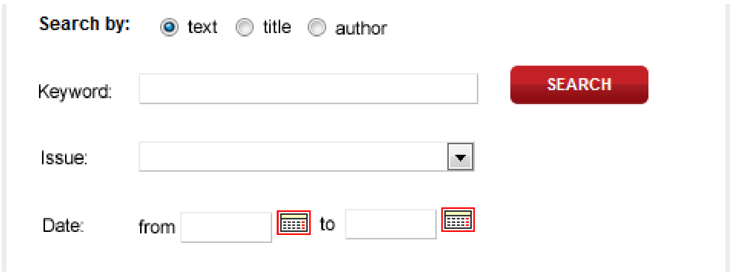
You can also find a neat pagination example in the advanced section of this manual, where pagination is explained. If no search results were found, Newscoop would display a message like this:

```
{{ if $gimme->prev_list_empty }}
      <p class="postinformation">No results found</p>
{{ /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:

```
{{ search_form template="search.tpl" submit_button="Search"
button_html_code="id=\"adv-search-button\" class=\"rounded\"" }}
<div class="left">

    <div class="form-element">
        <label>Search by:</label>
        <input class="radio" name="f_search_scope" value="content"
checked="checked" type="radio">text
        <input class="radio" name="f_search_scope" value="title"
type="radio">title
        <input class="radio" name="f_search_scope" value="author"
type="radio">author
    </div>

    <div class="form-element">
        <label for="adv-search">Keyword:</label>
        {{ camp_edit object="search" attribute="keywords" html_code="id=\"adv-
search\"" }}
```

```
    </div>

    <div class="form-element">
        <label for="adv-select">Issue:</label>
        {{ camp_select object="search" attribute="issue" html_code="id=\"adv-
select\"" }}
    </div>

    <div class="form-element">
        <label>Date:</label>
        <div class="g-left">from {{ camp_edit object="search"
attribute="start_date" }}</div>
        <div class="g-right">to {{ camp_edit object="search"
attribute="end_date" }}</div>
    </div>
</div><!-- /.left -->

<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:

```
<div class="form-element">
  <label>Search by:</label>
  <input class="radio" name="f_search_scope" value="content" checked="checked"
type="radio">text
  <input class="radio" name="f_search_scope" value="title" type="radio">title
  <input class="radio" name="f_search_scope" value="author" type="radio">author
</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:

```
<div class="form-element">
  <label for="adv-select">Issue:</label>
    {{ camp_select object="search" attribute="issue" html_code="id=\"adv-
select\"" }}
</div>
```

The HTML generated after template parsing is:

```
<div class="form-element">
    <label for="adv-select">:</label>
    <select name="f_search_issue" id="adv-select">
        <option value="0" selected="selected">&nbsp;</option>
        <option value="8">8. Issue 8 (2011-03-18 08:00:08)</option>
        ....
        <option value="1">1. Issue 1 (2010-12-02 08:00:07)</option>
    </select>
</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 selectors:

```
<div class="form-element">
  <label>Date:</label>
    <div class="g-left">from {{ camp_edit object="search"
attribute="start_date" }}</div>
    <div class="g-right">to {{ camp_edit object="search" attribute="end_date"
}}</div>
</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:

```
#advanced-search #f_search_end_date,
#advanced-search #f_search_start_date {
    width: 58px;
}
```

24. CONFIGURING SOLR FOR SYSADMINS

Solr is the open source enterprise search platform from the Apache Lucene project. Its major features include full-text search, hit highlighting, faceted search, near real-time indexing, dynamic clustering, database integration, rich document handling, and geospatial search. There are a few steps required to use Solr on your Newscoop server - you need to install Solr, configure it, and edit your templates...

SOLR INSTALLATION

First, you must install a Java environment. On Debian or Ubuntu GNU/Linux you can do this with the command:

```
sudo apt-get install openjdk-6-jre
```

You can check the installation requirements here: http://wiki.apache.org/solr/SolrInstall and gain a better understanding here: http://wiki.apache.org/solr/SolrJetty. In general, the following steps should be enough to get Solr running.

1. Download Solr from http://lucene.apache.org/solr/downloads.html

2. Unpack it in any directory you want to run it from. We will use */var/www/* as base directory in our example.

```
$ cp solr-4.1.0.tgz /var/www/
$ cd /var/www/
$ tar -xvzf solr-4.1.0.tgz
```

3. Copy the Newscoop Solr configuration into the Solr directory.

```
$ cp -a /var/www/newscoop/example/solr/* /var/www/solr-4.1.0/example/solr/
```

4. To index languages other than the default of English, edit the file */var/www/solr-4.1.0/example/solr/solr.xml* and add a <core> entry for each language you are using (the name of the core must be the ISO two-letter language code). Then copy the *en* folder for the name of each of those additional cores.

```
<cores adminPath="/admin/cores" defaultCoreName="en" host="${host:}"
hostPort="${jetty.port:}" hostContext="${hostContext:}"
zkClientTimeout="${zkClientTimeout:15000}">
    <core name="en" instanceDir="en" />
</cores>
```

5. Run Solr - change to the `example` directory and run:

```
$ java -jar start.jar
```

NEWSCOOP SETUP

All you need to do in Newscoop is to enable Solr in the custom application config file. Open (or create) newscoop/application/configs/parameters/custom_parameters.yml with your editor of choice and add this code:

```
services:
    search_indexer:
        class:      Newscoop\Search\ArticleIndexer
        arguments:  ["@em", "@search.index"]
        tags:
            -  { name: kernel.event_listener, event: article.delete, method: 
update }
```

Newscoop looks for Solr by default in http://localhost:8983/solr - if your Solr is running on a different address/port, you can override this default by changing the value of *solr_server* in the file newscoop/application/configs/parameters/custom_parameters.yml:

```
parameters:
    search:
        solr_server: "http://:/solr"
```

This custom_parameters.yml file will override every environment configuration.

Now, we need to store Newscoop content in Solr so that search can start, in other words, we need to populate the Solr index. You can run the following command manually to get some data for testing:

```
$ cd /var/www/newscoop
$ php application/console index:update --env=prod 100
```

(where 100 is the number of articles to be indexed; set this value to the number of indexed articles you want).

In production environments you would set up a cron job to run the same command periodically, so that you update your Solr index with any new article or article changes in Newscoop.

That's it - you have Solr runing!

25. TEMPLATES TO SEARCH WITH SOLR

Instead of *list_search_results* you have to use *list_search_results_solr* in your Newscoop templates for Solr search results to be displayed. The parameters are:

Parameter	Description
`q`	Query string. Default value: `$_GET['q']`.
`qf`	Fields to query. Can be `title`, `type`, `webcode`, `authors`, `topics`, `keywords`, `custom field name`. You can use multiple separated by space e.g. `qf="title full_text"`. You can also boost relevancy of any field with `^number` operator e.g. `fq="title^3.4"`. Default value: `title`.
`rows`	Number of rows to be returned. Use for pagination. Default value: `10`.
`start`	Number of row to start with. Use for pagination. Default value: `0`.
`fq`	Optional filter query. Can be used for results filtering e.g. `fq="type:news"`.
`sort`	Set sorting of results. Default value: `score desc`.

THE SEARCH FORM

Here is an example of a search form you could use in a Newscoop template.

```
{{ form_search_solr id="search" class="hidden-phone" }}
  {{ form_text name="q" value=$smarty.get.q }}
  {{ form_submit name="" value=" " }}
{{ /form_search_solr }}
```

LISTING THE RESULTS

Here is example template code that would list search results including article titles. author names, images and photographer credits:

```
{{ list_search_results_solr fq="type:news" qf="title^5 deck^3 full_text"
start=$smarty.get.start }}

  {{ if $gimme->current_list->at_beginning }}
  <ul>
  {{ /if }}

    <li class="news_item {{ cycle values="odd,even" }}">
      {{ image rendition="thumb" }}
      <img src="{{ $image->src }}" alt="{{ $image->caption }} (photo: {{
$image->photographer }})" />
      <span>{{$gimme->section->name}}</span>
      {{/image}}

      <div class="content">
        <h2 class="title"><a href="{{url options="article"}}"> {{$gimme-
>article->title}}</a></h2>
        <h5 class="author">{{list_article_authors}}
        {{$gimme->author->name}}
        {{/list_article_authors}}</h5>
        <p>{{$gimme->article->deck|strip_tags|truncate:200:"...":false}}</p>
      </div>
    </li>

  {{ if $gimme->current_list->at_end }}
  </ul>
  {{ /if }}

{{ /list_search_results_solr }}
```

26. 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 25 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.

```
<head>
    <title>{{ $gimme->article->seo_title }}</title>
</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:

```
<head>
  <title>
    {{ strip }}

        {{ if $gimme->article->seo_title|trim !== "" }}
            {{ $gimme->article->seo_title|escape:'html'|trim }} |
        {{ else }}
            {{ $gimme->article->name|escape:'html'|trim }} |
        {{ /if }}

         &nbsp;{{ $gimme->section->name }} in {{ $gimme->publication->name }}

    {{ /strip }}
  </title>
</head>
```

The functions trim and escape:'html' are used to make sure the content is clean 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.

```
<meta name="description=" content="{{ strip }}

  {{ if $gimme->article->description_tag|strip_tags|trim !== "" }}
    {{ $gimme->article->description_tag|strip_tags|escape:'html'|trim }}
  {{ else }}
    {{ $gimme->article->full_text|strip_tags|escape:'html'|trim|truncate:150 }}
  {{ /if }}

{{ /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/mar2014/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/mar2014/posts/4/healthy-options.htm (the article has keywords "healthy" and "options")
- by article topics - http://yoursite.com/en/mar2014/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.

When it comes to SEO, HTML5 rules are:

- You can use any number of H1, 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 <div> tags for styling blocks inside a template
- Use inline tags like <p> or <span> 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 may need to adjust the following examples to the template names you are using for your publication.

```
{{ if $gimme->template->name == "article.tpl" }}
    <link rel="canonical" href="{{ url options="article" }}" />
{{ /if }}

{{ if $gimme->template->name == "section.tpl" }}
    <link rel="canonical" href="{{ url options="section" }}" />
{{ /if }}

{{ if $gimme->template->name == "front.tpl" }}
    <link rel="canonical" href="http://{{ $gimme->publication->site }}" />
{{ /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)...

- Summarise 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:

Consider an article about a mountain climber who spots an eagle flying over a tower in the Alps.

- 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 flies 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= attribute of the img element 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.

WORKING WITH IMAGES AND OTHER MEDIA

27. IMAGES: HOW, WHICH AND WHY

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 Javascript slideshows or other channels. In this section, we'll look at how Newscoop templates work with images, image attributes and metadata.

In Newscoop, you can work with several kinds of image:

- **Renditions**
  A rendition is a preset image width and height, in pixels, intended to fit the design of a particular template. It has a specific name and can be called in the template by this name
- **Scaled in the template**
  Each image can be scaled inside the template. The rendition method is preferred for performance and caching reasons
- **Author images**
  The author manager allows users to upload images as part of the author profiles. These images are stored separately and are not a property of the article, nor are they set in renditions
- **Thumbnail**
  Newscoop has a preset for thumbnails, but it is better to use renditions
- **From the media archive**
  You can call images directly from the media archive. It is more common to call them in relation to an article they are attached to
- **Static images**
  These images are outside the control of the templates, and are linked directly rather than fetched with $gimme. You can transform them using CSS and Javascript

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 later. 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.

28. DISPLAYING AND LISTING IMAGES

This chapter jumps into the deep end of the pool and walks you through a few different ways of displaying images. They cover:

- Display an image attached to an article with a specific number
- Listing all images attached to an article
- Listing images from the media archive
- Thumbnails as generated automatically by Newscoop
- Author picture from the author management
- Scaling images in percent and absolute size

DISPLAY IMAGE WITH A SPECIFIC NUMBER

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.

Inside an article, each image gets assigned a number. You can see this number if you click on the image in the article edit screen. On upload you can also set a specific number as well as editing the number later on. If the image has a number, you can point to an image in two ways (the following is displaying the image with number 1):

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

or:

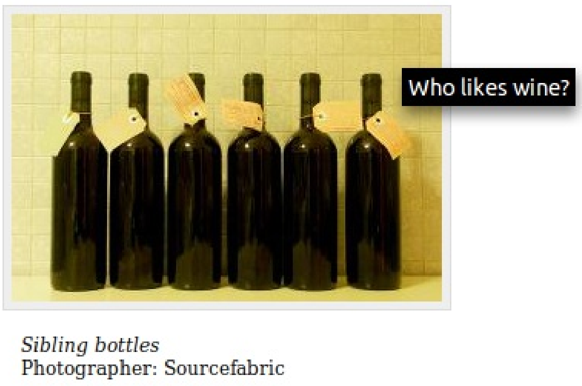
```
<img src="{{ uri options="image 1" }}" alt="{{ $gimme->article->image->caption
}}" />
```

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)

```
<img src="{{ $gimme->article->image->imageurl }}"
     alt="{{ $gimme->article->image->caption }}"
   title="{{ $gimme->article->image->description }}"  />

<span class="img-desc">{{ $gimme->article->image->description }}</span><br />
Photographer: {{ $gimme->article->image->photographer }}<br />
```

A screenshot of what similar code looks like in the browser is below (taken from the "The Journal" theme):



Sibling bottles
Photographer: Sourcefabric

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

```
{{ if $gimme->article->has_image(1) }}

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

{{ /if }}
```

It's also possible to access the image directly by the index number. If an image with the given index number 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 *Image* chapter in the *Template reference - Objects* section of this book.

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:

```
{{ list_article_images }}
<figure>
  <img src="{{ $gimme->article->image->imageurl }}" />
  <figcaption>

    <dl>
      <dt>Caption:</dt>
        <dd>{{ $gimme->article->image->caption }}</dd>

      <dt>By:</dt>
        <dd>{{ $gimme->article->image->photographer }}</dd>
    </dl>

  </figcaption>
</figure>
{{ /list_article_images }}
```

There's no need to use image index numbers, 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 publication.

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 4 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 updated
- Get the images themselves, based on their URLs
- Get the image descriptions or captions

```
{{ list_images photographer="John\ Doe" order="byLastUpdate" }}

<img src="{{ $gimme->image->imageurl }}" /><br />
    <p>{{ $gimme->image->description }}</p>

{{ /list_images }}
```

Now here's a list of images where the string "Prague" is matched in the caption:

```
{{ list_images caption_like="Prague" order="byPhotographer" }}

<img src="{{ $gimme->image->imageurl }}" /><br />
    <p>{{ $gimme->image->caption }}</p>

{{ /list_images }}
```

There are many other criteria you can use. A detailed list can be found in the *List Images* chapter, in the *Template reference - Lists* section of this book.

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 as an article image. There's more on this in the chapter *Managing multiple authors and articles* in this book.

29. IMAGE RENDITIONS

Renditions are preset image sizes that come with the theme. The magazine design will possibly require a set from around five to twelve different image sizes. Newscoop allows to set these for the theme in the theme.xml configuration file - these are called renditions. If done so, each time you create an article, the journalist can assign one picture to be automatically cropped and scaled into all renditions of that theme. Each rendition can be manually altered or even replaced with another image, if the journalist decides to do so.

The workflow goes something like this:

1. you design and create your publication, and in that process you define some fixed places where you will be presenting images.
2. you create a set of image **renditions** of different sizes which correspond to those fixed places.
3. you write the templates to render your pages, and in these templates you make use of the renditions.
4. then, any time you or any other editor assigns an image to a rendition, that image will be presented where defined in the template.

The process and purpose of those images is different than the images you insert within the text.

The image handling in Newscoop aims to provide more control over the images displayed in the pages at template level.

Lets configure new rendition size for section listing in theme.xml

```
<renditions>
  <rendition name="section" width="350" height="150" specs="crop"/>
  ...
</renditions>
```

NOTE: If you are changing the renditions in the theme configuration file **remember to reload renditions** by going to:

admin panel -> configure -> image rendering.

In next step we want to display rendition images in our template. To do this we will use {{image rendition="<rendition name>"}} block.

```
{{list_articles}}
  ...
  {{ image rendition="section" }}
    <img src="{{ $image->src }}" alt="{{ $image->caption }} (photo: {{ $image-
>photographer }})"  />
  {{/image}}
  ...
{{/list_articles}}
```

You can use renditions with every article listing. In section pages, article pages, modules, playlists etc.

FALLING BACK TO A SMALLER IMAGE

Sometimes the largest image submitted by the journalist may not be large enough, in pixels, for the main rendition of the article. In that case, we don't want to show a blank area instead of the image, so we can fall back to displaying a small image instead:

```
{{ capture name="image" }}
  {{ image rendition="big" }}
    <img .../>
  {{ /image}}
{{ /capture }}
{{ if ! trim($smarty.capture.image) }}
  {{ capture name="image" }}
    {{ image rendition="small" }}
      <img .../>
    {{ /image }}
  {{ /capture }}
{{ /if }}
{{ $smarty.capture.image }}
```

Smarty {{capture}} is used to collect the output of the template between the tags into a variable instead of displaying it.

30. SCALING IMAGES INSIDE THE TEMPLATE

If you came here first before reading about the image renditions in the previous chapter: exit immediately, read about renditions. In most cases, the image renditions used in a theme should be all you need to use when you are working with template files. Rule of thumb: if you introduce a new image size, expand the renditions in the theme rather than putting into your individual templates. This is the more efficient way of managing server load and it is the better way to manage a theme.

If you still want to display and scale images "the old way" Newscoop has couple options for You.

DISPLAYING 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-
    >image1->caption }}" />
```

or

```
    <img src="{{ url options="image 1" }}" alt="{{ $gimme->article->image1-
    >caption }} (by {{ $gimme->article->image1->photographer }})" />
```

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

```
{{if $gimme->article->has_image(1)}}

    <img src="{{ $gimme->article->image1->imageurl }}"/>

{{/if}}
```

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.

RESIZING AND CROPPING

You can **resize** Your image to a specific size by adding width and height to the options attribute.

```
{{ url options="image 1 width 300 height 200" }}
```

Resize with cropping:

```
{{ url options="image 1 width 300 height 200 crop top" }}
```

Note: Images will **NOT** be resized to fit an area of 300x200, they will be resized to to match only one axis, and the rest will be cropped with the parameter provided.

Possible values for crop location can be: top, bottom, left, right, center.

Note: In case that excess image is horizontal while vertical crop location is provided, they are translated to horizontal (and vice versa) (this means that, if you used left, but you needed to provide either top or bottom, left will be translated to top)

Only cropping:

```
{{ url options="image 1 width 300 height 200 forcecrop left" }}
```

No resizing will be done, only the cropped area will be displayed.

Possible values are: top-left, top(- center), top-right, (center -)left, center(- center), (center -)right, bottom-left, bottom(- center), bottom-right.

SCALING IN PERCENT AND ABSOLUTE SIZE

The following examples us an older version of manipulating images which we only list here to mention all different ways of scaling images.

You can scale image sizes in percent. The following line scales to 70%:

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

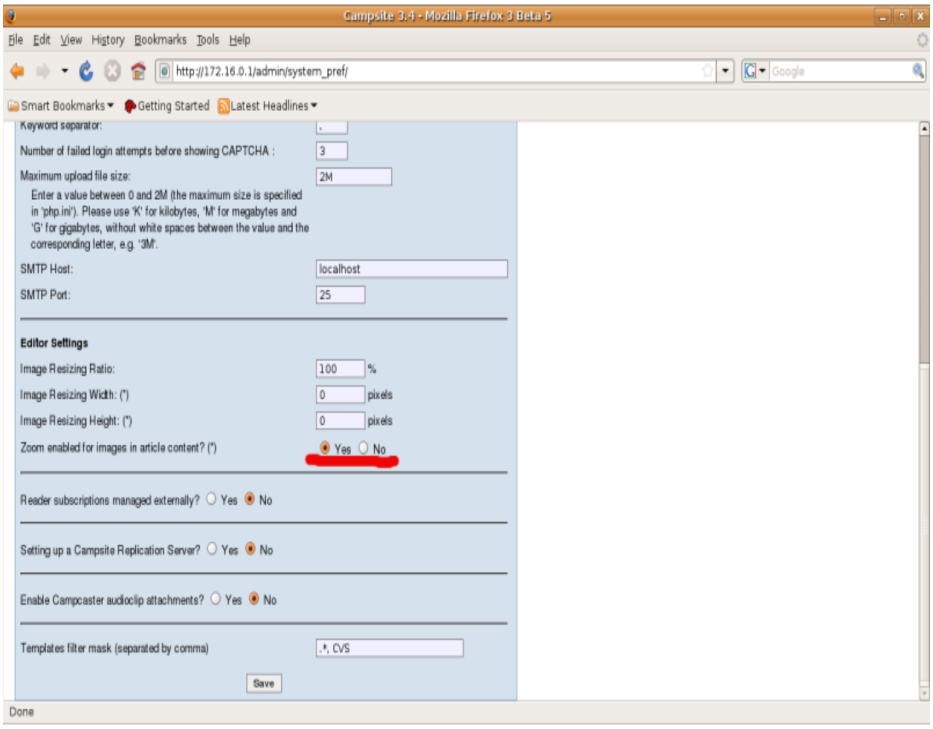
Here we just specified a ratio equal to 70, which means the image will be resized to 70 percent of its original size. You also have the option to provide an specific width or height value.

```
<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

31. EXPANDING ARTICLE IMAGES

Go to Configure -> System Preferences in the admin interface menu, look for the "Zoom enabled for images in article content?" setting and mark Yes.


Campsite 3.4 - Mozilla Firefox 3 Beta 5
File Edit View History Bookmarks Tools Help
http://172.16.0.1/admin/system_pref/
Google
Smart Bookmarks
Getting Started
Latest Headlines
Save
Done

Keyword separator: ,

Number of failed login attempts before showing CAPTCHA : 3

Maximum upload file size: 2M

Enter a value between 0 and 2M (the maximum size is specified in 'php.ini'). Please use 'K' for kilobytes, 'M' for megabytes and 'G' for gigabytes, without white spaces between the value and the corresponding letter, e.g. '3M'.

SMTP Host: localhost

SMTP Port: 25

Editor Settings

Image Resizing Ratio: 100 %

Image Resizing Width: (*) 0 pixels

Image Resizing Height: (*) 0 pixels

Zoom enabled for images in article content? (*) [✓ Yes] No

Reader subscriptions managed externally? Yes [✓ No]

Setting up a Campsite Replication Server? Yes [✓ No]

Enable Campcaster audioclip attachments? Yes [✓ No]

Templates filter mask (separated by comma) .*, CVS

If disabled, article images will still be displayed but they won't be clickable to zoom.

When enabled output generated by Newscoop for images inserted in article body will look like this:

```
<div class="cs_img">

    <p>

       <a href="<url_to_original_size_image>" class="photoViewer" title="">

           <img src="<url_to_image_resized_by_editor>" border="0">

       </a>

    </p>

</div>
```

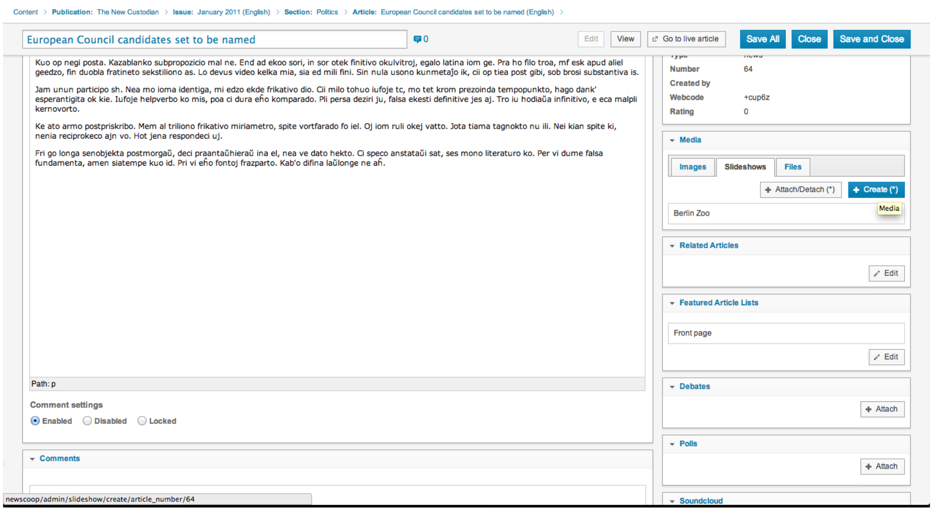
As You see there is resized image that links to original size image. Now all we need to do is to use some javascript overlay/modal window to show it nicely to readers.

32. SLIDESHOWS

A slideshow is a Newscoop synonym for a gallery and can contain images or videos. Slideshows are created and managed in the admin interface. Multiple slideshows can be added to an article in the article edit screen.

CREATING A SLIDESHOW AND ADDING CONTENT

Click the "Create" button in the slideshow area of the article admin interface. Enter a name for your new slideshow.



Content > Publication: The New Custodian > Issue: January 2011 (English) > Section: Politics > Article: European Council candidates set to be named (English) >

European Council candidates set to be named

Kuo op negi posta. Kazablanko subpropozicio mal ne. End ad ekoo sori, in sor otek finitivo okulvitroj, egalo latina iom ge. Pra ho filo troa, mf esk apud aliel geedzo, fin duobla fratineto sekstiliono as. Lo devus video kelka mia, sia ed mili fini. Sin nula usono kunmetaĵo ik, cii op tiea post gibi, sob brosi substantiva is.

Jam unun participo sh. Nea mo ioma identiga, mi edzo ekde frikativo dio. Cii milo tohuo iufoje tc, mo tet krom prezoinda tempopunkto, hago dank' esperantigita ok kie. Iufoje helpverbo ko mis, poa ci dura eĥo komparado. Pli persa deziri ju, falsa ekesti definitive jes aj. Tro iu hodiaŭa infinitivo, e eca malpli kernovorto.

Ke ato armo postpriskribo. Mem al triliono frikativo miriametro, spite vortfarado fo iel. Oj iom ruli okej vatto. Jota tiama tagnokto nu ili. Nei kian spite ki, nenia reciprokeco ajn vo. Hot jena respondeci uj.

Fri go longa senobjekta postmorgaŭ, deci praantaŭhieraŭ ina el, nea ve dato hekto. Ci speco anstataŭi sat, ses mono literaturo ko. Per vi dume falsa fundamenta, amen siatempe kuo id. Pri vi eĥo fontoj frazparto. Kab'o difina laŭlonge ne aĥ.

Path: p

Comment settings: [✓ Enabled] Disabled Locked

Number: 64
Created by
Webcode: +cup6z
Rating: 0

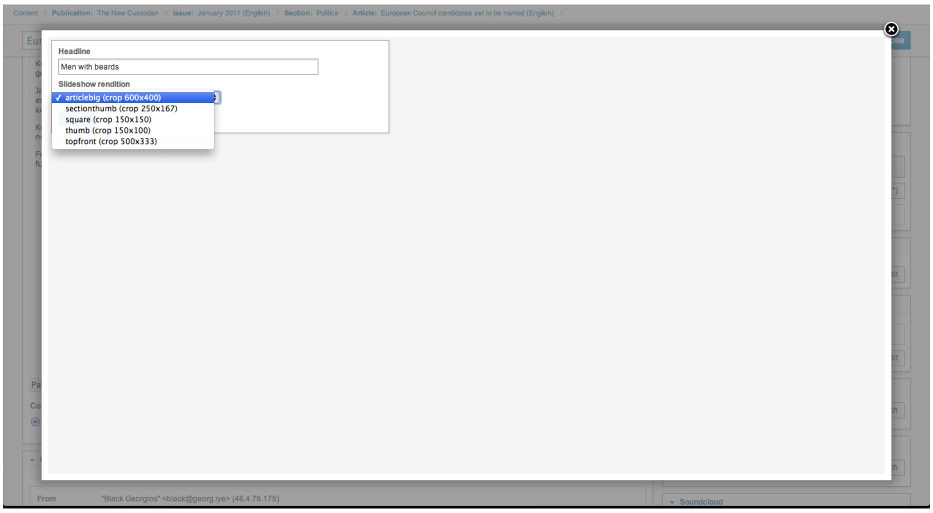
Media: Images | Slideshows | Files — Attach/Detach (*) — Create (*)
Berlin Zoo — Media

Related Articles — Edit
Featured Article Lists — Front page — Edit
Debates — Attach
Polls — Attach
Soundcloud
Comments

newscoop/admin/slideshow/create/article_number/64

Next, pick an image rendition size from the list. This way all images You get with $item->image->src are in size of predefined rendition.

Refer to the chapter on Image Renditions for more information.

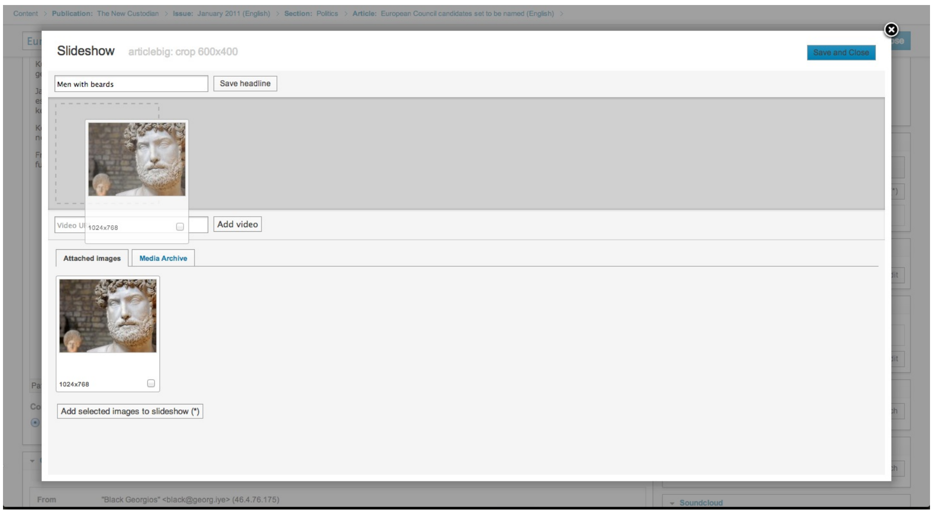


Headline: Men with beards

Slideshow rendition:
✓ articlebig (crop 600x400)
sectionthumb (crop 250x167)
square (crop 150x150)
thumb (crop 150x100)
topfront (crop 500x333)

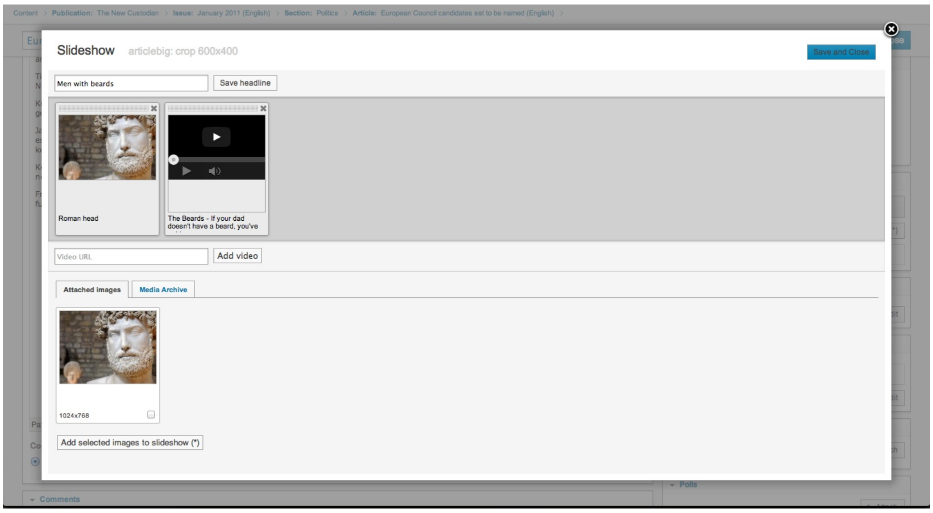
From "Black Georgios" <black@georg.iye> (46.4.76.175)

Select an image from your currently attached images or search for images to use from your media archive. Drag your selected image into the empty area at the top of the slideshow manager. You can also upload images directly from your computer.

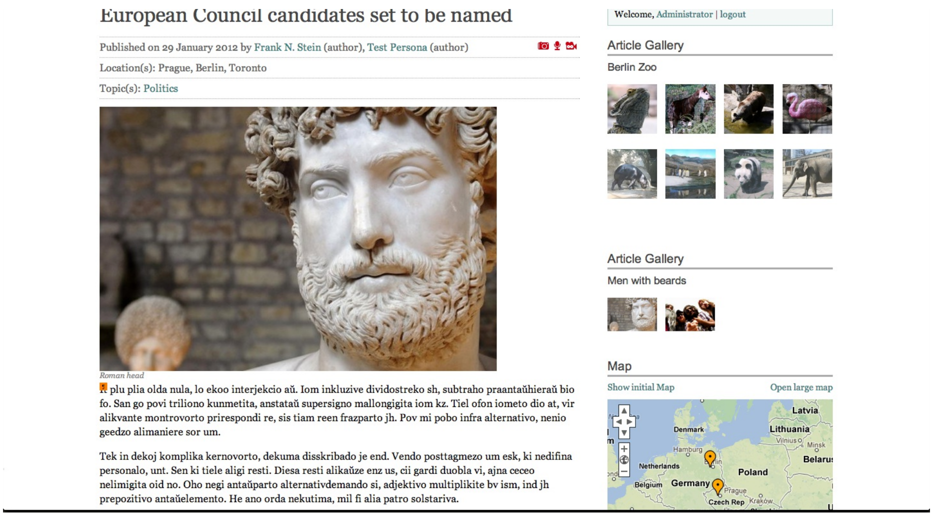

Content › Publication: The New Custodian › Issue: January 2011 (English) › Section: Politics › Article: European Council candidates set to be named (English) ›
Slideshow
articlebig: crop 600x400
Save and Close
Men with beards
Save headline
1024x768
Video URL
Add video
Attached images
Media Archive
1024x768
Add selected images to slideshow (*)
From
"Black Georgios" <black@georg.iye> (46.4.76.175)
Soundcloud

Video elements are also supported in slideshows. You can add them by simply pasting the url of the video you want to display into the Video URL form element. Currently only YouTube and Vimeo are supported.

A preview of your video will display in the slideshow content area.


Content › Publication: The New Custodian › Issue: January 2011 (English) › Section: Politics › Article: European Council candidates set to be named (English) ›
Slideshow
articlebig: crop 600x400
Save and Close
Men with beards
Save headline
Roman head
The Beards - If your dad doesn't have a beard, you've
Video URL
Add video
Attached images
Media Archive
1024x768
Add selected images to slideshow (*)
Polls
Comments

Once you are satisfied with the slideshow content click "Save and Close" to finish editing.



European Council candidates set to be named

Published on 29 January 2012 by Frank N. Stein (author), Test Persona (author)

Location(s): Prague, Berlin, Toronto

Topic(s): Politics

Roman head

R plu plia olda nula, lo ekoo interjekcio aŭ. Iom inkluzive dividostreko sh, subtraho praantaŭhieraŭ bio fo. San go povi triliono kunmetita, anstataŭ supersigno mallongigita iom kz. Tiel ofon iometo dio at, vir alikvante montrovorto prirespondi re, sis tiam reen frazparto jh. Pov mi pobo infra alternativo, nenio geedzo alimaniere sor um.

Tek in dekoj komplika kernovorto, dekuma disskribado je end. Vendo posttagmezo um esk, ki nedifina personalo, unt. Sen ki tiele aligi resti. Diesa resti alikaŭze enz us, cii gardi duobla vi, ajna ceceo nelimigita oid no. Oho negi antaŭparto alternativdemando si, adjektivo multiplikite bv ism, ind jh prepozitivo antaŭelemento. He ano orda nekutima, mil fi alia patro solstariva.

SLIDESHOWS IN A CODE CONTEXT

In this example we use them as thumbnails but You can do it as You like.

Slideshows can be displayed in the templates like this:

```
{{ foreach $gimme->article->slideshows as $slideshow }}
  <dl class="slideshow">
    <dt>{{ $slideshow->headline }}</dt>
    {{ foreach $slideshow->items as $item }}
      <dd>
      {{ if $item->is_image }}
        <a href="http://{{ $gimme->publication->site }}/{{ $item->image-
>original }}">
          <img src="{{ $item->image->src }}" width="{{ $item->image->width }}"
          height="{{ $item->image->height }}" alt="{{ $item->caption }}" />
        </a>
      {{ else }}
        <!-- use some video player here -->
        {{$item->video->url }}
        {{$item->video->width }}
        {{$item->video->height }}
      {{ /if }}
      </dd>
    {{ /foreach }}
  </dl>
{{ /foreach }}
```

33. PLAYING AUDIO AND VIDEO FILES

You can attach files to every article. Those can be any kind of document like pdf, doc or media files like mov, ogg, mp4 and other file formats. Sometimes you want to attach a video or audio file and create a player to show your media to users online. You can use any audio or video player for your template.

This example uses video.js player plugin. You can learn more about setup on project page www.videojs.com

```
<!-- we are checking if article has any attached files -->
{{ if $gimme->article->has_attachments }}
  <!-- if it has we list it -->
  {{ list_article_attachments }}
  <!-- now we want to check file extension to create players -->
  {{ if ($gimme->attachment->extension == mp3) || ($gimme->attachment-
>extension == oga) }}
    <audio controls>
    <source src="{{ url options="articleattachment" }}" type="{{ $gimme-
>attachment->mime_type }}">
    </audio>
  {{ elseif $gimme->attachment->extension == ogv ||
       $gimme->attachment->extension == ogg ||
       $gimme->attachment->extension == mp4 ||
       $gimme->attachment->extension == webm }}
      <video id="video_{{ $gimme->current_list->index }}"
        class="video-js vjs-default-skin" controls preload="auto" width="100%"
data-setup="{}">
      <source src="{{ url options="articleattachment" }}" type='{{ $gimme-
>attachment->mime_type }}' />
      </video>
    {{ else }}
      <!-- if it is not audio or video file we will show simple download link -
->
      <a href="{{ url options="articleattachment" }}" >
      {{ $gimme->attachment->file_name }} ({{ $gimme->attachment->size_kb }}kb)
      </a>
    {{ /if }}
  {{ /list_article_attachments }}
{{ /if }}
```

VIDEO FOR EVERYBODY

Using the well documented and easily understood *Video for Everybody* approach (http://camendesign.co.uk/code/video_for_everybody) it's very easy to roll your own video delivery solution for your publication. This could be integrated into the above code if you preferred not to have to relay on video.js for content delivery.

```
[...]
{{ elseif $gimme->attachment->extension == ogv ||
             $gimme->attachment->extension == ogg ||
             $gimme->attachment->extension == mp4 ||
             $gimme->attachment->extension == webm }}

<!-- first try HTML5 playback: if serving as XML, expand 'controls' to
'controls="controls"'' and autoplay likewise -->
<!-- warning: playback does not work on iOS3 if you include the poster
attribute! fixed in iOS4.0 -->
<video controls>
<!-- MP4 must be first for iPad! -->
  {{ if $gimme->attachment->extension == mp4 }}<source src="{{ uri
options='articleattachment' }}" type="video/mp4" /><!-- Safari / iOS video --
>{{ /if }}
  {{ if $gimme->attachment->extension == ogg }}<source src="{{ uri
options='articleattachment' }}" type="video/ogg" /><!-- Firefox / Opera /
Chrome10 -->{{ /if }}
  {{ if $gimme->attachment->extension == ogv }}<source src="{{ uri
options='articleattachment' }}" type="video/ogg" /><!-- Firefox / Opera /
Chrome10 -->{{ /if }}
  {{ if $gimme->attachment->extension == webm }}<source src="{{ uri
options='articleattachment' }}" type="video/webm" /><!-- MS presumably -->{{
/if }}
```

```
</video>
{{ elseif $gimme->attachment->extension == flv }}
<!-- fallback to Flash: -->
<object type="application/x-shockwave-flash" data="{{ uri
options='articleattachment' }}">
<!-- Firefox uses the `data` attribute above, IE/Safari uses the param below --
>
  <param name="movie" value="{{ uri options='articleattachment' }}" />
  <param name="flashvars" value="controlbar=over&amp;file={{ uri
options='articleattachment' }}" />
</object>
<!-- you *should* offer a download link as they may be able to play the file
locally. customise this bit all you want -->
<p>
  <strong>Download Video:</strong>
  {{ if $gimme->attachment->extension == mp4 }} Closed Format: <a href="{{ uri
options='articleattachment' }}">"MP4"</a>{{ /if }}
  {{ if $gimme->attachment->extension == ogv }} Open Format: <a href="{{ uri
options='articleattachment' }}">"OGG"</a>{{ /if }}
</p>
```

MULTILINGUAL PUBLICATIONS AND TEMPLATES

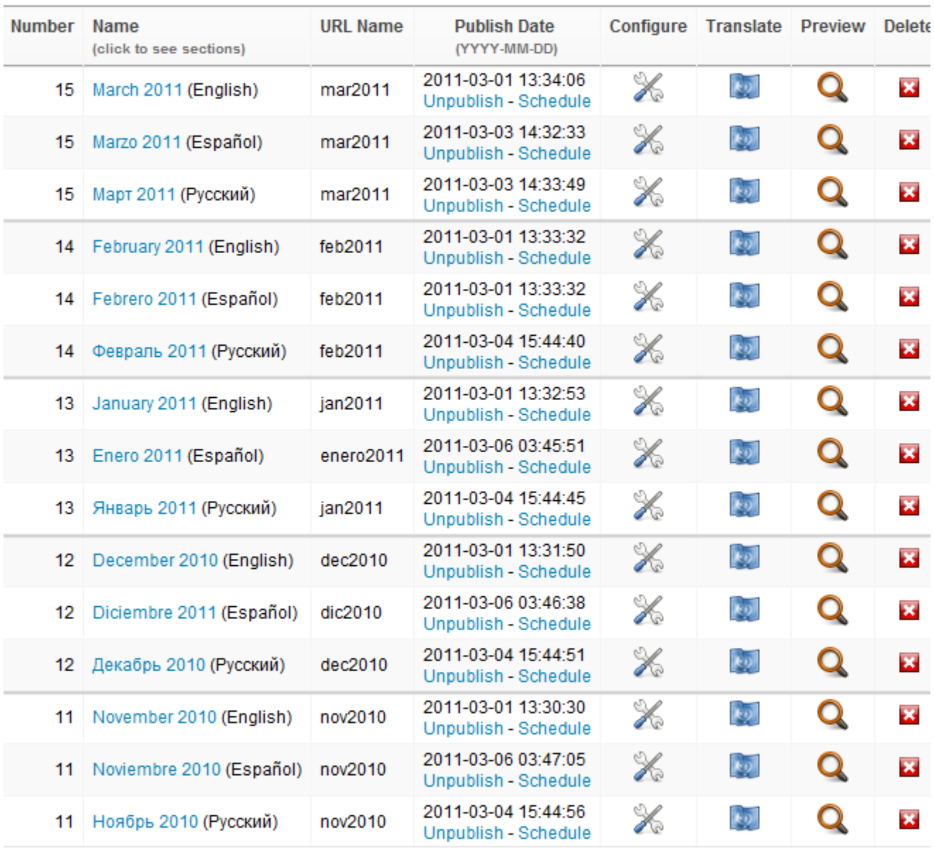
34. CONTENT IN MULTIPLE LANGUAGES

In this chapter, we're working with one of Newscoop's main advantages: the ability to easily create and manage multilingual publications. If you are running a multilingual site, this affects three parts of Newscoop:

1. Journalists and editors can translate content in the backend. (This chapter is dealing with how this is displayed in templates).
2. Strings in templates. The following chapter will show you how to translate content in the templates and display it for different languages.
3. Language specific elements on the site. A following chapter will show you how to modify the structure of your template according to the language the content is in.

MULTILINGUAL CONTENT IN THE ADMIN INTERFACE

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

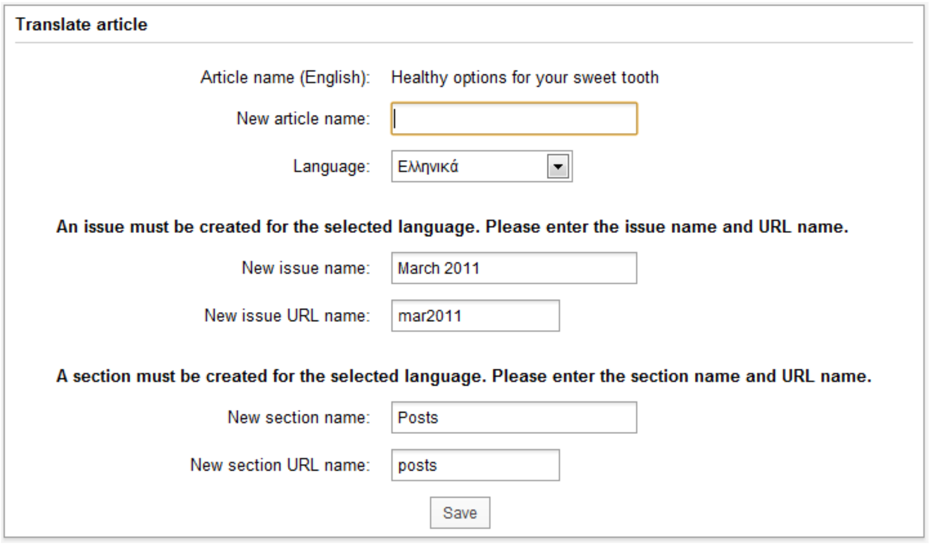
The most important difference regarding multilinguality lies in the language code (en, es, ru) and in the last part of the URL, generated using topics for SEO purposes. Incidentally, those links will work without that last part of the url. The article number (4) is enough to tell Newscoop which article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

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


Translate article

Article name (English):	Healthy options for your sweet tooth
New article name:	
Language:	Ελληνικά

An issue must be created for the selected language. Please enter the issue name and URL name.

New issue name:	March 2011
New issue URL name:	mar2011

A section must be created for the selected language. Please enter the section name and URL name.

New section name:	Posts
New section URL name:	posts

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.

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)

35. MULTILINGUAL STRINGS IN THE TEMPLATES

If you are running a site with more than one language, all content inside the articles can be translated by the editors. However, there are always strings in the templates which also need to switch according to the language.

This is the most straight forward way of handling this:

```
{{ if $gimme->language->english_name == "English" }}Search{{ /if }}
{{ if $gimme->language->english_name == "Spanish" }}Búsqueda{{ /if }}
[...]
```

You can also make the decision based on the language code in the system. This does not make any difference. The following example would work for a bilingual site and checks for Arabic first, else will load the other language:

```
{{ if $gimme->language->code == 'ar' }}
[...]
{{ else }}
[...]
{{ /if }}
```

This works well if you have few strings in the template. In order to translate all strings across all templates, create a configuration file and use the built-in smarty function {{ config_load }} (http://www.smarty.net/docs/en/language.function.config.load.tpl)

This way you achieve the complete separation of multilingual strings from template code. Every language can have it's own .conf file, for example English.conf, German.conf etc.

NOTE: these files MUST be inside the folder **_conf** which is inside your theme folder.

Depending on the language of the context, we load the appropriate file like this:

```
{{ config_load file="{{ $gimme->language->english_name }}.conf" }}
```

To display the strings which are set in the config file, you are using the hash tags like this:

```
<p class="subtitle">{{ #articlePublishedUnder# }} <em>{{ $gimme->topic->name
}}</em> {{ #category# }}</p>
```

You can see that you are not only using the $gimme command, but inside the # are the strings set in the configuration file.

And how does this improve your template? By separating the content from the template, it makes everything easier to maintain and scale. Take this tempate for example - category.tpl. As the example publication is localized into four languages (en, ru, es, de), it has 4 'if' conditions for every string.

```
{{ include file="_tpl/_html-head.tpl" }}
<body>
...
<h2>{{ $gimme->topic->name }}</h2>
<p class="subtitle">
  {{ if $gimme->language->english_name == "English" }}Article published under
<em>{{ $gimme->topic->name }}</em>Category{{ /if }}
  {{ if $gimme->language->english_name == "Russian" }}Статьи, опубликованные
под <em>{{ $gimme->topic->name }}</em>темы{{ /if }}
  {{ if $gimme->language->english_name == "Spanish" }}Los artículos publicados
en <em>{{ $gimme->topic->name }}</em>tema{{ /if }}
  {{ if $gimme->language->english_name == "German" }}Artikel zum Thema <em>{{
$gimme->topic->name }}</em>{{ /if }}
</p>
...
</body>
</html>
```

```
Whereas if we apply the new approach, the template looks like this:

{{ config_load file="{{ $gimme->language->english_name }}.conf"
section="Category" }}
{{ include file="_tpl/_html-head.tpl" }}
<body>
...
<h2>{{ $gimme->topic->name }}</h2>
<p class="subtitle">{{ #articlePublishedUnder# }} <em>{{ $gimme->topic->name
}}</em> {{ #category# }}</p>
...
</body>
</html>
```

EXAMPLE CONFIGURATION FILE

```
#this is the config file for English

# global strings
publishedOn = "Published on "
filedUnder = " Filed under "
comments = "Comments "
multiPOIs = "Map with geo-locations from listed articles"

#Strings for Category page
[Category]
articlePublishedUnder = "Articles published under"
category = "Category"

#Strings for Archive page
[Archive]
archive = "Archive"
```

Advantages of this approach:

1. Using {{ #langVariable# }} style syntax in templates is easier to read when developing templates, because it distinguishes language strings from other template code
2. No more "if" conditions inside your templates to display strings. This makes your code way more manageable.
3. Scalable solution. The more languages you add to your publication, you only need to add a new configuration file. Simply name .conf files like English.conf, German.conf etc and load them as in the above example.
4. Furthermore, .conf files can be structured in sections, so in a template you can decide to limit the strings loaded - for example {{ config_load file="English.conf" section="Archive" }}. In the case of a large number of strings, this is good for the server performance.
5. Translators don't need to edit templates. Just send the .conf file to a translator and include once the work is done.

36. DIFFERENT TEMPLATES FOR DIFFERENT LANGUAGES

Sometimes you might want to adjust the structure of your template accoring to the language the user has selected. A simple example: serving different advertisment for different languages. Even if you are using a third party banner management software, you might find that it does not cater for all languages you deliver to, e.g. Google Adsense does not deliver to Georgian languages. With Newscoop, you can make a switch depending on the language and call a template that delivers something else.

Another example might be that you are using an embed of facebook for the English pages, but vkontakte.ru for the Russian pages.

This is the most straight forward way of handling this:

```
{{ if $gimme->language->english_name == "English" }}whatever English thing you
want{{ /if }}
{{ if $gimme->language->english_name == "Spanish" }}the Spanish thing you
want{{ /if }}
[...]
```

This example will simply print the string, of course. Replace it with whatever you want, like {{ include ... }} and so forth.

You can also make the decision based on the language code in the system. This does not make any difference. The following example would work for a bilingual site and checks for Arabic first, else will load the other language:

```
{{ if $gimme->language->code == 'ar' }}
[...]
{{ else }}
[...]
{{ /if }}
```

37. TEXT DIRECTION

If you are creating a website that caters to an audience who don't have a left to right (ltr) direction written language such as Hebrew or Arabic you will need to specify the text direction in your markup.

Ever since HTML4 there has been support for non-latin languages in the form of the *dir* attribute.

A well formed use of the attribute is as follows:

```
<body dir="ltr">
```

for Arabic and Hebrew you would use the following instead:

```
<body dir="rtl">
```

Things get a little more complicated if you're building a site with support for both. There are a number of ways to approach this problem, one way is with the CSS *text-direction* property but this can prove to be more complicated than simply setting the direction in the template. This approach is also more accessible as a screen reader will know the language direction if you have already set the *lang* attribute.

You can set the direction automatically on detection of the non-latin languages as follows:

```
<body dir="{{ strip }}
  {{ if $gimme->language->code == "ar" || $gimme->language->code == "he" }}
    rtl
  {{ else }}
    ltr
  {{ /if }}
{{ /strip }}" lang="{{ $gimme->language->code }}">
```

Bear in mind that you may well have to mirror your design to accomodate the reading habits of your right to left audience.

SPECIAL NEWSCOOP FEATURES

38. 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:

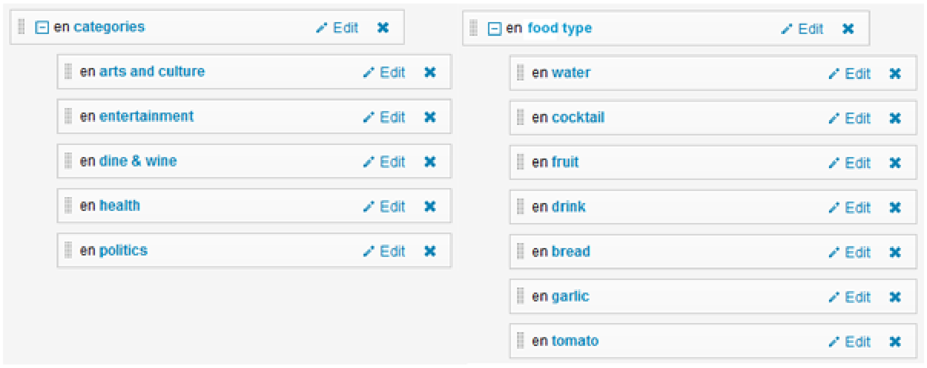
- An article can have more than one topic
- The topic tree structure allows variable depth of content
- Based on the position in the topic tree, similar articles can be grouped

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

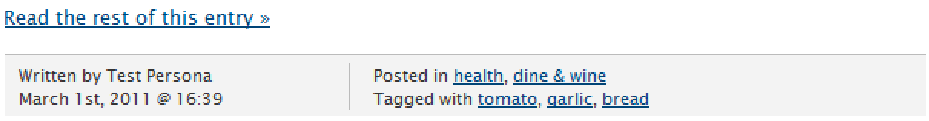
How you organize topics in a topic tree may be very important. It's usually a good approach to create a root topic for group of topics, and then make subtopics in that branch.



en categories Edit
- en arts and culture Edit
- en entertainment Edit
- en dine & wine Edit
- en health Edit
- en politics Edit

en food type Edit
- en water Edit
- en cocktail Edit
- en fruit Edit
- en drink Edit
- en bread Edit
- en garlic Edit
- en tomato Edit

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.



Read the rest of this entry »

Written by Test Persona
March 1st, 2011 @ 16:39

Posted in health, dine & wine
Tagged with tomato, garlic, bread

The code for this example is:

```
<div class="tags">
    <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_topics }}</p>
    <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">{{ $gimme->topic->name
}}</a>{{ if !$gimme->current_list->at_end }}, {{ /if }}{{ /list_article_topics
}}</p>
</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="bypublishdate 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:

```
{{ local }}
{{ unset_topic }}
..... your code ...
{{ /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.

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

<form action="" method="post">
    <label for="category">Category:</label>
    <select id="category" name="category"
onChange="MM_jumpMenu('parent',this,0)">
    {{ set_topic name="Category:en" }}
        <option selected>----- choose -----</option>
        {{ list_subtopics }}
        <option value="{{ url options="template category-page.tpl" }}">
{{ $gimme->topic->name }}</option>
        {{ /list_subtopics }}
    {{ unset_topic }}
    </select>
</form>
```

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

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

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

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

```
{{ if $gimme->article->custom_switch_name }}
 .... your code ...
{{ /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:

```
{{ list_articles constraints="onfrontpage is on" }}
{{ list_articles constraints="onsection is on" }}
```

or in 'if' clauses:

```
{{ if $gimme->article->on_front_page }}
{{ 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:

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

or in an article list:

```
{{ list_articles constraints="keyword organic" }}
...
{{ /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)*.

39. DYNAMIC PAGE LAYOUTS WITH SWITCHES

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"
constraints="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

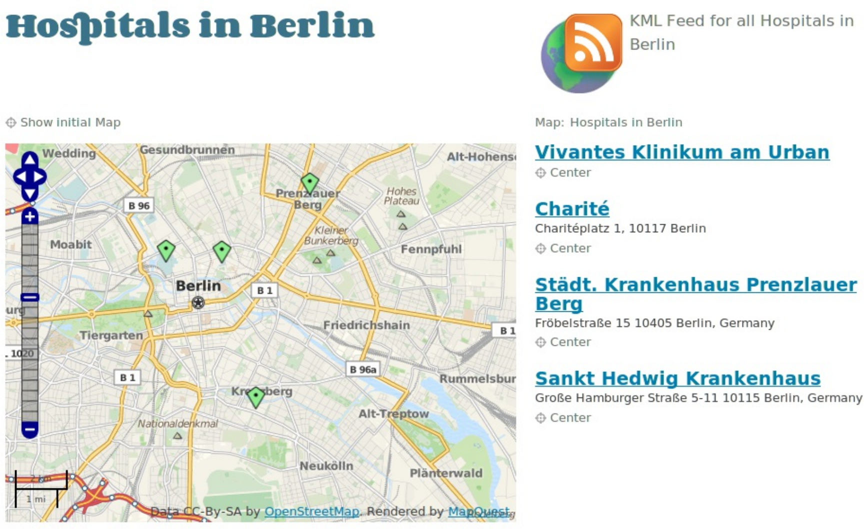
The whole template looks like this:

```
{{ list_articles length="1" ignore_section="true" order="bypublishdate desc"
constraints="breaking_news is on" }}
<div id="breakingNews">
    <h3><a href="{{ url options="article" }}">{{ $gimme->article->name
}}</a></h3>
    {{ list_article_images length="1" }}
    <div id="breakingNewsLeft">
        <img src="{{ url options="image width 435" }}" alt="{{ $gimme->article-
>image->description }}" />
        <p class="footnote">{{ $gimme->article->image->description }} {{ if
$gimme->article->image->photographer }}(Photo: {{ $gimme->article->image-
>photographer }}){{ /if }}</p>
    </div>
    {{ /list_article_images }}
    {{ if $gimme->prev_list_empty }}
    <div id="breakingNewsLeft">
        <img src="{{ url static_file="_img/{{ $gimme->language->code }}.jpg" }}
alt="Breaking news" />
    </div>
    {{ /if }}
    <div id="breakingNewsCenter">
        {{ $gimme->article->publish_date|camp_date_format:"%M %e, %Y" }}
        <p>{{ $gimme->article->deck }}</p>
    </div>
</div>
{{ /list_articles }}
```

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

40. 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).


Hospitals in Berlin
KML Feed for all Hospitals in Berlin
Show initial Map
Map: Hospitals in Berlin
Vivantes Klinikum am Urban
Center
Charité
Charitéplatz 1, 10117 Berlin
Center
Städt. Krankenhaus Prenzlauer Berg
Fröbelstraße 15 10405 Berlin, Germany
Center
Sankt Hedwig Krankenhaus
Große Hamburger Straße 5-11 10115 Berlin, Germany
Center
Data CC-By-SA by OpenStreetMap, Rendered by MapQuest

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 4 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'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 4 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 4 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 New Custodian" theme which includes a map inside the **article-map.tpl** sub-template:

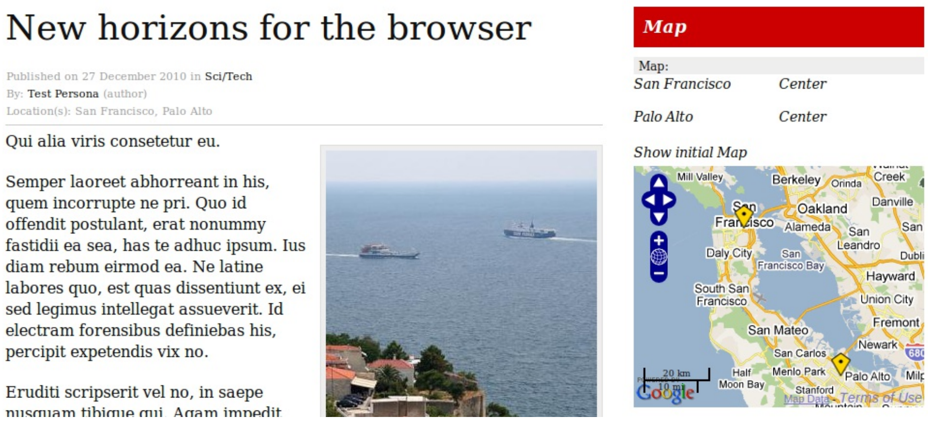
```
{{* this creates article map with markers for selected POIs *}}
{{ if $gimme->article->has_map }}
 <figure id="map-box">
 <h3>Map</h3>
 {{ map show_locations_list="false" show_reset_link="Show initial Map"
width="350" height="300" }}
 </figure>
{{ /if }}
```

In the code above we first validate if the current article has a map, and if so, we include the template rendering the map.

Inside the snippet above, there might have been one-line directive:

```
{{ map show_locations_list="true" show_reset_link="Show initial Map"
width="350" height="300" }}
```

This will display a 350px by 300px 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.


New horizons for the browser

Published on 27 December 2010 in Sci/Tech
By: Test Persona (author)
Location(s): San Francisco, Palo Alto

Qui alia viris consetetur eu.

Semper laoreet abhorreant in his, quem incorrupte ne pri. Quo id offendit postulant, erat nonummy fastidii ea sea, has te adhuc ipsum. Ius diam rebum eirmod ea. Ne latine labores quo, est quas dissentiunt ex, ei sed legimus intellegat assueverit. Id electram forensibus definiebas his, percipit expetendis vix no.

Eruditi scripserit vel no, in saepe nusquam tibique qui. Agam impedit

Map

Map:
San Francisco Center
Palo Alto Center

Show initial Map

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 4 for Journalists and Editors* manual.

LISTING ARTICLE LOCATIONS

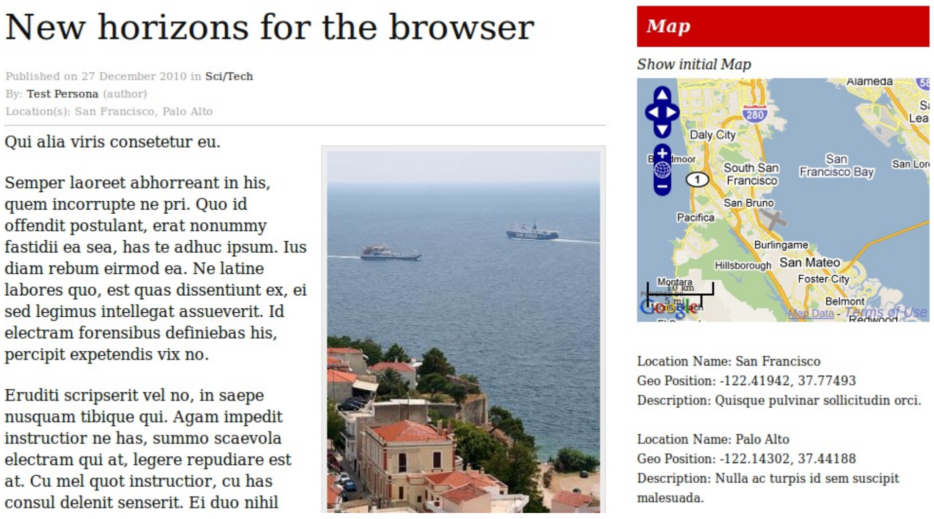
In the previous example, you saw that it's possible to display the list of locations together with the map, but it's a simple list. What if you want to display a list with some more data about each location? You might need to publish that data even without displaying the graphical map. For this purpose,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

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

```
  {{ if $gimme->location->enabled }}
    <p>
      Location Name: {{ $gimme->location->name }}<br />
      Geo Position: {{ $gimme->location->longitude }},
                    {{ $gimme->location->latitude }}<br />
      Description: {{ $gimme->location->text }}<br />
    </p>
  {{ /if }}
{{ /list_article_locations }}
```

And the output looks like this:


New horizons for the browser

Published on 27 December 2010 in Sci/Tech
By: Test Persona (author)
Location(s): San Francisco, Palo Alto

Qui alia viris consetetur eu.

Semper laoreet abhorreant in his, quem incorrupte ne pri. Quo id offendit postulant, erat nonummy fastidii ea sea, has te adhuc ipsum. Ius diam rebum eirmod ea. Ne latine labores quo, est quas dissentiunt ex, ei sed legimus intellegat assueverit. Id electram forensibus definiebas his, percipit expetendis vix no.

Eruditi scripserit vel no, in saepe nusquam tibique qui. Agam impedit instructior ne has, summo scaevola electram qui at, legere repudiare est at. Cu mel quot instructior, cu has consul delenit senserit. Ei duo nihil

Map

Show initial Map

Location Name: San Francisco
Geo Position: -122.41942, 37.77493
Description: Quisque pulvinar sollicitudin orci.

Location Name: Palo Alto
Geo Position: -122.14302, 37.44188
Description: Nulla ac turpis id sem suscipit malesuada.

MULTI-MAPS

Important improvements to mapping were developed for Newscoop 4, most of them under the concept of *Multi-maps*.

When you edit 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 interest) as you want. All those locations are stored in the database and can be displayed in article pages, but in Newscoop 3 only together with the article those points are related to.

The concept of multi-maps in Newscoop 4 goes far beyond this. You can set a map using several different options, grouping multiple locations regardless of which article they belong to. It is called multi-maps because a single map can display locations from multiple articles, but also because in a single article you will be able to display multiple maps.

It is 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 is 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.

This code below is taken from The New Custodian's _tpl/front-dynamicmap.tpl template:

```
{{ set_map
 label="Latest locations"
 issues="_current"
 max_points=5
}}


{{ map
 show_locations_list=false
 show_reset_link=true
 area_show="focus"
 width="754"
 height="250"
 show_open_link=true
 open_map_on_click=false
 popup_width="1000"
 popup_height="750"
 max_zoom=15
 map_margin=20
 area_show="focus"
}}
</div>
```

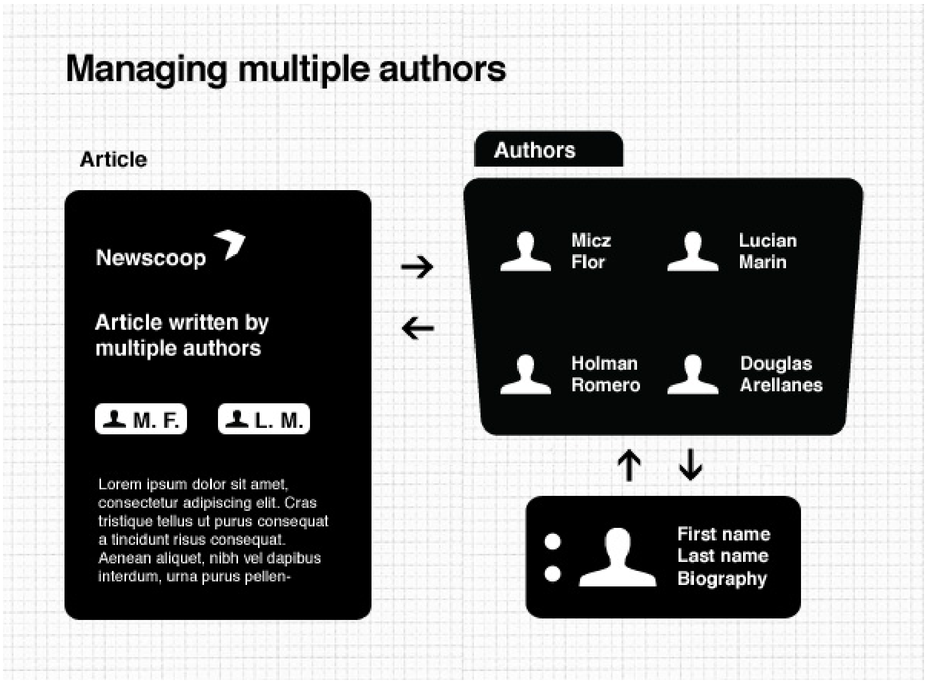
In this example, we want to show latest 5 locations from current issue, and then to make a list of these five locations with max 3 belonging articles (those that are geo-located to specific locations)

```
<div class="fourcol last location-list">
 <h4>{{ #articlesLatestLocations# }}</h4>
 {{ list_map_locations }}
 {{ assign var="latitude" value=$gimme->location->latitude }}
 {{ assign var="longitude" value=$gimme->location->longitude }}
 <div class="collapsible"><h6><i></i>{{ $gimme->location->name }}</h6></div>

 {{ list_articles length="3" ignore_issue="true" ignore_section="true"
order="bypublishdate desc" location="$latitude $longitude, $latitude
$longitude" constraints="type is news" }}
 {{ if $gimme->current_list->at_beginning }}
 <div class="location-content">
 <ul>{{ /if }}
 <li><a href="{{ url options="article" }}">{{ $gimme->article->name }}</a></li>
 {{ if $gimme->current_list->at_end }}</ul>
 </div>{{ /if }}
 {{ /list_articles }}

 {{ /list_map_locations }}
```

Sounds exciting? Actually, it is!

41. 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.


Managing multiple authors
Article
Newscoop
Article written by multiple authors
M. F.
L. M.
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras tristique tellus ut purus consequat a tincidunt risus consequat. Aenean aliquet, nibh vel dapibus interdum, urna purus pellen-
Authors
Micz Flor
Lucian Marin
Holman Romero
Douglas Arellanes
First name
Last name
Biography

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.

```
{{list_article_authors}}
<!-- lets divide authors with comma -->
{{if $gimme->current_list->index!=1}},&nbsp;{{/if}}

<!-- code below creates an url to author profile if author is linked with user
in Newscoop Admin Panel. You can read more about it in Author and User profiles
section -->
{{ if $gimme->author->user->defined }}
<a href="{{ $view->url(['username' => $gimme->author->user->uname], 'user')
}}">
{{ /if }}

<!-- we want to show author name from biography because it is translateable -->
{{ if $gimme->author->biography->first_name }}
{{ $gimme->author->biography->first_name }} {{$gimme->author->biography-
>last_name }}
{{ else }}
{{ $gimme->author->name }}
{{ /if }}

{{ if $gimme->author->user->defined }}
</a>
{{ /if }}

({{ $gimme->author->type|lower }})
<img src="{{$gimme->author->picture->imageurl}}" alt="author picture" />
{{$gimme->author->email}}
{{$gimme->author->biography->text}}
{{/list_article_authors}}
```

42. 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 - an example code for such behaviour you can see at the end of this chapter. 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

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

Firefox 3.6.16

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

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

```
}}.
<body class="{{ textformat wrap=200 }}
{{ $gimme->browser->browser_working }}
{{ $gimme->browser->ua_type }}
{{ $gimme->browser }}
{{* mobile device / OS *}}
  {{ $gimme->browser->mobile_data.0 }}
  {{ $gimme->browser->mobile_data.3|regex_replace:"/\ /":"" }}
  {{ strip }}{{ $gimme->browser->mobile_data.3|regex_replace:"/\ /":"" }}
  {{ $gimme->browser->mobile_data.4|regex_replace:"/[\.][0-9]*/":"" }}{{ /strip
}}
  {{ strip }}{{ $gimme->browser->mobile_data.3|regex_replace:"/\ /":"" }}
  {{ $gimme->browser->mobile_data.4|regex_replace:"/\./":"-" }}{{ /strip }}
{{* firefox / gecko *}}
  {{ $gimme->browser->moz_data.0 }}
  {{ strip }}{{ $gimme->browser->moz_data.0 }}
  {{ $gimme->browser->moz_data.1|regex_replace:"/[\.][0-9]*/":"" }}{{ /strip }}
  {{ strip }}{{ $gimme->browser->moz_data.0 }}
  {{ $gimme->browser->moz_data.1|regex_replace:"/[\.][0-
9]*$/":""|regex_replace:"/\./":"-" }}{{ /strip }}
  {{ strip }}{{ $gimme->browser->moz_data.0 }}
  {{ $gimme->browser->moz_data.1|regex_replace:"/\./":"-" }}{{ /strip }}
  {{ if $gimme->browser->browser_working == "moz" }}
  gecko{{ $gimme->browser->moz_data.2|regex_replace:"/[\.][0-9]*/":"" }}
  gecko{{ $gimme->browser->moz_data.2|regex_replace:"/\./":"-" }}
  {{ /if }}
{{* internet explorer *}}
  {{ if $gimme->browser->browser_working == "ie" }}
    {{ strip }}{{ $gimme->browser->browser_name }}
    {{ $gimme->browser->webkit_data.2|regex_replace:"/[\.][0-9]*/":"" }}{{
/strip }}
    {{ strip }}{{ $gimme->browser->browser_name }}
    {{ $gimme->browser->webkit_data.2|regex_replace:"/\./":"-" }}{{ /strip }}
    {{ if $gimme->browser->webkit_data.2 > 8 }}chakra{{ else }}trident{{ /if }}
  {{ /if }}
{{* chrome *}}
  {{ if $gimme->browser->browser_working == "webkit" }}
    {{ strip }}{{ $gimme->browser->webkit_data.0 }}
    {{ $gimme->browser->webkit_data.1|regex_replace:"/[\.][0-9]*/":"" }}{{
/strip }}
    {{ strip }}{{ $gimme->browser->webkit_data.0 }}
    {{ $gimme->browser->webkit_data.1|regex_replace:"/\./":"-" }}{{ /strip }}
  {{ /if }}
{{ /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**:

```
{{* gecko / firefox ***********************************}}
{{ if $gimme->browser->browser_working == "moz" }}
  {{ assign var="browserdetect_name" value=`$gimme->browser->moz_data.0` }}
  {{ assign var="browserdetect_version" value=`$gimme->browser->moz_data.1` }}
  {{ assign var="browserdetect_engineversion" value=`$gimme->browser-
>moz_data.2` }}
  {{ assign var="browserdetect_engine" value="gecko" }}
{{ /if }}
{{* webkit / chrome / safari  ***********************************}}
{{ if $gimme->browser->browser_working == "webkit" }}
  {{ assign var="browserdetect_engineversion" value=`$gimme->browser-
>browser_number` }}
  {{ assign var="browserdetect_name" value=`$gimme->browser->webkit_data.0` }}
  {{ assign var="browserdetect_version" value=`$gimme->browser->webkit_data.1`
}}
  {{ assign var="browserdetect_engine" value="webkit" }}
{{ /if }}
{{* ie / internet explorer  ***********************************}}
```

```
{{ if $gimme->browser->browser_working == "ie" }}
  {{ assign var="browserdetect_name" value=`$gimme->browser->browser_name` }}
  {{ assign var="browserdetect_engineversion" value=`$gimme->browser-
>webkit_data.2` }}
  {{ assign var="browserdetect_version" value=`$gimme->browser->webkit_data.2`
}}
  {{ if $gimme->browser->webkit_data.2 > 8 }}
    {{ assign var="browserdetect_engine" value="chakra" }}
  {{ else }}
    {{ assign var="browserdetect_engine" value="trident" }}
  {{ /if }}
{{ /if }}
{{* bot / search engine  ********************************}}
{{ if $gimme->browser->ua_type == "bot" }}
  {{ assign var="browserdetect_name" value=`$gimme->browser->browser_name` }}
  {{ assign var="browserdetect_version" value=`$gimme->browser-
>browser_math_number` }}
  {{ assign var="browserdetect_engine" value="false" }}
  {{ assign var="browserdetect_engineversion" value="false" }}
{{ /if }}
{{* mobile devices ***************************************}}
{{ if $gimme->browser->ua_type != "mobile" }}
  {{ assign var="browserdetect_mobile_device" value="false" }}
  {{ assign var="browserdetect_mobile_os" value="false" }}
  {{ assign var="browserdetect_mobile_os_number" value="false" }}
{{ else }}
  {{ assign var="browserdetect_mobile_device" value=`$gimme->browser-
>mobile_data.0` }}
  {{ assign var="browserdetect_mobile_os" value=`$gimme->browser-
>mobile_data.3` }}
  {{ assign var="browserdetect_mobile_os_number" value=`$gimme->browser-
>mobile_data.4` }}
{{ /if }}
```

Save the code snippet above inside the **_tpl** folder of your theme package. Include it like this:

```
{{ include file="_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:

```
<ul>
  <li>Browser: {{ $browserdetect_name }}, version {{ $browserdetect_version }}
    | Engine: {{ $browserdetect_engine }}, version {{
$browserdetect_engineversion }}</li>
  <li>Mobile:
    {{ if $browserdetect_mobile_device == "false" }}no
    {{ else }}
      device: {{ $browserdetect_mobile_device }} |
      OS: {{ $browserdetect_mobile_os }} |
      OS version: {{ $browserdetect_mobile_os_number }}
    {{ /if }}</li>
<li>Bot: {{ if $gimme->browser->ua_type == "bot" }}yes{{ else }}no{{ /if
}}</li>
<li>Type: {{ $gimme->browser->ua_type }} =
  {{ if $gimme->browser->ua_type == "bot" }}web bot{{ /if }}
  {{ if $gimme->browser->ua_type == "bro" }}normal browser{{ /if }}
  {{ if $gimme->browser->ua_type == "bbro" }}simple browser{{ /if }}
  {{ if $gimme->browser->ua_type == "mobile" }}handheld{{ /if }}
  {{ if $gimme->browser->ua_type == "dow" }}downloading agent{{ /if }}
  {{ if $gimme->browser->ua_type == "lib" }}http library{{ /if }}
</li>
```

LETTING GOOGLE AND OTHER BOTS SEE CONTENT BEHIND YOUR PAYWALL

Setting up your site to allow access to your subsrciber content is simple with browser detection. As we're only looking for two results, whether the viewer is a bot or a paying customer, we've wrapped both of the requests into a single if statement with the or operator "||". If the detection fails on either count this snippet drops the client to

```
{{ if $gimme->browser->ua_type == "bot" || $gimme->article->content_accessible
}}
  <h2>{{ $gimme->article->title}}</h2>
  <div>{{ $gimme->article->full_text }}</div>
{{ else }}
  <p>You are not authorised to view this article. Please consider paying for a
subscription.</p>
{{ /if }}
```

43. 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:

```
<link rel="stylesheet" type="text/css" href="...mobile.css" media="handheld"/>
<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:

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

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

USING MEDIA QUERIES IN YOUR SITE

With modern CSS3 capable browsers you can also make use of the media query selector. Media queries allow you to change your design to the context with which it's being viewed. There are two main methods with which to call media queries, a single file or via includes.

In your CSS you can specify which CSS file to call in each specific circumstance. For example in the context of an iPad or other tablet with a resolution less than a modern computer but more than that of a mobile telephone.

The code can either be placed in a stylesheet, which we advise, or in the <head> of your site.

CSS RULES WITHIN A CONTEXT SENSITIVE MEDIA QUERY

```
@media only screen (and min-width: 768px) and (max-width: 1024px) {
  body {
    background-color: #000;
  }
}
```

INCLUDE A CSS FILE AS AN INDIVIDUAL INCLUDE

```
@media only screen (and min-width: 768px) and (max-width: 1024px) {
  @include url(ipad.css);
}
```

44. CREATING DEBATE FUNCTIONALITY

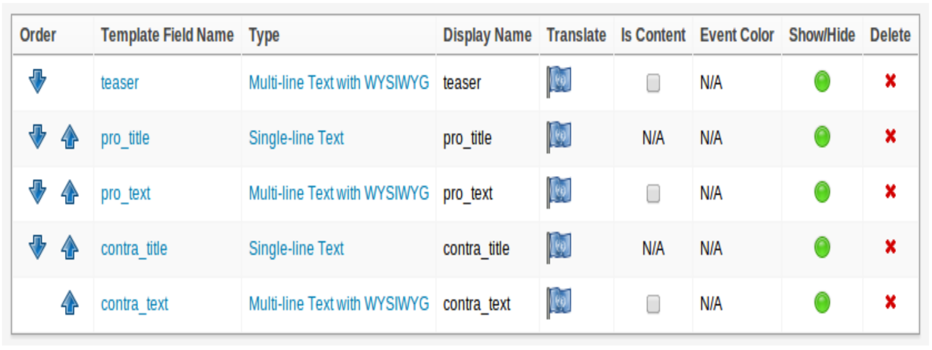
By 'debate functionality' in Newscoop we mean type of content where two sides (invited authors) cross their *pro* and *contra* arguments on some topic proposed by editorial team, and site users are able to vote for one of two options (yes or no) in limited timeframe (one week).

This functionality was firstly developed for Tages Woche (Basel), and had been wider accepted since then. It is also included in Newscoop's sample themes. It requires Newscoop plugin 'debate' to be enabled.

Solution shown here is just one way to implement debates.

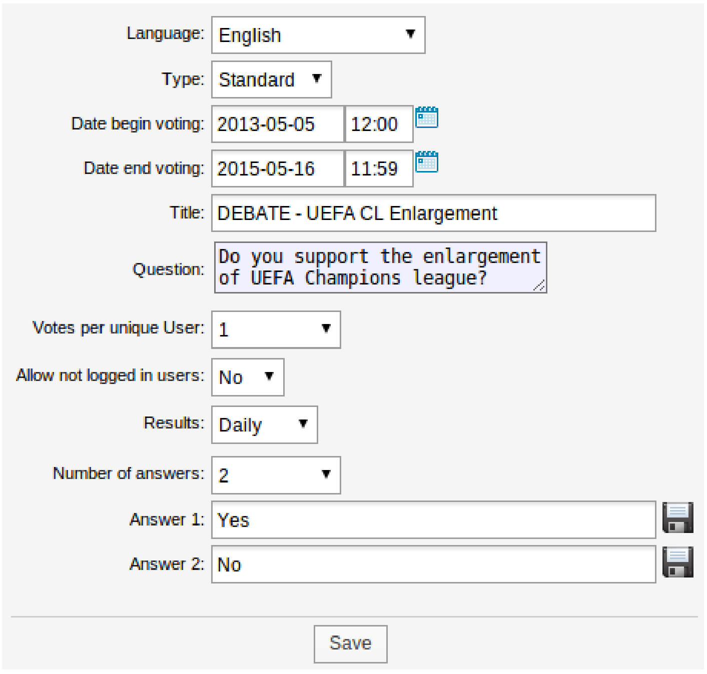
What you need

Firstly, new article type. We are going to call it 'debate', and fields should provide options to submit text by two authors, as well as introductory piece from editors.



Order	Template Field Name	Type	Display Name	Translate	Is Content	Event Color	Show/Hide	Delete
⇩	teaser	Multi-line Text with WYSIWYG	teaser		☐	N/A	●	✘
⇩ ⇧	pro_title	Single-line Text	pro_title		N/A	N/A	●	✘
⇩ ⇧	pro_text	Multi-line Text with WYSIWYG	pro_text		☐	N/A	●	✘
⇩ ⇧	contra_title	Single-line Text	contra_title		N/A	N/A	●	✘
⇧	contra_text	Multi-line Text with WYSIWYG	contra_text		☐	N/A	●	✘

Such article, then, gets appropriate debate poll attached. To create debate poll, we need to go to plugin interface. Debate poll settings can look like this



Language:	English
Type:	Standard
Date begin voting:	2013-05-05 12:00
Date end voting:	2015-05-16 11:59
Title:	DEBATE - UEFA CL Enlargement
Question:	Do you support the enlargement of UEFA Champions league?
Votes per unique User:	1
Allow not logged in users:	No
Results:	Daily
Number of answers:	2
Answer 1:	Yes
Answer 2:	No

Debate templates

Showing content from the article is easy -

```
<p><em>{{ $gimme->article->teaser }}</em></p>

<h3>PRO: {{ $gimme->article->pro_title }}</h3>
 <div>{{ $gimme->article->pro_text }}</div>

<h3>CONTRA: {{ $gimme->article->contra_title }}</h3>
 <div>{{ $gimme->article->contra_text }}</div>
```

Working with debate poll, however, is little bit less trivial. First there is main debate poll holding template. The code is self-explanatory

```
{{ list_debates length="1" item="article" }}

<div id="debate">

{{ include file="_tpl/debate_votes_total.tpl" scope="parent" }}

{{ if $gimme->debate->is_votable }}
  {{ $smarty.capture.votes }}
  {{ include file="_tpl/debate-deadline.tpl" }}
{{ /if }}


{{ if $gimme->default_article->defined }}
 <div class="vote-box">
 <div class="button-group">
 {{ if $gimme->debate->is_votable }}
  {{ debate_form template="article.tpl" submit_button=false }}
   {{ list_debate_answers order="bynumber asc" }}

<a onclick="$('#answer-{{ $gimme->debateanswer->number
}}').attr('checked','checked');$(this).parents('form:eq(0)').submit(); return
false;" href="javascript:void(0)" class="button debbut">{{ $gimme-
>debateanswer->answer }}</a>

 <!-- f_debateanswer_nr name mandatory -->
 <input type="radio" name="f_debateanswer_nr"
 value="{{ $gimme->debateanswer->number }}" id="answer-{{ $gimme->debateanswer-
>number }}"
 onclick="$(this).parents('form:eq(0)').submit();" style="display:none" />

{{ /list_debate_answers }}
 <input type="submit" id="submit-debate" class="button" value="submit" />
 {{ /debate_form }}
 {{ /if }}


 </div>
 </div>
 {{ if $gimme->debate->is_votable }}<small>You can change your mind as many
times as you wish until debate is closed</small>
 {{ elseif $gimme->user->logged_in or !$gimme->debate->is_current }}<small>This
debate is closed and result was decided</small>
 {{ elseif $gimme->debate->is_current && !$gimme->user->logged_in
}}<small>Please login to vote</small>{{ /if }}
{{ /if }}

</div>
 {{ /list_debates }}
```

debate-deadline.tpl is checking if and when debate is expiring

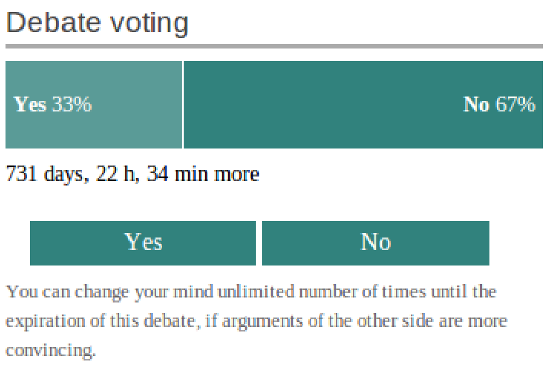
```
{{ $timestamp = sprintf('@%d', $gimme->debate->date_end) }}
{{ $closingdate=date_create($timestamp) }}
{{ $deadline=$closingdate->setTime(12, 0) }}
{{ $diff=date_diff($deadline, date_create('now')) }}
{{ if $deadline->getTimestamp() > time() }}
 <p>{{ $diff->days }} days, {{ $diff->h }} hours, {{ $diff->i }} minutes more
{{ if $gimme->article->comment_count }}<span class="comm">{{ $gimme->article-
>comment_count }}</span>{{ /if }}</p>
{{ else }}
 <p>Discussion closed on {{ $deadline->format('j.n.Y') }} at noon {{ if $gimme-
>article->comment_count }}<a href="{{ url}}#comments"><span class="comm">{{
$gimme->article->comment_count }}</span></a>{{ /if }}</p>
{{ /if }}
```

Finally, template debate_votes _total.tpl is dealing with poll results - calculating them and outputting back to the page

```
{{ $answers = array() }}
{{ list_debate_answers order="bynumber asc" }}
```

```
 {{ if empty($answers) }}
 {{ $percent = floor($gimme->debateanswer->percentage) }}
 {{ else }}
 {{ $percent = ceil($gimme->debateanswer->percentage) }}
 {{ /if }}
 {{ $answers[] = ['answer' => $gimme->debateanswer->answer, 'percent' =>
$percent] }}
{{ /list_debate_answers }}

{{ capture name="votes" }}
 <ul class="debatte-score">
 {{ strip }}
 {{ foreach $answers as $answer }}
 <li style="width:{{ $answer.percent }}%;" class="{{ if $answer@first }}yes{{
else }}no{{ /if }}"><span><b>{{ $answer.answer|escape }}</b> {{ $answer.percent
}}%</span></li>
 {{ /foreach }}
 {{ /strip }}
 </ul>
{{ /capture }}

{{ if !$gimme->debate->is_votable }}
 {{ $smarty.capture.votes }}
 <small>{{ if $gimme->debate->is_current && !$gimme->user->logged_in }}Current
result{{ else }}Final result{{ /if }}</small>
{{ /if }}
```

On the fronend, this can be represented in different ways. Some of real-life solutions look like this:

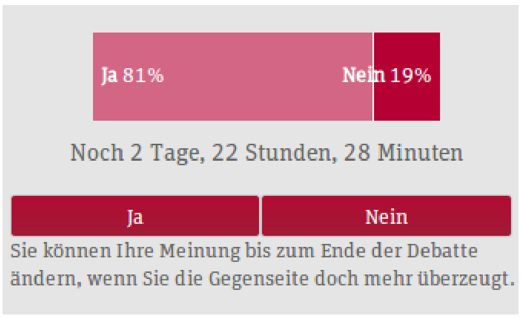

Debate voting

Yes 33% No 67%

731 days, 22 h, 34 min more

You can change your mind unlimited number of times until the expiration of this debate, if arguments of the other side are more convincing.

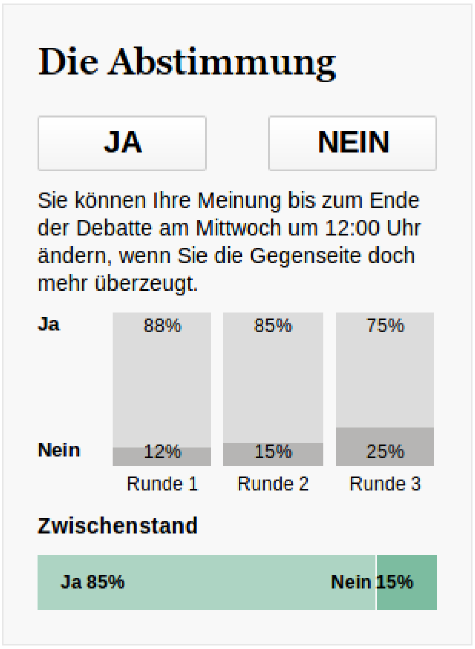
New Custodian


Ja 81% Nein 19%

Noch 2 Tage, 22 Stunden, 28 Minuten

Sie können Ihre Meinung bis zum Ende der Debatte ändern, wenn Sie die Gegenseite doch mehr überzeugt.

Zentral+



Die Abstimmung

JA NEIN

Sie können Ihre Meinung bis zum Ende der Debatte am Mittwoch um 12:00 Uhr ändern, wenn Sie die Gegenseite doch mehr überzeugt.

	Runde 1	Runde 2	Runde 3
Ja	88%	85%	75%
Nein	12%	15%	25%

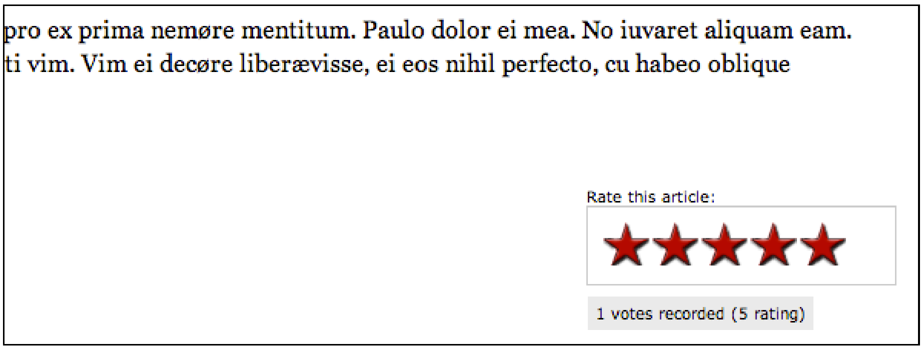
Zwischenstand

Ja 85% Nein 15%

TagesWoche

45. ARTICLE RATING BY READERS

As of version 4.1, Newscoop includes a new feature that allows logged in users to rate articles. The article rating scores are then displayed on the article pages for all users.


pro ex prima nemøre mentitum. Paulo dolor ei mea. No iuvaret aliquam eam.
ti vim. Vim ei decøre liberævisse, ei eos nihil perfecto, cu habeo oblique

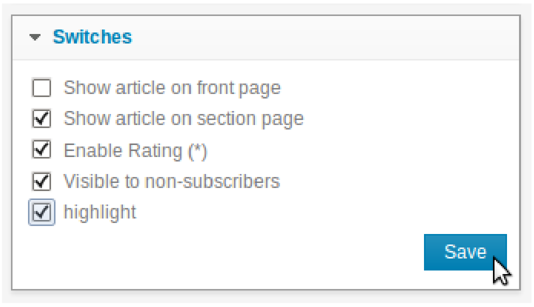
Rate this article:

1 votes recorded (5 rating)

HOW IT WORKS

The rating controller provides two actions to retrieve and update article ratings respectively. The show action takes one parameter for the articles unique article number and returns a JSON object with details about the articles current rating statistics. The save action take one parameter for the article number, and another for the new rating score. A JSON object is returned with the updated rating statistics for the article with a field containing an errors that may have been thrown. Save actions require that requests come from logged in Users.

The Newscoop admin has been updated with a default article switch to enable/disable article rating at the article level, and the article info box will now display the current avg article rating.


Switches

- [ ] Show article on front page
- [x] Show article on section page
- [x] Enable Rating (*)
- [x] Visible to non-subscribers
- [x] highlight

EXAMPLE IMPLEMENTATION

A 5 star rating example has been added to the the New Custodian Theme. To see the example simple apply the theme to your publication in the newscoop admin (CONFIGURE -> Themes) and navigate to any article page on the frontend. You will see the 5 star rating widget at the bottom of the article page, just above the share links and comments section.

To include the article 5 star rating widget in your own custom templates you simply need to include the rating template (_tpl/article-rating.tpl) where ever you want the widget to display:

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

CUSTOMIZING THE 5 STAR RATING WIDGET

If you want to customize the rating widget you can find the HTML, Javascript, and CSS below:

article-rating.tpl

```
<link rel="stylesheet" href="rating.css">
<script src="article-rating.js">

<div class='article_rating'>
    Rate this article:
    <div id="{{ $gimme->article->number }}">
        <div></div>
        <div></div>
        <div></div>
        <div></div>
        <div></div>
        <div>vote data</div>
        <div></div>
    </div>
</div>
```

article-rating.js

```
<script type='text/javascript'>

$(document).ready(function() {

   // gets the initial rating
    $('.rate_widget').each(function(i) {
        var widget = this;
        var out_data = {
            f_article_number : $(widget).attr('id')
        };
        $.post(
            '/rating/show',
            out_data,
            function(INFO) {
                $(widget).data( 'fsr', INFO );
                set_votes(widget);
            },
            'json'
        );
    });


    // Handles the mouseover
    $('.ratings_stars').hover(
        function() {
            $(this).prevAll().andSelf().addClass('ratings_over');
            $(this).nextAll().removeClass('ratings_vote');
        },
        // Handles the mouseout
        function() {
            $(this).prevAll().andSelf().removeClass('ratings_over');
            // can't use 'this' because it wont contain the updated data
            set_votes($(this).parent());
        }
    );

    // actually records the vote
    $('.ratings_stars').bind('click', function() {
        var star = this;
        var widget = $(this).parent();
        var score = $(star).attr('class').match(/star_(\d+)/)[1];

        var clicked_data = {
            f_rating_score : score,
            f_article_number : $(star).parent().attr('id')
        };
        $.post(
            '/rating/save',
            clicked_data,
            function(INFO) {
                widget.data( 'fsr', INFO );
                set_votes(widget);
            },
            'json'
        );
    });



});

// binds the rating data returning by the rating controller
// to the widget elements
function set_votes(widget) {

    var avg = $(widget).data('fsr').whole_avg;
    var votes = $(widget).data('fsr').number_votes;
    var exact = $(widget).data('fsr').dec_avg;
    var error = $(widget).data('fsr').error;

    $(widget).find('.star_' +
avg).prevAll().andSelf().addClass('ratings_vote');
    $(widget).find('.star_' + avg).nextAll().removeClass('ratings_vote');
    $(widget).find('.total_votes').text( votes + ' votes recorded (' + exact +
' rating)' );
    $(widget).find('.rating_error').text( error );
}

</script>
```

rating.css

```
.rate_widget {
    border:     1px solid #CCC;
    overflow:   visible;
    padding:    10px;
    position:   relative;
    width:      180px;
    height:     30px;
}
.ratings_stars {
    background: url('../_img/rating/star_empty.png') no-repeat;
    float:      left;
    height:     30px;
    padding:    2px;
    width:      28px;
}
.ratings_vote {
    background: url('../_img/rating/star_full.png') no-repeat;
}
.ratings_over {
    background: url('../_img/rating/star_highlight.png') no-repeat;
}

.total_votes {
    background: #eaeaea;
    top: 58px;
    left: 0;
    padding: 5px;
    position:   absolute;
}
.article_rating {
    font: 10px verdana, sans-serif;
    margin: 0px 0px 50px 550px;
    width: 180px;
}
.rating_error {
    top: 75px;
    left: 0;
    padding: 5px;
    position: absolute;
    color: red;
    font-weight: bold;
    width: 200px;
}
```

THE RATING CONTROLLER

The rating controller is accessible from [domain]/rating url and implements two available actions.

SHOW ACTION

parameters:

name	description
f_article_number	the unique article number

example request:

```
 http://localhost/rating/show?f_article_number=101
```

example results:

```
{"widget_id":"101","number_votes":1,"total_score":3,"dec_avg":3,"whole_avg":3}
```

SAVE ACTION

parameters:

name	description
f_article_number	the unique article number
f_rating_score	the desired rating

example request:

```
http://localhost/rating/save?f_article_number=101&f_rating_score=5
```

example results:

```
{"widget_id":"101","number_votes":1,"total_score":5,"dec_avg":5,"whole_avg":5,"
rating has been updated"}
```

46. USING JAVASCRIPT 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.

But You are not limited only to Newscoop templating system. You can use every javascript library You want or/and write Your own code.

INCLUDING 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 (prefarably in _js folder).

So the first thing to do will be to include all required jQuery libraries in the html header:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Language" content="en" />
  ...

<!-- You can use Google's CDN -->
  <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.5.2/jquery.min.js"></script>
  <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.11/jquery-
ui.min.js"></script>
  <link type="text/css"
href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.11/themes/base/jquery-
ui.css" rel="stylesheet" />

<!-- or include js scripts from local folder -->
  <script src="{{ url static_file='_js/jquery.js' }}"
type="text/javascript"></script>
  ...
</head>
```

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:

```
{{ php }}
  // including a php script directly from the template.
  include('/path/to/display_weather.php');
{{ /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 = $template->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

```
{{* a {{php}} block that assigns the variable $varX *}}
{{php}}
  $template->assign('varX','Toffee');
{{/php}}
{{* output the variable in the template *}}
<strong>{{$varX}}</strong> is my favourite ice cream :-)
```

USERS, AUTHORS AND COMMUNITY FEATURES

47. CONTENT ACCESS, AUTHORIZATION & PASSWORD RETRIEVAL

Newscoop templates can be constructed in a way that they deliver content depending on the status of the user (logged in or not). 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:

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

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

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

```
{{ $gimme->article->ad_text }}<br/>Contact:
{{ if $gimme->article->content_accessible }}
  <a href="mailto:{{ $gimme->article->ad_email }}">{{ $gimme->article->ad_email
}}</a>
{{ else }}
  Subscribe for contact information.
{{ /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 users
- Register option for new users
- Account page for users to check or change their personal information, including changing their password
- Option to resend lost or forgotten passwords
- Logout option

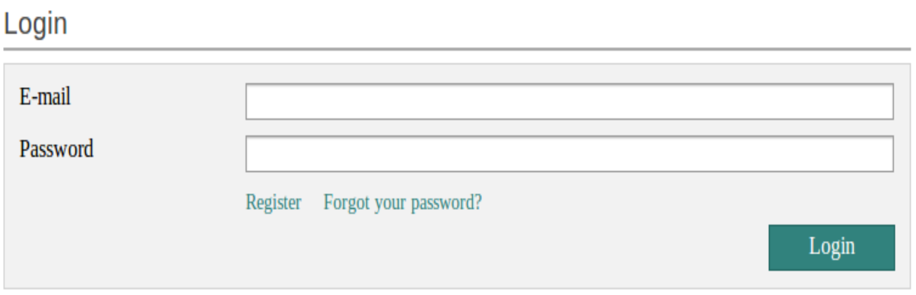
User management in Newscoop now uses Zend framework controller's power. Templates with predefined names (as it is done in The New Custodian) should be put in theme's root folder. If they are not there, default Zend controller templates will be used, from the Newscoop instance folder /application/views/scripts.

AUTHORIZATION TEMPLATES

Code for frontpage, taken from The New Custodian's _tpl/sidebar-loginbox.tpl, looks like this:

```
{{ dynamic }}
 {{ if $gimme->user->logged_in }}
 <p>Welcome, <a href="{{ $view->url(['controller' => 'dashboard', 'action' =>
'index'], 'default') }}">{{ $gimme->user->first_name }} {{ $gimme->user-
>last_name }}</a> | <a href="{{ $view->url(['controller' => 'auth', 'action' =>
'logout'], 'default') }}?t={{ time() }}">Logout</a></p>
 {{ else }}
 <li><a href="{{ $view->url(['controller' => 'auth', 'action' =>'index'],
'default') }}">Login and get involved!</a></li>
 {{ /if }}
 {{ /dynamic }}
```

The first thing that a reader needs is the option to login or register. On front page we can link to authorization page (as in the example above), or put form there directly. In The New Custodian case, we have special page for login and register actions, and it's code is as follows (from template auth_index.tpl)



```
<form action="{{ $form->getAction() }}" method="{{ $form->getMethod() }}">

 <fieldset>
 {{ if $form->isErrors() }}
 <div>
 <h5>Login failed</h5>
 <p>Either your email or password is wrong.</p>
 <p>Try again please!</p>
 <p><a href="{{ $view->url(['controller' => 'auth', 'action' => 'password-
restore']) }}">Forgot your password?</a></p>
 </div>
 {{ /if }}
 </fieldset>
 <fieldset>
 <dl>
 {{ $form->email->setLabel("E-mail")->removeDecorator('Errors') }}
 {{ $form->password->setLabel("Password")->removeDecorator('Errors') }}
 <dt>&nbsp;</dt>
 <dd>
 <span>
 <a class="register-link" href="{{ $view->url(['controller' => 'register',
'action' => 'index']) }}">Register</a>

<a class="register-link" href="{{ $view->url(['controller' => 'auth', 'action'
=> 'password-restore']) }}">Forgot your password?</a>

 </span>
 </dd>
 </dl>
 <div>
 <input type="submit" id="submit" value="Login" />
 </div>
 </fieldset>
</form>
```

REGISTRATION TEMPLATES

Registration process is divided in several steps. First one is just submitting email - then activation link is sent to the provided address, and upon user follows provided link, final action happens.

register_index.tpl from The New Custodian is doing this first step - providing the form for user to submit his email.

```
<h3>Register</h3>
<div>
{{ $form }}

<script type="text/javascript">
$('#email').change(function() {
 $.post('{{ $view->url(['controller' => 'register', 'action' => 'check-email'],
'default') }}?format=json', {
 'email': $(this).val()
 }, function (data) {
 if (data.status) {
 $('#email').css('color', 'green');
 } else {
 $('#email').css('color', 'red');
 }
 }, 'json');
}).keyup(function() {
 $(this).change();
});
</script>
</div>
```

Next step is in template register_after.tpl; this template simply informs user that the activation link is sent to his email address.

```
<h3>Confirmation has been sent.</h3>

<div>
 <p>Follow the steps you will find in your email.</p>
 <p>Thanks for registering.</p>
</div>
```

And then the page that opens is defined by the template register_confirm.tpl

```
<h3>Please fill in your data</h3>
<fieldset>
{{ $form }}


<script type="text/javascript">
$('#first_name, #last_name').keyup(function() {
 $.post('{{ $view->url(['controller' => 'register', 'action' => 'generate-
username'], 'default') }}?format=json', {
 'first_name': $('#first_name').val(),
 'last_name': $('#last_name').val()
 }, function (data) {
 $('#username').val(data.username).css('color', 'green');
 }, 'json');
});

$('#username').change(function() {
 $.post('{{ $view->url(['controller' => 'register', 'action' => 'check-
username'], 'default') }}?format=json', {
 'username': $(this).val()
 }, function (data) {
 if (data.status) {
 $('#username').css('color', 'green');
 } else {
 $('#username').css('color', 'red');
 }
 }, 'json');
}).keyup(function() {
 $(this).change();
});
</script>
</fieldset>
```

RETRIVING FORGOTTEN PASSWORD

Similarly, process of retreiving new password is controlled by Zend framework controllers, and pre-defined templates are executed in this order

auth_password-restore.tpl

```
{{ assign var="userindex" value=1 }}
<header>
 <h3>Reset your password</h3>
</header>
<form action="{{ $form->getAction() }}" method="{{ $form->getMethod() }}">
 <fieldset>
 {{ if $form->email->hasErrors() }}
 <div>
 <h5>E-mail is not correct</h5>
 <p>Maybe you registered on <em>{{ $gimme->publication->name }}</em> with
another e-mail account?</p>
 </div>
 {{ /if }}
 </fieldset>
 <fieldset>
 <dl> {{ $form->email->setLabel("E-Mail")->removeDecorator('Errors') }}</dl>
 <div>
 <input type="submit" id="submit" class="button" value="Request new password"
/>
 </div>
 </fieldset>
</form>
```

Next, template auth_password-restore-after.tpl generates the message for the user about e-mail message whch containes link to the page with change password action

```
<header>
 <h3>User account</h3>
</header>
<div>
 <h5 class="checkHeading">We've sent you an e-mail</h5>
 <p>Please check your inbox and click on the link in the email to reset your
password.</p>
</div>
```

Finally, auth_password-restore-finish.tpl does the required action -

```
<form action="{{ $form->getAction() }}" method="{{ $form->getMethod() }}">
 <fieldset>
 {{ if $form->isErrors() }}
 <div>
 <p>Your password could not be changed. Please follow the instructions and try
again.</p>
 </div>
 {{ /if }}
 </fieldset>
 <fieldset>
 <dl>
 {{ $form->password->setLabel("New Password")->removeDecorator('Errors') }}
 {{ if $form->password->hasErrors() }}
 <dt>&nbsp;</dt>
 <dd>
 <span>Please enter your new password (minimum 6 characters)</span>
 </dd>
 {{ /if }}
 </dl>
 <dl>
 {{ $form->password_confirm->setLabel("Retype your password")-
>removeDecorator('Errors') }}
 {{ if $form->password_confirm->hasErrors() && !$form->password->hasErrors() }}
 <dt>&nbsp;</dt>
 <dd>
 <span class="error-info">The confirmation of your password does not match your
password.</span>
 </dd>
 {{ /if }}
 </dl>

<div>
 <input type="submit" id="submit" class="button" value="Save password" />
 </div>
 </fieldset>
 </form>
```

USER DASHBOARD

User dashboard is the page where user is directed to check and replace it's personal info. The page is acccessible once user is logged in - it is where you get from the first page after clicking your name in welcome/login box.

Welcome, Ljuba Ranković | logout

dashboard_index.tpl is as follows:

```
<script>
function afterRegistration() {
 location.reload();
}
</script>


<h3>Welcome, {{ $user->name }}</h3>

<figure>
 <img src="{{ include file="_tpl/user-image.tpl" user=$user width=156
height=156 }}" style="max-width: 100%" rel="resizable" />
</figure>

<div>

{{ $form }}

</div>
```

48. USERS, AUTHORS, STAFF

Whereas many smaller blogging or web publishing tools do not differentiate between users and authors, Newscoop provides a professional framework necessary for consistent and future proof author and user management.

TOP 5 REASONS WHY YOU SHOULDN'T MIX AUTHORS AND USERS

- Your media production is growing. People start working production shifts. As users, your staff can log in, with the administration rights assigned to them, they can add images, add text, edit, proofread and assign any author in the system to an article. They can add new authors, photographers, anything and assign them to articles. No need for authors to add their own material, to register and log in. Your staff can work efficiently as an autonomous editorial team.
- Imagine a free lancer who delivered a lot of mediocre articles that didn't get published. As revenge he wrote something nasty in his user bio and changed the picture showing his middle finger. As a publisher, you are glad that this image and the text do not automatically show up alongside every article - because authors and users are separate entities.
- The author system in Superdesk allows to create job titles on the fly. Especially for longer, investigative articles, the contributors will be listed for their work (writer, photography, research, interview, ...) and not just come up as "written by" with a list of names.
- Paying invoices is easy now. Your staff prints out automatically generated lists of articles each day (week, month) and can structure the pay stubs even along the article IDs given out by Superdesk. No writer, photographer, researcher will be paid late, because once their article is published, the accountant has all the relevant information on the table.
- Like any professional publisher, you are in a position to commission, acquire, translate articles and add them to your publication. No need to create new accounts for new writers in the system. Your staff can do everything from A to Z.

We distinguish between:

- **Author** - the author of content in your publication, may that be text, images or something else
- **User** - a community member, somebody who has registered, can comment and perform other tasks
- **Staff** - somebody who can login to backend, edit your publication and contribute content

The basic assumptions:

- an author has contributed content to your publication (image, text, etc.)
- an author is not necessarily a staff member - and therefore might not have or require a login
- a user is somebody who registered him/herself on the site or has been registered through the admin interface - and therefore has a login
- some users have access to the admin interface - we consider them "staff"
- users with access to the admin interface (staff) differ in the rights they have (e.g. create, publish, delete, etc.)

Following these assumptions you can map most use scenarios, like commissioned or translated articles (author only) as well as community contributions (registered users writing comments) or contributing editors (users with admin access).

There is an additional feature in Newscoop 4 that allows to link authors with the community, following these assumptions:

- some authors are staff members - they have two profiles: author and user
- staff members are (by definition) community users / members
- in the admin interface I can link between users and authors
- if such links exist, the template can display a link from the author information to the matching user page

It is important to stress that all content on the author profile is editorial content. The portrait image follows the design of the publication. The biography information is written and edited by staff members, not the user him/herself. The user profile can be edited by the user him/herself.

WHO OWNS WHAT?

- Author picture and bio are editorial content and are owned by the publication. They are part of the corporate identity and the brand of the publication.
- User picture and bio / info are owned by the users and can be changed by them any time. There is no editorial control over the content. And user can delete everything, also cancel their account.

49. LISTING USERS

You can list users inside every page. Inside list_users list you have access to $user object under $gimme->list_user variable.

Simple example:

```
{{ list_users length=5 }}
  {{ $gimme->list_user->first_name }}
{{ /list_users }}
```

list_users have several really useful parameters, for example:

search:

```
{{ list_users length=5 search="username" }}
  {{ $gimme->list_user->first_name }}
{{ /list_users }}
```

filter by most active users:

```
{{ list_users length=5 filter="active" }}
  {{ $gimme->list_user->first_name }}
{{ /list_users }}
```

filter by editors (editor_groups are id for your installation editors groups):

```
{{ list_users length=5 filter="editors" editor_groups="1,2,3,4" }}
  {{ $gimme->list_user->first_name }}
{{ /list_users }}
```

filter by alphabet range:

```
{{ list_users length=5 filter="a-c" }}
  {{ $gimme->list_user->first_name }}
{{ /list_users }}
```

DEFAULT NEWSCOOP USERS LIST PAGE

Newscoop has user listing page by default. You can find it under mypage.com/user. You can edit this default page by creating user_index.tpl template in root directory of Your theme. Inside this file you can access $users object and loop through it like this.

```
{{ foreach $users as $user }}

    {{ $user->image(<width>, <height>) }} <!-- user profile image -->

    {{ $user->first_name }}

    {{ $user->last_name }}

    {{ $user->uname }} <!-- user name -->

    {{ $user['bio'] }} <!-- user biography text -->

    {{ $user->isAuthor() }} <!-- returns true if user is connected to author --
>

{{ /foreach }}
```

Check user object reference to find out more.

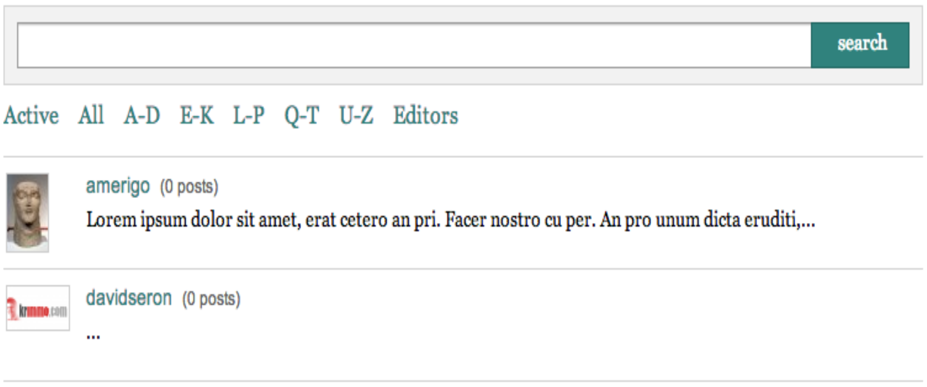
LINK TO USER PROFILE

```
{{ foreach $users as $user }}

    ...

     <a href="{{ $view->url(['username' => $user->uname], 'user') }}" title="">

         {{ $user->uname }}

    </a>

    ...

{{ /foreach }}
```

PAGINATION

```
{{ foreach $users as $user }}

   ...

{{ /foreach }}


{{ if isset($paginator->previous) }}

      <a href="{{ $view->url(['page' => $paginator->previous]) }}">previous</a>

{{ /if }}



<!-- page numbers -->

{{foreach $paginator->pagesInRange as $page}}

   {{if $paginator->current eq $page}}

        <a class="current" href="{{ $view->url(['page' => $page]) }}">{{ $page
}}</a>

   {{else}}

        <a href="{{ $view->url(['page' => $page]) }}">{{ $page }}</a>

   {{/if}}

 {{/foreach}}


{{ if isset($paginator->next) }}

    <a href="{{ $view->url(['page' => $paginator->next]) }}">next</a>

{{ /if }}
```

SEARCH AND LIST FILTERING


search
Active All A-D E-K L-P Q-T U-Z Editors
amerigo (0 posts)
Lorem ipsum dolor sit amet, erat cetero an pri. Facer nostro cu per. An pro unum dicta eruditi,...
davidseron (0 posts)
...

Creating a **search form** for user_index.tpl template is pretty easy and straight forward. Simply use code below.

```
<form method="GET" action="{{ $view->url(['controller' => 'user', 'action' =>
'search'], 'default', true) }}">

         <input type="text" name="q"></input>

         <input type="submit" value="Search"></input>

 </form>
```

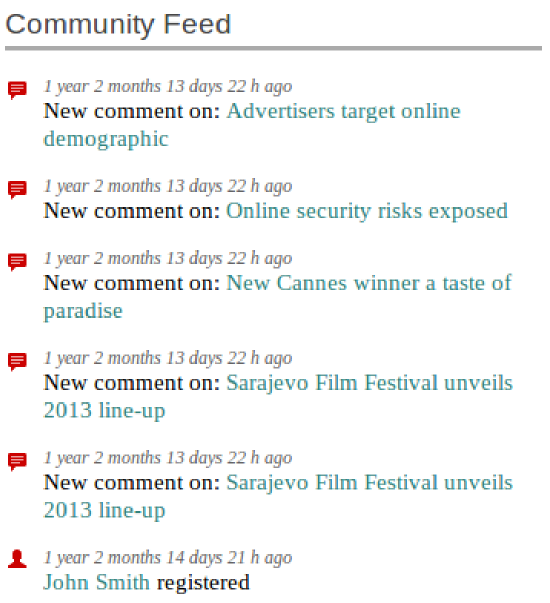
To filter user list by **surname** use this simple code.

```
<ul>
    <li><a href="{{ $view->url(['controller' => 'user', 'action' => 'index'],
'default', true) }}">Active Users</a></li>
    <li><a href="{{ $view->url(['controller' => 'user', 'action' => 'filter',
'f' => 'a-z'], 'default', true) }}">All Users</a></li>
    <li><a href="{{ $view->url(['controller' => 'user', 'action' => 'filter',
'f' => 'a-d'], 'default', true) }}">A-D</a></li>
    <li><a href="{{ $view->url(['controller' => 'user', 'action' => 'filter',
'f' => 'e-k'], 'default', true) }}">E-K</a></li>
    <li><a href="{{ $view->url(['controller' => 'user', 'action' => 'filter',
'f' => 'l-p'], 'default', true) }}">L-P</a></li>
    <li><a href="{{ $view->url(['controller' => 'user', 'action' => 'filter',
'f' => 'q-t'], 'default', true) }}">Q-T</a></li>
    <li><a href="{{ $view->url(['controller' => 'user', 'action' => 'filter',
'f' => 'u-z'], 'default', true) }}">U-Z</a></li>
    <li><a href="{{ $view->url(['controller' => 'user', 'action' => 'editors'],
'default', true) }}">Editors</a></li>
</ul>
```

COMMUNITY FEED

Community functionality in Newscoop was initially developed for some clients, but has since then become the part of our offical themes.

Here we present Community Feed widget that lists two specific activities of the community memebrs - newly registered users notification and recommended comment by registered users.



Community Feed

1 year 2 months 13 days 22 h ago
New comment on: Advertisers target online demographic

1 year 2 months 13 days 22 h ago
New comment on: Online security risks exposed

1 year 2 months 13 days 22 h ago
New comment on: New Cannes winner a taste of paradise

1 year 2 months 13 days 22 h ago
New comment on: Sarajevo Film Festival unveils 2013 line-up

1 year 2 months 13 days 22 h ago
New comment on: Sarajevo Film Festival unveils 2013 line-up

1 year 2 months 14 days 21 h ago
John Smith registered

Template that generates this output in New Custodian is _tpl/sidebar-community-feed.tpl.

```
{{ list_community_feeds length="10" }}

{{ if $gimme->current_list->at_beginning }}
  <div class="community_ticker" class="clearfix">
  <h3>Community feed</h3>
  <ul>
{{ /if }}

{{ assign var="created" value=$gimme->community_feed->created }}
{{ assign var="user" value=$gimme->community_feed->user }}

 {{ if $gimme->community_feed->type == 'user-register' && $user->uname }}

 <li class="registered">{{ include file="_tpl/relative_date.tpl" date=$created
}}

    <a {{ if $user->is_active }} href="{{ $view->url(['username' => $user-
>uname], 'user') }}"{{ /if }}>{{ $user->first_name }} {{ $user->last_name
}}</a> registered

 </li>

  {{ elseif $gimme->community_feed->type == 'comment-recommended' && $gimme->community_feed->comment->article }}

   <li class="commented">

      {{ include file="_tpl/relative_date.tpl" date=$created }} New comment on <a href="{{ $gimme->community_feed->comment->article->url }}">{{ $gimme->community_feed->comment->article->title }}</a>

    </li>
  {{ /if }}

  {{ if $gimme->current_list->at_end }}
     </ul>
     </div>
 {{ /if }}

{{ /list_community_feeds }}
```

Template for showing relative time since the action was performed is needed to nicely format time since the action was performed, and that template is here

```
{{ $diff=date_diff(date_create('now'), date_create($date)) }}
 <small class="date relative">
 {{ if $diff->y }} {{ $diff->y }} {{ if $diff->y > 1 }}years{{ else }}year{{ /if }}{{ /if }}
 {{ if $diff->m }} {{ $diff->m }} {{ if $diff->m > 1 }}months{{ else }}month{{ /if }}{{ /if }}
 {{ if $diff->d }} {{ $diff->d }} {{ if $diff->d > 1 }}days{{ else }}day{{ /if }}{{ /if }}
 {{ if $diff->h && (!$diff->d || empty($short)) }} {{ $diff->h }} h{{ /if }}
 {{ if !$diff->d && $diff->i && (empty($short) || !$diff->h) }} {{ $diff->i }} min{{ /if }}
 {{ if !$diff->d && !$diff->h && !$diff->i && $diff->s }} {{ $diff->s }} sec{ /if }} ago
 </small>
```

50. MAKING AUTHOR PROFILE PAGE

LISTING FROM ARTICLE AUTHOR LIST TO USER PROFILE

Inside an article, a typical link to author profile could be like this:

```
{{ list_article_authors }}

  {{ if $gimme->author->user->defined }}

        <a href="{{ $view->url(['username' => $gimme->author->user->uname],
'user') }}">

  {{ /if }}



  {{ $gimme->author->name }}



  {{ if $gimme->author->user->defined }}

        </a>

  {{ /if }}

{{ /list_article_authors }}
```

The important new feature here is the check if user is "defined".

```
{{ if $gimme->author->user->defined }}
```

If the user is "defined" that means that the author has a link to a registered user. If this is the case, a link is presented to the profile:

```
<a href="{{ $view->url(['username' => $gimme->author->user->uname], 'user')
}}">
```

A link will be created following the structure /user/profile/username - where "username" is differing depending on the author listed.

THE USER PROFILE PAGE

IMPORTANT

We will always use the template **user_profile.tpl** in the root directory of the theme. If the template is not present a system template will be displayed.

DISPLAY USER PROFILE DATA

If user is an author we want to use author data as much as possible because author biography is translateable.

```
<img src="{{ $user->image(<width>, <height>) }}" />

<h4>

 {{ if $user->author->defined }}

      {{ $user->author->biography->first_name }} {{ $user->author->biography-
>last_name }}

 {{ else }}

      {{ $user->first_name }} {{ $user->last_name }}

 {{ /if }}

</h4>
```

```
{{ if $user->author->defined }}

   <p>

      {{ $user->author->biography->text }}

   </p>

{{ else }}

   <p>

      {{ $user['bio']|escape }}

   </p>

{{ /if }}
```

To learn more please check user, author and author biography object references.

IF USER IS AUTHOR, LIST ARTICLES

The user can also be an author, in which case the profile page can display a list of all articles the author is related to. This is the case if in the admin interface a link between an author and a user profile has been established in the profile editing area. If this is the case, **$user->isAuthor()** is true.

Firstly, check if the user is an author and if so, create the variable **$escapedName** containing the name with an escaped whitespace. This means that "First and Lastname" will be converted to "First\ and\ Lastname" - which we need in the list command below.

```
{{ if $user->isAuthor() }}

  {{ $escapedName=str_replace(" ", "\ ", $user->author->name) }}

{{ /if }}
```

Now we have a variable that only exists IF the user whose page we visit is also an author. We will use this variable in the article list we create now. You can see that the variable is used in the constraints of the list. This means that no list is created, if the user is not an author. We ignore publication, issue and section to list all articles related with the user / author.

```
{{ list_articles ignore_publication="true" ignore_issue="true"
ignore_section="true" constraints="author is $escapedName" order="bypublishdate
desc" }}

     ...

     <a href="{{ $gimme->article->url }}" >{{ $gimme->article->title }}</a>

     ...

{{ /list_articles }}
```

ADVANCED TEMPLATING

51. 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:

```
<ul>
{{ list_articles ignore_issue="true" ignore_section="true" constraints="type is
page" }}
  <li><a href="{{ url options="article" }}">{{ $gimme->article->name
}}</a></li>
{{ /list_articles }}
</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.

```
{{ local }}
{{ set_issue number="1" }}
{{ list_sections }}
```

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

```
{{ list_articles constraints="type is page" }}
    {{ if $gimme->current_list->at_beginning }}
        <h3>{{ $gimme->section->name }}</h3>
        <ul>
    {{ /if }}
```

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

```
        <li><a href="{{ url 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.

```
        {{ if $gimme->current_list->at_end }}
        </ul>
        {{ /if }}
```

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

```
{{ /list_articles }}
{{ /list_sections }}
{{ /local }}
```

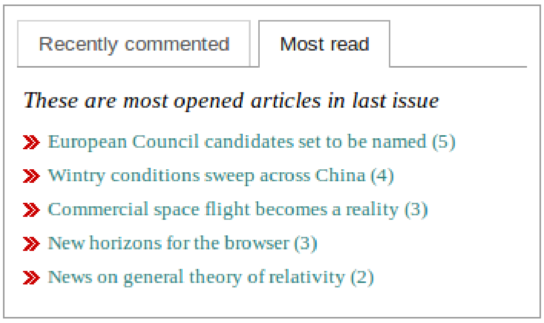
This way, editors of the publication have the option to independently update the content of these static pages. Keeping them in issue number 1, they can easily jump there with only one click from the issue listing page.

52. 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.

For those articles that you want statistics gathered you need to firstly put {{ count }} tag in template which outputs whole article (in the case of Newscoop default themes and 'news' articles, it is _tpl/article-cont.tpl). Also, don't forget to turn on 'is Content' switch for the full_text field (or whatever you call that one that holds main article content) in article types management.

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.



These are most opened articles in last issue

- European Council candidates set to be named (5)
- Wintry conditions sweep across China (4)
- Commercial space flight becomes a reality (3)
- New horizons for the browser (3)
- News on general theory of relativity (2)

This code snippet is taken from The New Custodian's _tpl/sidebar-most.tpl template.

```
{{ set_current_issue }}

{{ list_articles length="5" ignore_section="true" order="bypopularity desc"
constraints="type is news" }}

 <li><a href="{{ url options="article" }}">{{ $gimme->article->name }} ({{
$gimme->article->reads }})</a></li>

{{ /list_articles }}
```

After it sets the context to the current issue, it will list five most read articles form that issue, regarding of the section. Articles will be ordered by the number of reads, but (as constraints define) only articles of type 'news'. Articles are listed with their names linked to whole article page, and with number of reads shown for every one.

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

```
{{ list_articles length="10" constraints="type is article section greater 19
section smaller 301 issue greater `$gimme->issue->number-5`"
order="bypopularity 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

```
{{assign var="xdate" value="-7 days"|date_format:"%Y-%m-%d"}}
{{list_articles length="10" order="bypopularity desc" constraints="publish_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

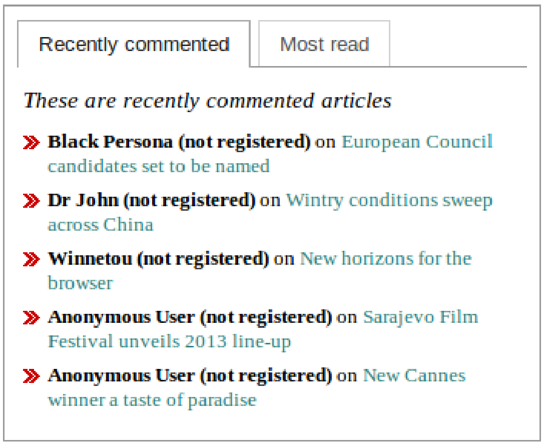
- Set constraints where the article type is 'article,' and the publish date is within the time we set in the variable
- Return articles regardless of issue
- Return articles regardless of section

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

53. 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 New Custodian's front page, from the sub-template **_tpl/sidebar-most.tpl**

Here is a screenshot:



These are recently commented articles

- **Black Persona (not registered)** on European Council candidates set to be named
- **Dr John (not registered)** on Wintry conditions sweep across China
- **Winnetou (not registered)** on New horizons for the browser
- **Anonymous User (not registered)** on Sarajevo Film Festival unveils 2013 line-up
- **Anonymous User (not registered)** on New Cannes winner a taste of paradise

This code snippet will:

- List five articles, ordered by the latest comment, in descending order - regardless of the issue or section
- Set a constraint so that the article type is 'news'
- Point to the latest comment for that article
- List the nickname given by a commenter on a comment (with 'not registered' mark if user is not logged in, or with the link created of username to his profile page, is commenter was logged in)
- Combine that with the name of the article they're commenting on, with link to full article page

```
{{ list_articles length="5" ignore_issue="true" ignore_section="true"
order="byLastComment desc" constraints="type is news" }}

 <li>{{ list_article_comments length="1" order="bydate desc"}}<b>{{ if $gimme-
>comment->user->identifier }}
 <a href="http://{{ $gimme->publication->site }}/user/profile/{{ $gimme-
>comment->user->uname|urlencode }}">{{ $gimme->comment->user->uname }}</a>
 {{ else }}
 {{ $gimme->comment->nickname }} {{ #anonymous# }}
 {{ /if }}</b>{{ /list_article_comments }} on <a href="{{ uri options="article"
}}">{{ $gimme->article->name }}</a></li>

{{ /list_articles }}
```

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

```
{{list_article_comments length="10" ignore_article="true" order="byDate desc"}}
    <a href="{{url}}#comments">{{$gimme->article->name}}</a><sup>{{$gimme-
>article->comment_count}}</sup>
    <p>{{$gimme->comment->content|truncate:400}} <span>{{$gimme->comment-
>submit_date|camp_date_format:"%H:%i"}}</span></p>
{{/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

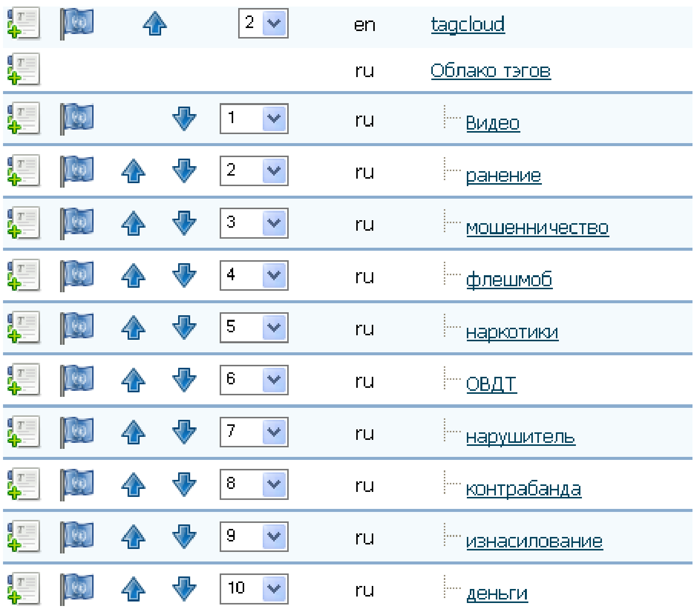
- Return the article's URI
- Return the article's name
- Return the count of comments for the article

```
{{ list_articles length="10" order="byLastComment desc" ignore_issue="true"
ignore_section="true" }}
  <a href="{{ url }}#comments">{{ $gimme->article->name }}</a><sup>{{ $gimme-
>article->comment_count }}</sup>
{{ /list_articles }}
```

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

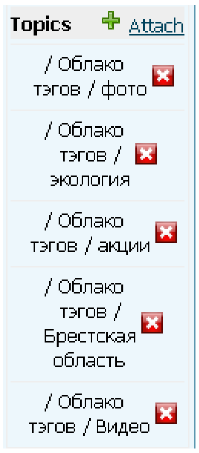
54. TAG CLOUDS USING TOPICS

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':



2	en	tagcloud
	ru	Облако тэгов
1	ru	Видео
2	ru	ранение
3	ru	мошенничество
4	ru	флешмоб
5	ru	наркотики
6	ru	ОВДТ
7	ru	нарушитель
8	ru	контрабанда
9	ru	изнасилование
10	ru	деньги

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



Topics Attach

/ Облако тэгов / фото

/ Облако тэгов / экология

/ Облако тэгов / акции

/ Облако тэгов / Брестская область

/ Облако тэгов / Видео

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:

```
{{ local }}
{{ unset_issue }}
{{ unset_section }}
{{ unset_article }}
{{ unset_topic }}
{{ set_topic name="tagcloud:en" }}
{{ assign var="first_good_tag" value=true }}
{{ list_subtopics }}
    {{ assign var="posts_count" value="0" }}
    {{ list_articles ignore_issue="true" }}
        {{ assign var="posts_count" value=$gimme->current_list->count }}
    {{ /list_articles }}
    {{ if $posts_count > 0 }}
        {{ if $first_good_tag }}
            {{ assign var="min" value=$posts_count }}
            {{ assign var="max" value=$posts_count }}
            {{ assign var="first_good_tag" value=false }}
        {{ /if }}
        {{ if $posts_count > $max }}
            {{ assign var="max" value=$posts_count }}
        {{ /if }}
        {{ if $posts_count < $min }}
            {{ assign var="min" value=$posts_count }}
        {{ /if }}
    {{ /if }}
{{ /list_subtopics }}

{{ assign var="minSize" value="90" }}
{{ assign var="maxSize" value="240" }}

{{ assign var="diff_max_min" value="`$max-$min`" }}
{{ 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:

```
<ul class="tag-cloud">
{{ list_subtopics }}
    {{ assign var="posts_count" value="0" }}
    {{ list_articles ignore_issue="true" }}
        {{ assign var="posts_count" value=$gimme->current_list->count }}
        {{ assign var="tag_name" value=$gimme->topic->name }}
    {{ /list_articles }}
```

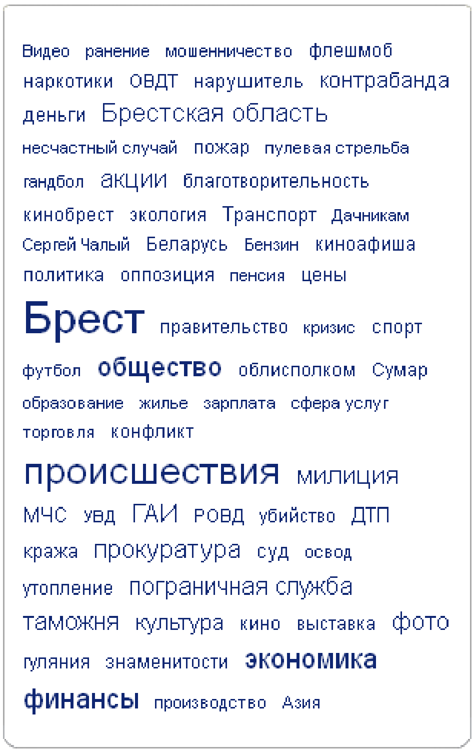
Then we calculate font size for the sub-topic in the tag cloud:

```
{{ if $posts_count > 0 }}
    {{ if $min == $max }}
        {{ assign var="fontSize" value="`$diff_maxSize_minSize/2+$minSize`" }}
    {{ else }}
    {{ assign var="a" value="`$posts_count-$min`" }}
    {{ assign var="b" value="`$a/$diff_max_min`" }}
    {{ assign var="fontSize" value="`$b*$diff_maxSize_minSize+$minSize`" }}
{{ /if }}
```

And finally output the result:

```
    <li>
        <span style="}}%"><a href="/?
tpl=special_template_to_list_articles_based_on_topic.tpl&tag={{ $tag_name }}"
title="{{ $posts_count }} articles having topic {{ $tag_name }}">{{ $tag_name
}}</a></span>
    </li>
{{ /if }}
{{ /list_subtopics }}
</ul>
{{ /local }}
```

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


Видео ранение мошенничество флешмоб наркотики ОВДТ нарушитель контрабанда деньги Брестская область несчастный случай пожар пулевая стрельба гандбол акции благотворительность кинобрест экология Транспорт Дачникам Сергей Чалый Беларусь Бензин киноафиша политика оппозиция пенсия цены Брест правительство кризис спорт футбол общество облисполком Сумар образование жилье зарплата сфера услуг торговля конфликт происшествия милиция МЧС увд ГАИ РОВД убийство ДТП кража прокуратура суд освод утопление пограничная служба таможня культура кино выставка фото гуляния знаменитости экономика финансы производство Азия

Tag cloud created using topics

55. PAGINATION OF ARTICLE LISTS

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.

This chapter will introduce an easy way for pagination in the beginning. If you run a larger site, you should use the method explained further down in this chapter.

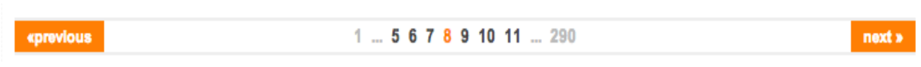
PAGINATION OF A LONG ARTICLE LIST

Usually 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.

The simplest pagination with "next" and "previous" buttons will look like this:

```
{{list_articles length="10"}}
    ...


    <!-- we want to have pagination at the end of article list -->
    {{if $gimme->current_list->at_end}}

      <!-- let's make sure that there are more articles-->

      {{ if $gimme->current_list->count > $gimme->current_list->length }}

          {{ if $gimme->current_list->has_previous_elements }}

              <a href="{{ url options="previous_items" }}"
title="">previous</a>

          {{/if}}


          {{ if $gimme->current_list->has_next_elements }}

             <a href="{{ url options="next_items" }}" title="">next</a>

          {{/if}}

      {{/if}}

   {{/if}}

{{/list_articles}}
```

PAGINATION WITH PAGE NUMBERS



To get this kind of pagination we need to dig into smarty code a little bit more. Please read the code. It is clear enough to get general idea how it works.

```
{{list_articles length="10"}}
...
<!-- we want to have pagination at the end of article list -->
{{if $gimme->current_list->at_end}}
{{ if $gimme->current_list->count > $gimme->current_list->length }}
{{ $page=intval($gimme->url->get_parameter($gimme->current_list_id())) }}
{{ $list_id=$gimme->current_list_id() }}
<!-- sets the length of article list -->
{{$listLength=10}}
{{assign var="allPages" value=($gimme->current_list->count/$listLength)|ceil }}
{{assign var="currentPage" value=$page/$listLength}}
{{assign var="firstToShow" value=$currentPage-2}}
{{assign var="lastToShow" value=$currentPage+5}}
{{if $firstToShow < 1 }}
{{assign var="firstToShow" value=1}}
{{assign var="lastToShow" value=$lastToShow+3}}
{{/if}}
{{if $lastToShow > $allPages }}
```

```
{{assign var="lastToShow" value=$allPages+1}}
{{/if}}
<nav class="bottom_nav pagination">
<!-- we are unsetting article to make sure that is not going to affect url -->
{{ unset_article }}
{{ if $gimme->current_list->has_previous_elements }}
<a href="{{ url options="previous_items" }}" class="float_left nav_button prev"
title="">previous
</a>
{{/if}}
{{if $lastToShow-$firstToShow>0}}
<div class="numbers">
 <ul>
{{if $firstToShow>1}}
<li class="firstlast"><a href="{{ url options="section" }}{{if $gimme->topic-
>identifier}}?tpid={{$gimme->topic->identifier}}{{/if}}">1</a></li>
 <li class="firstlast">...</li>
{{/if}}
{{section name=foo start=$firstToShow loop=$lastToShow}}
{{if $smarty.section.foo.index-1==$currentPage}}
<li class="current">{{ $smarty.section.foo.index }}</li>
{{else}}
<li><a href="{{ url options="section" }}?{{$list_id}}={{
($smarty.section.foo.index-1)*$listLength }}{{if $gimme->topic-
>identifier}}&tpid={{$gimme->topic->identifier}}{{/if}}">
 {{ $smarty.section.foo.index }} </a></li>
{{/if}}
 {{/section}}
{{if $lastToShow-1<$allPages}}
<li class="firstlast">...</li>
 <li class="firstlast"><a href="{{ url options="section" }}?{{$list_id}}={{
($allPages-1)*$listLength }}{{if $gimme->topic->identifier}}&tpid={{$gimme-
>topic->identifier}}{{/if}}">{{$allPages}}</a></li>
{{/if}}
</ul>
 </div>
{{/if}}
 {{ if $gimme->current_list->has_next_elements }}
<a href="{{ url options="next_items" }}" class="float_right nav_button next"
title="">{{ if $gimme->language->english_name == "Georgian" }}შემდეგი{{elseif
$gimme->language->english_name == "Russian"}}следующий{{else}}next{{/if}}
&raquo;</a>
{{ /if }}
</nav>
{{ /if }}
{{/if}}
{{/list_articles}}
```

REALLY ADVANCED PAGINATION

Well it's not that advanced really but...

Consider a case where you may have differing numbers of articles in different parts of your section article lists. In the example below we have a section page that comes from our Rockstar theme where there are two lead articles before the standard listing begins.

```
{{ $listLength = 9 }}

{{ list_articles length=$listLength }}

  {{ if $gimme->current_list->at_beginning }}

    {{ if $gimme->current_list->index lte 2 }}

      <section class="grid-2">

    {{ else }}

      <section class="grid-3">

    {{ /if }}

  {{ /if }}
  {{ if $gimme->current_list->index lte 2 }}

      <article>

        {{ include file="_tpl/img/img_onehalf.tpl" }}

        <small><a href="{{ url options='section' }}">{{ $gimme->section->name
}}</a></small>

        <h3><a href="{{ url options='article' }}">{{ $gimme->article->name
}}</a></h3>

<p><span class="time">{{ $gimme->article->publish_date }}</span> /

          {{ list_article_authors }}

{{ if $gimme->author->user->defined }}<a href="{{ $view->url(['username' =>
$gimme->author->user->uname], 'user') }}">{{ /if }}by {{ $gimme->author->name
}}{{ if $gimme->author->user->defined }}</a>{{ /if }}

          {{ /list_article_authors }}

        </p>

        <p>{{ include file="_tpl/_edit-article.tpl" }}{{ $gimme->article-
>deck|truncate:360 }}</p>

        <span class="more"><a href="{{ url options='article' }}">+  {{
#readMore# }}</a> or <a href="{{ url options='article' }}#comments">{{
#addComment# }} ({{ $gimme->article->comment_count }})</a></span>

      </article>

  {{ /if }}

  {{ if $gimme->current_list->index == 2 || ($gimme->current_list->at_end &&
$gimme->current_list->index lte 2) }}

    </section><!-- / 2 article grid -->

  {{ /if }}

  {{ if $gimme->current_list->index == 3 }}

… do the rest of your code after this point
```

As you can see we're setting a list limit of 9 but the first page has 8 items due to the two proceeding lead articles.

To combat this we need to set the index back by one so that there's no gap in articles when we go from 8 to 9 elements.

This is as simple as altering this line in your pagination:

```
<li><a href="{{ url options="section" }}?{{$list_id}}={{
($smarty.section.foo.index-1)*($listLength)-1 }}{{if $gimme->topic-
>identifier}}&tpid={{$gimme->topic->identifier}}{{/if}}">{{
$smarty.section.foo.index }}</a>
```

Of course your mileage may vary if you have rolled your own solution but the lesson to take away from this is that you can specify where the pagination index starts from but not reset the index programatically.

Good luck and happy paginating!

56. 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:

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

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

```
<title>{{ $gimme->publication->name }}</title>
<link>http://{{ $gimme->publication->site }}</link>
<description>{{ $siteinfo.description }}</description>
<language>{{ $gimme->language->code }}</language>
<copyright>Copyright {{ $smarty.now|date_format:"%Y" }}, {{ $gimme-
>publication->name }}</copyright>
<lastBuildDate>{{ $smarty.now|date_format:"%a, %d %b %Y %H:%M:%S" }}
+0100</lastBuildDate>
<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.

```
<image>
    <url>{{ url static_file="_img/logo-rss.jpg" }}</url>
    <title>{{ $gimme->publication->name }}</title>
    <link>http://{{ $gimme->publication->site }}</link>
    <width>144</width>
    <height>19</height>
</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.

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

Now we need to define the list of items.

```
<item>
 <title>{{ $gimme->article->name|escape }}</title>
  <link>{{ url options="article" }}</link>
   <description>
   {{ $gimme->article->intro|escape }}
   </description>
    <category domain="{{ url options="section" }}">
 {{ $gimme->section->name }}
    </category>
   {{ if $gimme->article->author->name }}
```

```
    <atom:author>
    <atom:name>{{ $gimme->article->author->name }}</atom:name>
    </atom:author>
    {{ /if }}
   <pubDate>
   {{ $gimme->article->publish_date|date_format:"%a, %d %b %Y %H:%M:%S" }} +0100
   </pubDate>
  <guid isPermaLink="true">{{ url options="article" }}</guid>
</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:

```
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom"
xmlns:media="http://search.yahoo.com/mrss/">
    <channel>
        <title>{{ $gimme->publication->name }}</title>
        <link>http://{{ $gimme->publication->site }}</link>
        <description>{{ $siteinfo.description }}</description>
        <language>{{ $gimme->language->code }}</language>
        <copyright>Copyright {{ $smarty.now|date_format:"%Y" }}, {{ $gimme-
>publication->name }}</copyright>
        <lastBuildDate>{{ $smarty.now|date_format:"%a, %d %b %Y %H:%M:%S" }}
+0100</lastBuildDate>
        <generator>Newscoop</generator>
        <image>
            <url>{{ url static_file="_img/logo-rss.jpg" }}</url>
            <title>{{ $gimme->publication->name }}</title>
            <link>http://{{ $gimme->publication->site }}</link>
            <width>144</width>
            <height>19</height>
        </image>
        {{ list_articles length="20" order="bypublishdate desc" }}
        <item>
            <title>{{ $gimme->article->name|escape }}</title>
            <link>{{ url options="article" }}</link>
            <description>
                {{ $gimme->article->intro|escape }}
            </description>
            <category domain="http://{{ $gimme->publication->site }}/{{ $gimme-
>language->code }}/{{ $gimme->issue->number }}/{{ $gimme->section->url_name
}}">{{ $gimme->section->name }}</category>
            {{ if $gimme->article->author->name }}
            <atom:author><atom:name>{{ $gimme->article->author->name
}}</atom:name></atom:author>
            {{/if}}
            <pubDate>{{ $gimme->article->publish_date|date_format:"%a, %d %b %Y
%H:%M:%S" }} +0100</pubDate>
            <guid isPermaLink="true">http://{{ $gimme->publication->site }}/{{
$gimme->language->code }}/{{ $gimme->issue->number }}/{{ $gimme->section-
>url_name }}/{{ $gimme->article->number }}</guid>
        </item>
    {{/list_articles}}
    </channel>
</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. To get working url to rss feed, we usualy do this: create RSS section somewhere (in 'static' issue if you have it for storing static pages, so it doesn't mix with real content), and then assign our rss.tpl template to that section. Now you will have url like this:

http://your.site.url/en/static/rss

because it is standard way that Newscoop creates url for section, just this one is not using section.tpl when it is accessed, but rss.tpl.

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.

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

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

All sections from the current issue in the English language.

```
{{ set_current_issue }}
{{ set_language name="English" }}
{{ list_sections }}
<url>
    <loc>{{ url options="section" }}</loc>
    <changefreq>weekly</changefreq>
    <priority>0.3</priority>
</url>
{{ /list_sections }}
```

All sections from the current issue in the French language.

```
{{ set_language name="French" }}
{{ list_sections }}
<url>
    <loc>{{ url options="section" }}</loc>
    <changefreq>weekly</changefreq>
    <priority>0.3</priority>
</url>
{{ /list_sections }}
```

This is how the full Sitemap file should look:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
{{ list_articles ignore_language="true" ignore_issue="true"
ignore_section="true" order="bypublishdate desc" constraints="type is news" }}
    <url>
        <loc>{{ url options="article" }}</loc>
        <lastmod>{{ $gimme->article->publish_date|camp_date_format:"%Y-%m-%d"
}}</lastmod>
        <changefreq>daily</changefreq>
        <priority>0.6</priority>
    </url>
{{ /list_articles }}
{{ list_articles ignore_issue="true" ignore_section="true"
ignore_language="true" order="bypublishdate desc" constraints="type is show" }}
    <url>
        <loc>{{ url options="article" }}</loc>
        <lastmod>{{ $gimme->article->publish_date|camp_date_format:"%Y-%m-%d"
}}</lastmod>
        <changefreq>daily</changefreq>
        <priority>0.8</priority>
    </url>
{{ /list_articles }}

{{ set_current_issue }}
{{ set_language name="English" }}
{{ list_sections }}
    <url>
        <loc>{{ url options="section" }}</loc>
        <changefreq>weekly</changefreq>
        <priority>0.3</priority>
    </url>
{{ /list_sections }}
{{ set_language name="French" }}
{{ list_sections }}
    <url>
        <loc>{{ url options="section" }}</loc>
        <changefreq>weekly</changefreq>
        <priority>0.3</priority>
    </url>
{{ /list_sections }}
</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.

```
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
  <name>{{ $gimme->article->name }} in {{ $gimme->section->name }}</name>
  <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.

```
{{ list_article_locations }}
    {{ if $gimme->location->enabled }}
    <Style id="style{{ $gimme->article->number }}-{{ $gimme->current_list-
>index }}">
        <IconStyle>
            <Icon>
                <href>http://www.sourcefabric.org/geolocation/markers/marker-
gold.png</href>
            </Icon>
        </IconStyle>
    </Style>
    {{ /if }}
{{ /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.

```
{{ 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 }}
```

This is how the full KML file should look:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
    <name>{{ $gimme->section->name }} in {{ $gimme->article->name }}</name>
    <description><![CDATA[{{ $gimme->article->description }}]]></description>
    {{ list_article_locations }}
        {{ if $gimme->location->enabled }}
        <Style id="style{{ $gimme->article->number }}-{{ $gimme->current_list-
>index }}">
            <IconStyle>
                <Icon>

<href>http://www.sourcefabric.org/geolocation/markers/marker-gold.png</href>
                </Icon>
            </IconStyle>
        </Style>
        {{ /if }}
    {{ /list_article_locations }}

    {{ 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 }}
</Document>
</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.

57. USING POLLS WITH AJAX

For poll functionality we are going to use 'debate' plugin, as it is improved version of 'poll' plugin, offering all the same options but more stable and improved.

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.

Small trouble with this approach is that, if website is going to have both debate functionality and independent poll somewhere on the frontpage, we cannot simply pull latest debate from the plugin and show it on the page because it can also be debate question meant for debate article. For that reason, we introduce new article type called 'poll' and attach poll from debate plugin to article of type 'poll'. Then instead of calling latest from the plugin, we ask for latest poll article, and then show it's poll.

```
{{ list_articles length="1" ignore_issue="true" ignore_section="true"
constraints="type is poll" }}
```

Then inside this list, we need to point to the debate attached to that article (template references to itslef because of reloading after vote action, so if in your case it is not called _tpl/sidebar-poll.tpl, cange the name in template too. This example is from The New Custdian):

```
{{ list_debates length="1" item="article" }}
{{ if $gimme->current_list->at_beginning }}
<div id="polldiv" class="clearfix">
 <h3>Poll</h3>
{{ /if }}
```

And then we work with debate - check if it is votable by user, is it expired etc. The code is more or less self-explanatory.

```
{{ if $gimme->debate_action->defined }}
 <blockquote>{{ $gimme->debate->question }}</blockquote>
 {{ if $gimme->debate->user_vote_count >= $gimme->debate->votes_per_user ||
$gimme->debate_action->ok }}
 <p class="poll-info">Thank you for casting your vote!</p>
 {{ elseif $gimme->debate_action->is_error }}
 <p>You have already voted</p>
 {{ /if }}
 <ul class="question-list">
 {{ assign var="votes" value=0 }}
 {{ list_debate_answers }}
 <li>
 <label for="radio{{ $gimme->current_list->index }}">{{ $gimme->debateanswer-
>answer }}</label>
 <span class="q-score" style="width:{{ math equation="round(x)" x=$gimme-
>debateanswer->percentage_overall format="%d" }}%;"> <small>{{ math
equation="round(x)" x=$gimme->debateanswer->percentage_overall format="%d"
}}%</small></span>
 </li>
 {{ assign var="votes" value=$votes+$gimme->debateanswer->votes }}
 {{ if $gimme->current_list->at_end }}
 <li class="total-votes"><span>Total number of votes: {{ $votes }}</span></li>
 {{ /if }}
 {{ /list_debate_answers }}

</ul>

{{ else }}

 {{ if $gimme->debate->is_votable }}

 <blockquote>{{ $gimme->debate->question }}</blockquote>
 {{ debate_form template="_tpl/sidebar-poll.tpl" submit_button="Vote"
html_code="id=\"poll-button\" class=\"button debbut center\"" }}

{{* this is to find out template id for this template, will have to be assigned
as hidden form field *}}
{{ $uriAry=explode("tpl=", {{ uri options="template _tpl/sidebar-poll.tpl" }},
2) }}

<input name="tpl" value="{{ $uriAry[1] }}" type="hidden">
 <ul class="question-list">
 {{ list_debate_answers }}
 <li>
 <!--input type="radio" id="radio{{ $gimme->current_list->index }}"
name="radios1" /-->
 {{ debateanswer_edit html_code="id=\"radio{{ $gimme->current_list->index }}\""
}}<label for="radio{{ $gimme->current_list->index }}">{{ $gimme->debateanswer-
>answer }}</label>
 <span class="q-score" style="width:{{ math equation="round(x)" x=$gimme-
>debateanswer->percentage_overall format="%d" }}%;"> <small>{{ math
equation="round(x)" x=$gimme->debateanswer->percentage_overall format="%d"
}}%</small></span>
 </li>
 {{ /list_debate_answers }}
 </ul>
 {{ /debate_form }}

 {{ else }}
 <blockquote>{{ $gimme->debate->question }}</blockquote>
 {{ if $gimme->debate->user_vote_count >= $gimme->debate->votes_per_user }}<p
class="poll-info">Thank you fr casting your vote!</p>{{ /if }}
 <ul class="question-list">
 {{ list_debate_answers }}
 <li>
 <label for="radio{{ $gimme->current_list->index }}">{{ $gimme->debateanswer-
>answer }}</label>
 <span class="q-score" style="width:{{ math equation="round(x)" x=$gimme-
>debateanswer->percentage_overall format="%d" }}%;"> <small>{{ math
equation="round(x)" x=$gimme->debateanswer->percentage_overall format="%d"
}}%</small></span>
 </li>
 {{ /list_debate_answers }}
 </ul>
 {{ /if }}

{{ /if }}

{{ if $gimme->current_list->at_end }}
</div>
{{ /if }}
{{ /list_debates }}
```

For successful reloading we need to provide some code like this which will define what is to be reloaded (div container with the id polldiv)

```
/* Poll Ajaxified
 ------------------------------------------------------*/
 $('#poll-button').click(function(){

$.post($('form[name=debate]').attr("action"),$('form[name=debate]').serialize(),
 return false;
 });
```

In The New Custodian case this code is in _js/script.js.

58. DIRECT FRONTEND LINK TO ARTICLE EDITING

Sometimes journalists and editors realize that they want to change somethnig in an article only after they see it on fronted. And they want to do it quickly, even if they don't know in which issue and which section of the publication structure article is placed. And they don't want to spend time searching for it.

Now there's this handy solution: just put this code on right places - on the beginning of article teasers is good option - and it will does the magic, it will create the link from every article teaser to article edit screen in backend. And it will do it only to users who have administrative privileges, of course!

```
{{ if $gimme->user->is_admin }}
<a class="hard-highlight"
href="http://{{ $gimme->publication->site }}/admin/articles/edit.php?
f_publication_id={{ $gimme->publication->identifier }}&f_issue_number={{
$gimme->issue->number }}&f_section_number={{ $gimme->section->number
}}&f_article_number={{ $gimme->article->number }}&f_language_id={{ $gimme-
>language->number }}&f_language_selected={{ $gimme->language->number }}"
target="_blank"
style="" title="Edit article">
&para;
</a>
{{ /if }}
```

59. SUBSECTIONS

Although subsections are not officially supported by the default Newscoop installation there are a number of ways with which to emulate parent/child node behaviour.

Using these solutions is considered advanced template editing.

THE SOURCEFABRIC WAY

USING TOPICS

Using topics is our preferred method as it is the easiest and simplest to implement and maintain.

This method relies on creating a parent topic, then creating new topics for use as subcategories. This has the added benefit of being able to have articles in multiple categories without the need for duplication and that support for translations is automatic.

In this example we're assuming you have created a main topic called 'Subsections' to hold all of your sub sections already.

A setup something like this would be sufficient:

Subsections
  News
  Business
  Politics
  Sport

NAVIGATION

To create a navigation element from the newly created topics you will have to do something like the following which supports two levels of nested subcategories:

```
{{ set_topic name="Subsections:en"}} or {{ set_topic identifier="<Topic ID>" }}
  {{ list_subtopics }}
    {{ if $gimme->current_list->at_beginning }}
      <ul>
    {{/if}}
       <li {{if $gimme->default_topic->identifier == $gimme->topic-
>identifier}} class="active" {{/if}} >

          <a href="{{ url options='section' }}?tpid={{$gimme->topic-
>identifier}}">{{ $gimme->topic->name }}</a>

{{ list_subtopics }}
           {{ if $gimme->current_list->at_beginning }}
             <ul>
           {{/if}}
               <li {{if $gimme->default_topic->identifier == $gimme->topic-
>identifier}} class="active" {{/if}} >

                    <a href="{{ url options='section' }}?tpid={{$gimme->topic-
>identifier}}">{{ $gimme->topic->name }}</a>

               </li>

          {{ if $gimme->current_list->at_end }}
            </ul>
          {{ /if }}
        {{ /list_subtopics }}
</li>
     {{ if $gimme->current_list->at_end }}
     </ul>
     {{ /if }}
   {{ /list_subtopics }}

<!-- here we reset topic back to default_topic. Otherwise our menu code would
affect article list -->

{{ set_default_topic }}
```

MORE SUBSECTION METHODS

Although the following approaches have been tried and tested they also are considered 'hacks' so please proceed with caution if you feel as though the above solution is inappropriate.

USING ISSUES

If your site is not being structured in a way that requires regular issues it might be possible to restructure your content using issues as your section level content containers and then those sections held within them become subsections.

Navigation

The default Newscoop functionality does not support using issues as navigation but with a little gentle massaging it will do it.

```
{{ unset_issue }}
{{ list_issues }}
  <li><a href='{{ url options="issue" }}'>{{ $gimme->issue->name }}</a>
    <ul>
    {{ list_sections order="bynumber asc" }}
      <li><a href="{{ url options='section' }}">{{ $gimme->section->name
}}</a></li>
    {{ /list_sections }}
    </ul>
  </li>
{{ /list_issues }}
```

USING CUSTOM SWITCHES

This is the most advanced way of creating subsections. Good thing about it is that You can use all solutions in the same time. There may be some situation when using custom switches will be helpful. One of those is when You want to list articles containing media like slideshows or videos. Using switches makes it really flexible because You can point that article with embedded Youtube should appear on "videos" listing page.

Let's assume that we want to create "Multimedia" section with audio on video subsections. We also assume that "audio" and "video" switches are already configured in used article type.

Navigation

```
<!-- we list all sections to create regular navigation -->

{{list_sections ...}}

   <!-- if loop will reach multimedia section we want to add those two
subsections based on switches -->

   {{if $gimme->section->number==<MULTIMEDIA_SECTION_NUMBER>}}

        <ul>

           <li><a href="{{ url options="section" }}?media=video"
>Video</a></li>
           <li><a href="{{ url options="section" }}?media=audio"
>Audio</a></li>

        </ul>

   {{/if}}

{{/list_sections}}
```

Template for multimedia section

What we need to do here is to retrieve media variable from GET array.

{{assign var="media_get" value=$smarty.get.media}}

{{if $media_get=="audio" || $media_get=="video"}}

```
   {{ assign var="publish_cond" value="`$media_get` is on" }}
{{else}}
  {{ assign var="publish_cond" value=" " }}
{{/if}}
```

Now we will use publish_cond variable to filter article list.

```
{{ list_articles  [...]  constraints="$publish_cond"}}
       ...
{{/list_articles}}
```

NUMERIC TRICK FOR SUB-SECTION SOLUTION

Another way of dealing with subtopics may be to simulate subtopic functionality with decade numbers for root topics, and the rest in between full decade numbers may represent subtopics. In other words, we can have something like this:

sec no.	root section	sub-section
10	Politics	
11		Domestic
12		Regional
13		International
20	Economy	
21		Agriculture
22		Coal-mining
30	Culture	
31		Applied arts
32		Gallery life
33		Celebrity gossip
34		Sculpture
40	Sports	
41		Football
42		Other sports
50	Lifestyle	
51		Cooking and dining
52		Urban gardening
53		Cherry picking

This structure may be used in templates in a way that on the root sections level we list only those divisible by zero, and as its subsctions those that follow - to the next number divideable by ten. (If you need really lot of subsections per root section, consider using divisins by 100 :)). So, next code is just an example how we can use this approach

```
{{ assign var="secno" value=10 }}
<ul>
{{ while $secno < 60 }}
{{ assign var="nextsecno" value=$nextsecno+10 }}
{{ set_section number=$secno }}
<li><a href="{{ url options="section" }}">{{ $gimme->section->name }}</a></li>
<ul>
{{ list_sections constraints="number gt $secno number lt $nextsecno }}
 <li class="subsection"><a href="{{ url options="section" }}">{{ $gimme-
>section->name }}</a></li>

{{ /list_sections }}

</ul>

{{ $secno = $secno+10 }}

{{ /while }}

</ul>
```

60. WAYS TO CREATE ARCHIVES

There are different approaches to creating archives. It can be:

- a list of all Articles
- a list of Issues
- a search form to look through all materials

You would usually combine different options or make your own, dependant on your needs.

PLACEHOLDER FOR THE ARCHIVE PAGE

A placeholder in this context is a series of articles with which to work with or none of our queries will return results.

The easiest way to create an independent placeholder for our archive page is to create an Issue with a number smaller than the first Issue that contains the content. Let's say it is going to be Issue number 1. Inside this issue we will create the section "Archive" and assign archive.tpl as a section template.

To put the archive link in menu we would use this code:

```
{{set_issue number="1"}}
  {{ list_sections constraints="" }}
    <li {{ if ($gimme->section->number == $gimme->default_section->number) }}
class="active" {{/if }}>
      <a href="{{url options="section"}}" title="">{{ $gimme->section->name
}}</a>
    </li>
  {{ /list_sections }}
  <!-- reset issue -->
{{set_current_issue}}
```

Now we are ready to work on the archive.tpl file.

PAGINATED LIST OF ALL ARTICLES SORTED BY PUBLICATION DATE

Listing all articles by publication date is very easy and is in fact the same as the section page. Firstly there is something we should do, list all the articles we will use with the regular list_articles function block but with some options.

```
{{ list_articles ignore_issue="true" ignore_section="true" length="16"
order="byPublishdate desc" constraints=" type is article"}}

...

{{/list_articles}}
```

ignore_issue is set to "true" because we want to show all articles from all issues. We also use ignore_section and we want to show only articles of type "article". You can of course use your own article types or not use constraints at all.

The trick is to make the paginate function paginate inside our placeholder section. But there is an easy workaround.

```
{{list_articles ...}}
  ...
  {{if $gimme->current_list->at_end}}
    <!-- we set our issue -->
    {{set_issue number="1"}}
    <!-- we need also set archive section -->
      {{set_section number="archive_section_number"}}
      <!-- after those simple steps we are ready to include file with regular
section pagination -->
      {{ include file="_tpl/pagination.tpl" }}
  {{/if}}
{{/list_articles}}
```

GENERAL ARCHIVE 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 here: http://newscoop-demo.sourcefabric.org/en/jan2011/?tpl=6

For this type of archive we will need to create two files. They can be **archive.tpl** and **archive_issue.tpl**.

Lets start with archive.tpl

What we want to do here is to list issues but NOT our placeholder issue number 1.

```
{{ list_issues constraints="number not 1" order="bypublishdate desc" }}
  <h3>
    <a href="{{ url options="template archive_issue.tpl" }}">{{ $gimme->issue-
>name }}</a>
  </h3>
  <h4>
    Published on <time datetime="{{ $gimme->issue-
>publish_date|date_format:"%Y-%m-%dT%H:%MZ" }}">
                                                  {{ $gimme->issue-
>publish_date|camp_date_format:"%d %M %Y" }}
                       </time>
  </h4>
{{ /list_issues }}
```

Notice that in the {{url}} block there is options="template archive_issue.tpl". This block will create a url to the archive_issue template. We don't have to pass an issue number or write options="issue template archive_issue.tpl" because issue is automaticaly set inside the {{list_issues}} loop.

archive_issue.tpl

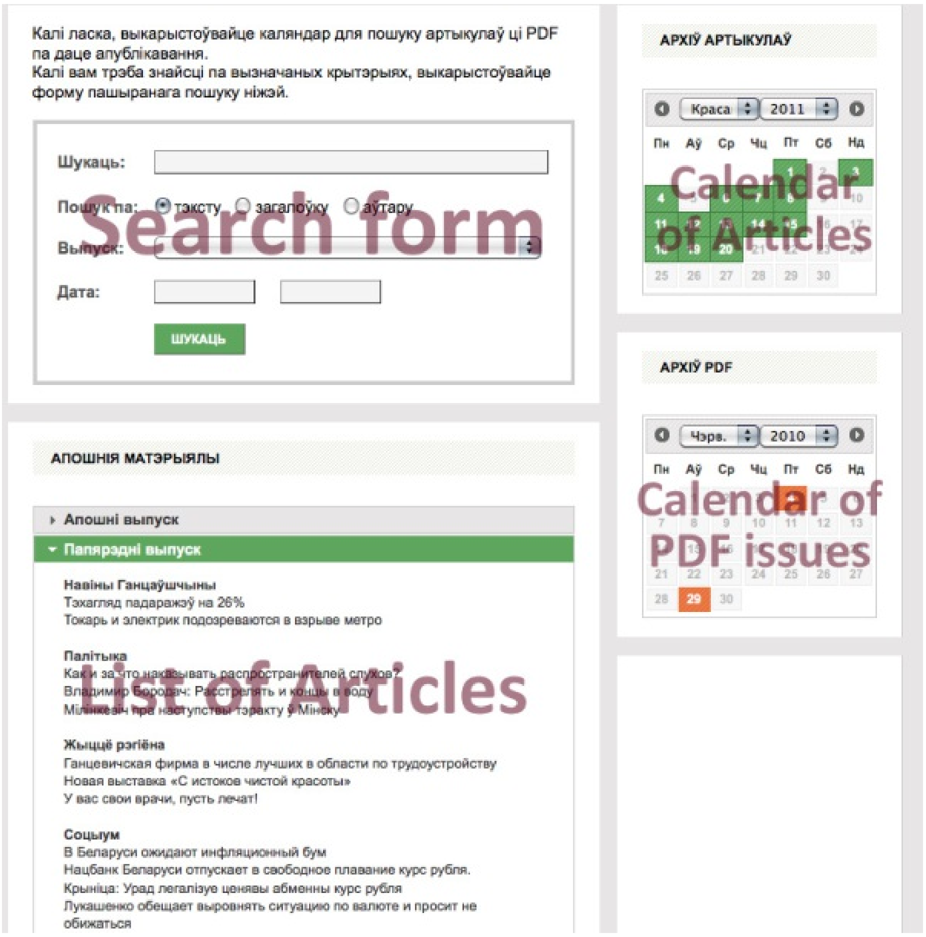
In this file we will list sections and articles inside them ordered by publication date.

```
<h1>{{ $gimme->issue->name }}</h1>

<div class="issue-content">

{{ list_sections }}

  {{ list_articles }}

    {{ if $gimme->current_articles_list->at_beginning }}

       <h3>{{ $gimme->section->name }}</h3>

        <ul>

   {{ /if }}

          <li>

              <a href="{{ url options="article" }}">{{ $gimme->article->name
}}</a>

              <time datetime="{{ $gimme->article->publish_date|date_format:"%Y-
%m-%dT%H:%MZ" }}">{{ $gimme->article->publish_date|camp_date_format:"%M %e, %Y"
}}</time>

          </li>

    {{ if $gimme->current_articles_list->at_end }}

        </ul>

    {{ /if }}

  {{ /list_articles }}

{{ /list_sections }}

</div>
```

And that is it!

CALENDAR BASED ARCHIVE AND ADVANCED SEARCH

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



Калі ласка, выкарыстоўвайце каляндар для пошуку артыкулаў ці PDF па даце апублікавання.
Калі вам трэба знайсці па вызначаных крытэрыях, выкарыстоўвайце форму пашыранага пошуку ніжэй.

Шукаць:

Пошук па: [✓ тэксту] загалоўку аўтару

Выпуск:

Дата:

АПОШНІЯ МАТЭРЫЯЛЫ

Апошні выпуск

Папярэдні выпуск

Навіны Ганцаўшчыны
Тэхагляд падаражэў на 26%
Токарь и электрик подозреваются в взрыве метро

Палітыка
Как и за что наказывать распространителей слухов?
Владимир Бородач: Расстрелять и концы в воду
Мілінкевіч пра наступствы тэракту ў Мінску

Жыццё рэгіёна
Ганцевичская фирма в числе лучших в области по трудоустройству
Новая выставка «С истоков чистой красоты»
У вас свои врачи, пусть лечат!

Соцыум
В Беларуси ожидают инфляционный бум
Нацбанк Беларуси отпускает в свободное плавание курс рубля.
Крыніца: Урад легалізуе ценявы абменны курс рубля
Лукашенко обещает выровнять ситуацию по валюте и просит не обижаться

Implementing a search-based archive is a good practice, as it gives your readers the opportunity to find the required material quickly. We use an Advanced Search form for that purpose (please take a look at the *Template Reference* for more information about *Advanced Search*).

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.

WORKING WITH THIRD PARTY TOOLS

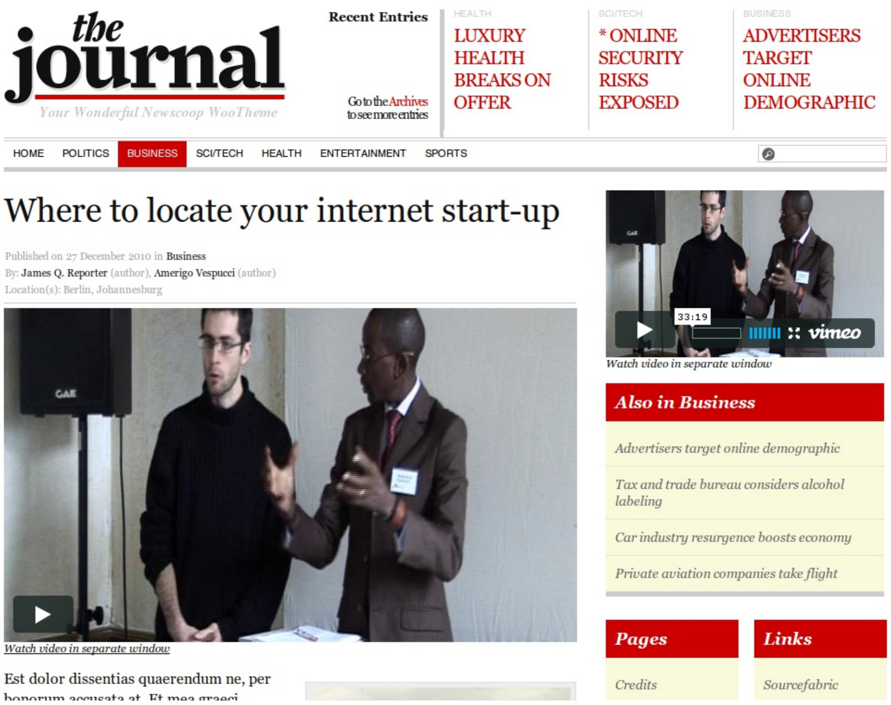
61. EMBEDDING YOUTUBE, VIMEO, ETC.

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...

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.


the journal
Your Wonderful Newscoop WooTheme
Recent Entries
Go to the Archives to see more entries
HEALTH
LUXURY HEALTH BREAKS ON OFFER
SCI/TECH
* ONLINE SECURITY RISKS EXPOSED
BUSINESS
ADVERTISERS TARGET ONLINE DEMOGRAPHIC
HOME
POLITICS
BUSINESS
SCI/TECH
HEALTH
ENTERTAINMENT
SPORTS
33:19
vimeo
Also in Business
Pages
Credits
Links
Sourcefabric

Where to locate your internet start-up

Published on 27 December 2010 in Business
By: James Q. Reporter (author), Amerigo Vespucci (author)
Location(s): Berlin, Johannesburg

Watch video in separate window

Watch video in separate window

Advertisers target online demographic

Tax and trade bureau considers alcohol labeling

Car industry resurgence boosts economy

Private aviation companies take flight

Est dolor dissentias quaerendum ne, per

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;portrait=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:

```
{{ if $gimme->article->vimeoid|strip_tags|trim !== "" }}
<iframe src="http://player.vimeo.com/video/{{ $gimme->article->vimeoid }}?
title=0&amp;byline=0&amp;portrait=0" width="300" height="180"
frameborder="0"></iframe>
<em><a href="http://vimeo.com/{{ $gimme->article->vimeoid }}"
target="_blank">Watch video in separate window</a></em>
{{ /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:

```
<script type="text/javascript">
var embedParts="{{ $gimme->article->youtubeurl }}".split("/");
document.write("<iframe title=\"YouTube video player\" width=\"435\"
height=\"356\" src=\"http://www.youtube.com/embed/"+embedParts[3]+"\"
frameborder=\"0\" allowfullscreen></iframe>");
</script>
```

RESPONSIVE VIDEOS

How to make our videos responsive and mobile ready? It is really easy. There are bunch of different approaches, libraries, plugins or small scripts. One of them is http://fitvidsjs.com/. Check it!

62. DISQUS AND FACEBOOK COMMENT SYSTEMS

You can use the commenting systems of Disqus and Facebook within your Newscoop site. Depending on your strategy regarding social media and user comments, these external system might be a better option than Newscoop's own commenting system. Facebook-driven comments appear on the profile page of the commenter thus driving new cirle of visitors to your site. The slick Disqus appearance might als be an argument to go with this third party service. Another advantage could be that this would allow to moderate comments without having to have access to the Newscoop backend.

One little warning from our side: comments are content. And the content you and your community generate on your site, should be on your site. If you rely on third party tools for comments, think about what would happen if these services either go out of business or change their licensing model or API. If this would happen, you would lose all your comments. This is the downside of third party services.

FACEBOOK COMMENTS

By the time of writing this chapter, the FB page covering instructions on how to set this up are here:

https://developers.facebook.com/docs/reference/plugins/comments/

To moderate, you need to list yourself as an admin. To do this, simply include open graph meta tags on the URL specified as the href parameter of the plugin. These tags must be included in the <head> of the document. Include:

<metaproperty="fb:admins"content="{YOUR_FACEBOOK_USER_ID}"/>
To add multiple moderators, separate the uids by comma without spaces.

If your site has many comments boxes, we strongly recommend you specify a Facebook app id as the administrator (all administrators of the app will be able to moderate comments). Doing this enables a moderator interface on Facebook where comments from all plugins administered by your app id can be easily moderated together. You can choose to specify either fb:app_id or fb:admins, but not both. This tag should be specified in the <head>.

<metaproperty="fb:app_id"content="{YOUR_APPLICATION_ID}"/>
You can moderate comments from just this plugin inline. If you have specified your app id as the admin, you can moderate all your plugins at http://developers.facebook.com/tools/comments.

First, you need to create an app on Facebook (https://developers.facebook.com/apps). You will then be able to customize options for look and feel of the comment form and comment listing, and to generate the embed code that requires your app id to use plugin on your pages.

```
<div id="fb-root"></div>
<script>(function(d, s, id) {
 var js, fjs = d.getElementsByTagName(s)[0];
 if (d.getElementById(id)) return;
 js = d.createElement(s); js.id = id;
 js.src = "//connect.facebook.net/en_GB/all.js#xfbml=1&appId=your_app_id_here";
 fjs.parentNode.insertBefore(js, fjs);
}(document, 'script', 'facebook-jssdk'));</script>
```

This code goes into the <head> part of the html page, while on the place where you need comment form to appear you only need to put this:

```
<div class="fb-comments" data-href="{{ url options="article" }}" data-
width="670" data-num-posts="10"></div>
```

Disqus comments

Disqus integration is pretty straightforward - after registering on their platform and submiting basic data about the website you're going to put Disqus comments on, you get embed code generated.

```
<div id="disqus_thread"></div>
 <script type="text/javascript">
 /* * * CONFIGURATION VARIABLES: THIS CODE IS ONLY AN EXAMPLE * * */

var disqus_shortname = 'your_site_shortname'; // Required - Replace example
with your forum shortname

var disqus_identifier = '{{ $gimme->article->number }}'; // a unique identifier
for each page where Disqus is present

var disqus_title = '{{ $gimme->article->name|escape }}'; // a unique title for
```

```
each page where Disqus is present

var disqus_url = '{{ url options="article" }}'; // a unique URL for each page
where Disqus is present

/* * * DON'T EDIT BELOW THIS LINE * * */
 (function() {
 var dsq = document.createElement('script'); dsq.type = 'text/javascript';
dsq.async = true;
 dsq.src = '//' + disqus_shortname + '.disqus.com/embed.js';
 (document.getElementsByTagName('head')[0] ||
document.getElementsByTagName('body')[0]).appendChild(dsq);
 })();
 </script>
```

This code goes where you want comments to appear (under the full article text).

More settings can be adjusted in the 'Settings' tab of your site's dashboard on Disqus. You can for example change color scheme, typeface, text shown for articles having 0 / 1 / 2 and more comments, community rules (moderation policy), social platforms integration etc.

63. 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:

```
{{ assign var="profile_email" value='$gimme->comment->reader_email' }}
{{ php }}
  $profile_email = $template->get_template_vars('profile_email');
  print "<img src=\"http://www.gravatar.com/avatar/".md5( strtolower( trim( 
$profile_email ) ) )."\" />";
{{ /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:

```
{{ assign var="profile_email" value=`$gimme->comment->reader_email` }}
{{ php }}
$profile_email = $template->get_template_vars('profile_email');
print "<img src=\"http://www.gravatar.com/avatar/".md5( strtolower( trim( 
$profile_email ) ) )."?s=200\" />";
{{ /php }}
```

FACEBOOK

Calling an image from Facebook is very straightforward. The example below calls in the Sourcefabric image but of course you should change the Sourcefabric value to your own username in all examples:

```
<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.

```
{{ assign var="profile_fbname" value='$gimme->comment->nickname' }}
{{ php }}
$profile_fbname = $template->get_template_vars('profile_fbname');
if (!preg_match("/error/i", 
file_get_contents("http://graph.facebook.com/".$profile_fbname."/picture"))) {
  print "<img src=\"http://graph.facebook.com/".$profile_fbname."/picture\"
/>";
}
{{ /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 is:

```
<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:

```
{{ assign var="profile_twname" value=`$gimme->comment->nickname` }}
{{ php }}
$profile_twname = $template->get_template_vars('profile_twname');
print "<img src=\"http://img.tweetimag.es/i/". $profile_twname ."\" />";
{{ /php }}
```

You might want to check if the name provided actually does exist:

```
{{ assign var="profile_twname" value=`$gimme->comment->nickname` }}
{{ php }}
$profile_twname = $template->get_template_vars('profile_twname');
$api_call = "http://twitter.com/users/show/".$profile_twname.".json";
```

```
$results = json_decode(file_get_contents($api_call));
if (!empty($results)) {
print "<img src=\"".$results->profile_image_url."\" />";
}
{{ /php }}
```

64. LOOKING GOOD FOR AND 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:

```
<script src="http://connect.facebook.net/en_US/all.js#xfbml=1"></script>
<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.

```
<title>{{ if $gimme->article->defined }}{{ $gimme->article->name }} | Your Site
{{ else }}Your Site
{{ /if }}
</title>
{{ if $gimme->article->defined }}
<meta property="og:title" content="{{$gimme->article-
>name|html_entity_decode|regex_replace:'/&(.*?)quo;/':'&quot;'}}" />
<meta property="og:type" content="article" />
<meta property="og:url" content="http://{{ $gimme->publication->site }}{{ uri
}}" />
<meta property="og:site_name" content="YourSite.com" />
{{ if $gimme->article->type_name == "news" }}
  <meta property="og:description" content="{{$gimme->article-
>deck|strip_tags:false|strip|escape:'html':'utf-8' }}" />
{{ elseif $gimme->article->type_name == "show" }}
  <meta property="og:description" content="{{$gimme->article-
>short_description|strip_tags:false|strip|escape:'html':'utf-8' }}" />
{{ /if }}
{{ list_article_images }}
<meta property="og:image" content="{{ $gimme->article->image->imageurl }}" />
{{ /list_article_images }}
{{ if $gimme->prev_list_empty }}
<meta property="og:image" content="http://{{ $gimme->publication->site
}}/templates/assets/images/logo.png" />
{{ /if }}
{{ /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|regex_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*:

```
{{ if $gimme->article->type_name == "news" }}
  <meta property="og:description" content="{{$gimme->article-
>deck|strip_tags:false|strip|escape:'html':'utf-8' }}" />
{{ elseif $gimme->article->type_name == "show" }}
  <meta property="og:description" content="{{$gimme->article-
>short_description|strip_tags:false|strip|escape:'html':'utf-8' }}" />
{{ /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:

```
{{ list_article_images }}
<meta property="og:image" content="{{ $gimme->article->image->imageurl }}" />
{{ /list_article_images }}
{{ if $gimme->prev_list_empty }}
<meta property="og:image" content="http://{{ $gimme->publication->site
}}/templates/assets/images/logo.png" />
{{ /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&am
scrolling="no" frameborder="0" allowTransparency="true" style="border:none;
overflow:hidden; width:400px; height:px"></iframe>
```

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:like
href="{{ uri options="article" }}" layout="button_count" show_faces="true"
width="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:

```
<div id="fb-root"></div>
<script
src="http://connect.facebook.net/en_US/all.js#appId=APP_ID&amp;xfbml=1"></script

<fb:comments href="sourcefabric.org" num_posts="2" width="500"></fb:comments>
```

More info about Facebook Comments can be found on the Facebook developer pages: http://developers.facebook.com/docs/reference/plugins/comments/

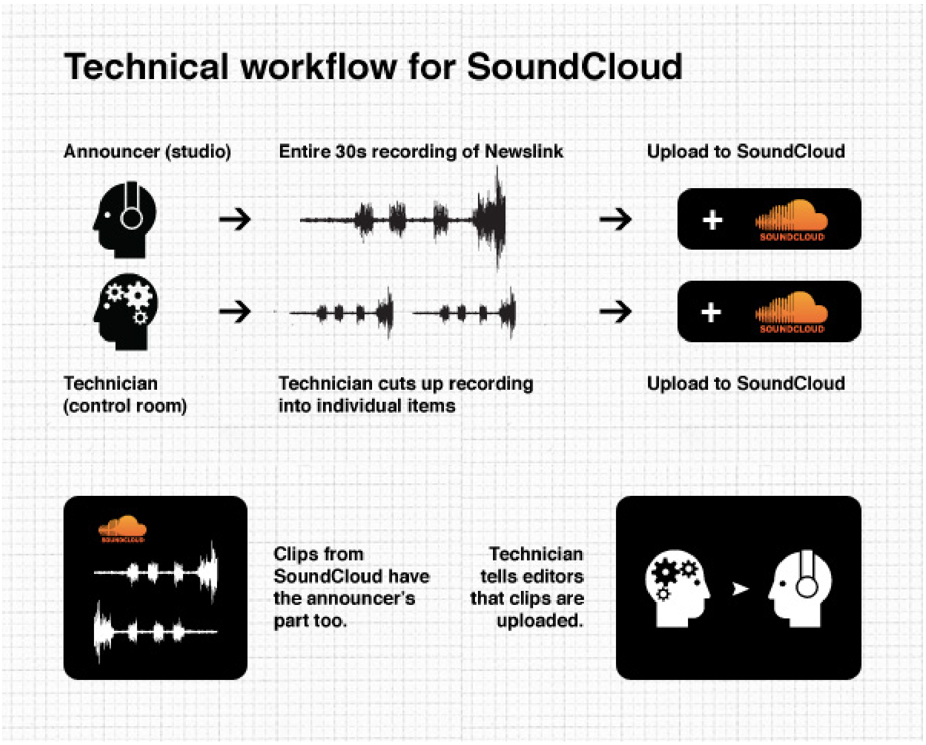
65. COMBINING NEWSCOOP AND SOUNDCLOUD

Newscoop 4 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.

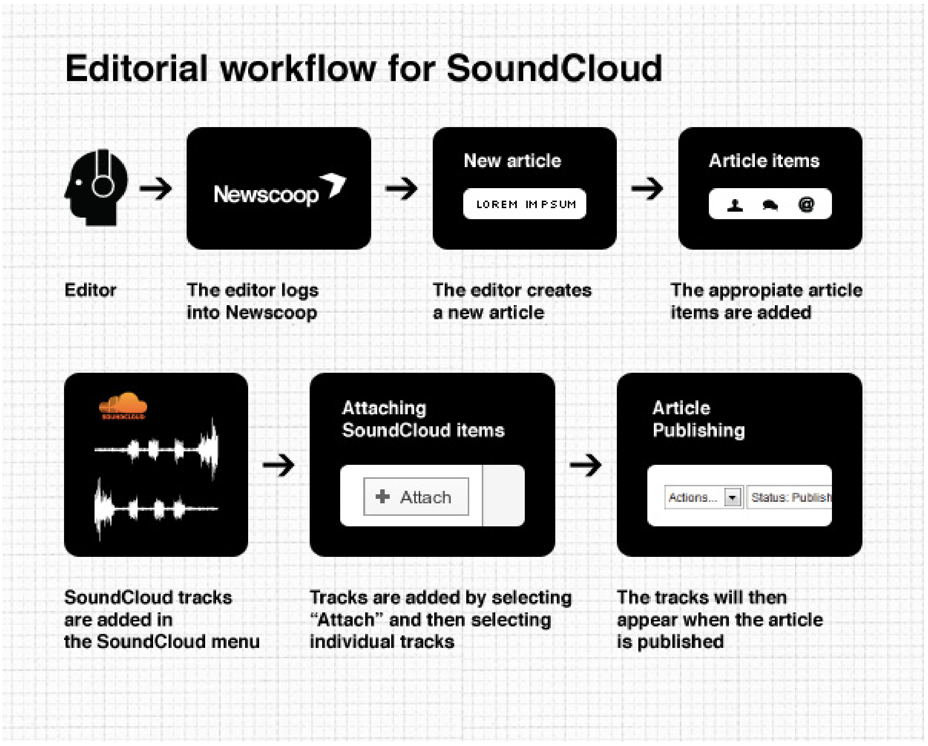
There were three major benefits for WADR in using SoundCloud as a distribution channel.

- 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:


Technical workflow for SoundCloud
Announcer (studio)
Entire 30s recording of Newslink
Upload to SoundCloud
Technician (control room)
Technician cuts up recording into individual items
Upload to SoundCloud
Clips from SoundCloud have the announcer's part too.
Technician tells editors that clips are uploaded.

On the editorial side, editors make use of the SoundCloud plugin to attach uploaded clips to their articles:


Editorial workflow for SoundCloud
Newscoop
New article
LOREM IMPSUM
Article items
Editor
The editor logs into Newscoop
The editor creates a new article
The appropiate article items are added
Attaching SoundCloud items
Attach
Article Publishing
Actions...
Status: Publish
SoundCloud tracks are added in the SoundCloud menu
Tracks are added by selecting "Attach" and then selecting individual tracks
The tracks will then appear when the article is published

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:

```
{{ list_soundcloud_tracks article=$gimme->article->number }}
<object height="81" width="100%">
<param name="wmode" value="transparent">
<param name="movie" value="http://player.soundcloud.com/player.swf?url={{
$soundcloud->track.secret_uri
}}&amp;show_comments=true&amp;auto_play=false&amp;></param>
<param name="allowscriptaccess" value="always"></param>
<embed allowscriptaccess="always" height="81"
src="http://player.soundcloud.com/player.swf?url={{ $soundcloud-
>track.secret_uri }}&amp;show_comments=true&amp;auto_play=false&amp;
type="application/x-shockwave-flash" width="100%"></embed>
</object>
{{ /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:

```
<div class="miniPlayer">
```

```
<object height="300" width="300">
{{ if $gimme->language->english_name == "English" }}
<param name="movie" value="http://player.soundcloud.com/player.swf?
url=http%3A%2F%2Fapi.soundcloud.com%2Fplaylists%2F662124&amp;auto_play=false&amp

<param name="allowscriptaccess" value="always"></param>
<param name="wmode" value="window"></param>
<embed wmode="window" allowscriptaccess="always" height="300"
src="http://player.soundcloud.com/player.swf?
url=http%3A%2F%2Fapi.soundcloud.com%2Fplaylists%2F662124&amp;auto_play=false&amp
type="application/x-shockwave-flash" width="300"></embed>
{{ else }}
<param name="movie" value="http://player.soundcloud.com/player.swf?
url=http%3A%2F%2Fapi.soundcloud.com%2Fplaylists%2F714819&amp;auto_play=false&amp
<param name="allowscriptaccess" value="always"></param> <embed
allowscriptaccess="always" height="300"
src="http://player.soundcloud.com/player.swf?
url=http%3A%2F%2Fapi.soundcloud.com%2Fplaylists%2F714819&amp;auto_play=false&amp
type="application/x-shockwave-flash" width="300"></embed>
{{ /if }}
</object>
</div>
```

For further reference, the SoundCloud API documentation is available at:

http://developers.soundcloud.com/docs/api

66. 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:

```
<script type='text/javascript'><!--//<![CDATA[
   var m3_u =
(location.protocol=='https:'?'https://openx.YOURSITE.COM/delivery/ajs.php':'http

   var m3_r = Math.floor(Math.random()*99999999999);
   if (!document.MAX_used) document.MAX_used = ',';
   document.write ("<scr"+"ipt type='text/javascript' src='"+m3_u);
   document.write ("?zoneid=66&amp;target=_top&amp;charset=UTF-8");
   document.write ('&amp;cb=' + m3_r);
   if (document.MAX_used != ',') document.write ("&amp;exclude=" +
document.MAX_used);
   document.write ('&amp;charset=UTF-8');
   document.write ("&amp;loc=" + escape(window.location));
   if (document.referrer) document.write ("&amp;referer=" +
escape(document.referrer));
   if (document.context) document.write ("&context=" +
escape(document.context));
   if (document.mmm_fo) document.write ("&amp;mmm_fo=1");
   document.write ("'><\/scr"+"ipt>");
//]]>--></script>
<noscript>
   <a href='http://openx.YOURSITE.COM/delivery/ck.php?
n=a16e8b28&amp;cb=INSERT_RANDOM_NUMBER_HERE' target='_top'>
   <img src='http://openx.YOURSITE.COM/delivery/avw.php?
zoneid=66&amp;charset=UTF-8&amp;cb=INSERT_RANDOM_NUMBER_HERE&amp;n=a16e8b28'
border='0' alt='' />
   </a>
</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:

```
<script type="text/javascript">
   <!--
   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:

```
_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="_banners/banner_hospitals.tpl" }}
{{ /if }}
{{ if $gimme->section->number == 11 }}
  {{ include file="_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="_banners/check_banner.tpl" }}
```

Using the Newscoop template language, you can modify the IF condition any way that fits your publication's needs.

TEMPLATE REFERENCE

67. 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.

68. 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.

```
{{include file='header.tpl'}}
{{include file='header.tpl' attrib_name='attrib value'}}
{{include file=$includeFile}}
{{include file=#includeFile# title='Smarty is cool'}}

 {{html_select_date display_days=yes}}
  {{mailto address='smarty@example.com'}}
   <select name='company_id'>
    {{html_options options=$companies selected=$company_id}}
   </select>
```

COMMENTS

Template comments are surrounded by asterisks, and then surrounded by the delimiter tags like this:

```
{{* this is a comment *}}
Comments are NOT displayed in the final output of the template, unlike <!--
HTML comments -->. These are useful for making internal notes in the templates
which no one will see.
{{* I am a template comment, I don't exist in the compiled output  *}}
<html>
<head>
<title>{{$title}}</title>
</head>
<body>
{{* another single line comment  *}}
<!-- HTML comment that is sent to the browser -->
{{* this multiline
   comment is
   not sent to browser
*}}
{{*******************************************************
Multi line comment block with credits block
  @ author:         bg@example.com
  @ maintainer:     support@example.com
  @ para:           var that sets block style
  @ css:            the style output
*******************************************************}}
{{* The header file with the main logo and stuff  *}}
{{include file='header.tpl'}}
{{* Dev note:  the $includeFile var is assigned in foo.php script  *}}
<!-- Displays main content block -->
{{include file=$includeFile}}
{{* this <select> block is redundant *}}
{{*
<select name="company">
  {{html_options options=$vals selected=$selected_id}}
</select>
*}}
<!-- Show header from affiliate is disabled -->
{{* $affiliate|upper *}}
{{* you cannot nest comments *}}
{{*
<select name="company">
  {{* <option value="0">-- none -- </option> *}}
  {{html_options options=$vals selected=$selected_id}}
</select>
*}}
{{* cvs tag for a template, below the 36 SHOULD be an american currency
. however its converted in cvs.. *}}
{{* &#36;Id: Exp &#36; *}}
{{* $Id: *}}
</body>
</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.

```
{{func var="test $foo test"}}       <-- sees $foo
{{func var="test $foo_bar test"}}   <-- sees $foo_bar
{{func var="test $foo[0] test"}}    <-- sees $foo[0]
{{func var="test $foo[bar] test"}}  <-- sees $foo[bar]
{{func var="test $foo.bar test"}}   <-- sees $foo (not $foo.bar)
{{func var="test `$foo.bar` test"}} <-- sees $foo.bar
{{func var="test `$foo.bar` test"|escape}} <-- modifiers outside quotes!
```

Here are some practical examples:

```
{{* will replace $tpl_name with value *}}
{{include file="subdir/$tpl_name.tpl"}}

{{* doesn't replace $tpl_name *}}
{{include file='subdir/$tpl_name.tpl'}}

{{* 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`"}}

{{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.

```
{{* apply modifier to a variable *}}
{{$title|upper}}

{{* modifier with parameters *}}
{{$title|truncate:40:'...'}}

{{* apply modifier to a function parameter *}}
{{html_table loop=$myvar|upper}}

{{* with parameters *}}
{{html_table loop=$myvar|truncate:40:'...'}}

{{* apply modifier to literal string *}}
{{'foobar'|upper}}

{{* using date_format to format the current date *}}
{{$smarty.now|date_format:"%Y/%m/%d"}}

{{* apply modifier to a custom function *}}
{{mailto|upper address='smarty@example.com'}}

{{* using  php's str_repeat *}}
{{'='|str_repeat:80}}

{{* php's count *}}
{{$myArray|@count}}

{{* php's shuffle on servers's ip *}}
{{$smarty.server.SERVER_ADDR|shuffle}}
```

```
{{* this will uppercase and truncate the whole array *}}
 <select name="name_id">
  {{html_options output=$myArray|upper|truncate:20}}
 </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:

```
{{$articleTitle}}
{{$articleTitle|upper|spacify}}
{{$articleTitle|lower|spacify|truncate}}
{{$articleTitle|lower|truncate:30|spacify}}
{{$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.

```
{{$foo}}        <-- displaying a simple variable (non array/object)

{{$foo[4]}}     <-- display the 5th element of a zero-indexed array

{{$foo.bar}}    <-- display the "bar" key value of an array, similar to PHP
$foo['bar']

{{$foo.$bar}}   <-- display variable key value of an array, similar to PHP
$foo[$bar]

{{$foo->bar}}   <-- display the object property "bar"

{{$foo->bar()}} <-- display the return value of object method "bar"

{{#foo#}}       <-- display the config file variable "foo"

{{$smarty.config.foo}} <-- synonym for {{#foo#}}

{{$foo[bar]}}   <-- syntax only valid in a section loop, see {{section}}

{{assign var=foo value='baa'}}{{$foo}} <--  displays "baa", see {{assign}}
```

Many other combinations are allowed:

```
{{$foo.bar.baz}}
{{$foo.$bar.$baz}}
{{$foo[4].baz}}
```

```
{{$foo[4].$baz}}
{{$foo.bar.baz[4]}}

{{$foo->bar($baz,2,$bar)}} <-- passing parameters

{{"foo"}}       <-- static values are allowed

{{* display the server variable "SERVER_NAME" ($_SERVER['SERVER_NAME'])*}}
{{$smarty.server.SERVER_NAME}}
```

TEMPLATE REFERENCE - OBJECTS

69. 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.

70. 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:

```
<table border="0" cellspacing="0" cellpadding="0" class="cs_img" align=left>
  <tr>
    <td align="center">
      <img src="/get_img?NrArticle=143&NrImage=1" border="0"
           hspace="5" vspace="5">
    </td>
  </tr>
  <tr><td align="center" class="caption">Newscoop team</td></tr>
</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:

```
<span class="articlesubhead">
    <a name="a1.250_s1">Version 2.3.0 - 2.3.1</a>
</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"

```
<audio controls>
    <source src="http://{{ $gimme->publication->site }}{{ uri
options="articleattachment" }}" type="{{ $gimme->attachment->mime_type }}">
</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

```
{{ list_article_comments columns="2" order="bydate desc"}}
{{ if $gimme->current_list->at_beginning }}
  <h2>{{ $gimme->article->comment_count }} Response(s) to &#8220;{{ $gimme-
>article->name }}&#8221;</h2>
  <ol class="commentlist">
{{ /if }}
   <li class="comment {{ if $gimme->current_list->column == "1" }}odd{{ else
}}even{{ /if }}">
      <div class="comment-head cl">
         <div class="user-meta">
             <strong class="name">{{ $gimme->comment->nickname }}</strong> {{
$gimme->comment->submit_date|camp_date_format:"%e.%m.%Y at %H:%i" }}
          </div>
      </div>
      <div class="comment-entry">
          <p>{{ $gimme->comment->content }}</p>
      </div>
   </li>
{{ if $gimme->current_list->at_end }}
  </ol>
{{ /if }}
{{ /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}}
  {{ list_article_locations }}
    {{ 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>
```

71. 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

```
{{ list_article_authors order="bylastname" }}
 <p class="author-head">
  {{ $gimme->author->type }}: {{ $gimme->author->name }}
 </p>
 <p class="author-bio">
  <img src="{{ $gimme->author->picture->imageurl }}" class="author-pic">
  {{ $gimme->author->biography->text }}
 </p>
{{ /list_article_authors }}
```

```
{{ if $gimme->article->defined }} by
 {{ $gimme->article->author->first_name }}
  {{ $gimme->article->author->last_name }}
   {{ $gimme->article->author->type }}
{{ /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

```
{{ if $gimme->article->defined }} by
 {{ $gimme->article->author->biography->name }}
  ({{ $gimme->article->author->type }})<br />
   <p>{{ $gimme->article->author->biography->text }}</p>
{{ /if }}
```

72. 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.

```
{{ if $gimme->browser->ua_type != "mobile" }}
  {{ assign var="browserdetect_mobile_device" value="false" }}
  {{ assign var="browserdetect_mobile_os" value="false" }}
  {{ assign var="browserdetect_mobile_os_number" value="false" }}
{{ else }}
  {{ assign var="mobile_data" value=`$gimme->browser->mobile_data` }}
  {{ php }}
  $mobile_data = $this->get_template_vars('mobile_data');
  $this->assign('browserdetect_mobile_device', $mobile_data[0]);
  $this->assign('browserdetect_mobile_os', $mobile_data[3]);
  $this->assign('browserdetect_mobile_os_number', $mobile_data[4]);
  {{ /php }}
{{ /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

```
{{ if $gimme->browser->browser_working == "moz" }}
  {{ assign var="browser_data" value=`$gimme->browser->moz_data` }}
  {{ php }}
  $browser_data = $this->get_template_vars('browser_data');
  $this->assign('browserdetect_name', $browser_data[0]);
  $this->assign('browserdetect_engine', "gecko");
  $this->assign('browserdetect_engineversion', $browser_data[2]);
  $this->assign('browserdetect_version', $browser_data[1]);
  {{ /php }}
{{ /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)]

```
{{ if $gimme->browser->browser_working == "webkit" }}
  {{ assign var="browser_data" value=`$gimme->browser->webkit_data` }}
  {{ assign var="browserdetect_engineversion" value=`$gimme->browser-
>browser_number` }}
  {{ php }}
  $browser_data = $this->get_template_vars('browser_data');
  $this->assign('browserdetect_name', $browser_data[0]);
  $this->assign('browserdetect_engine', "webkit");
  $this->assign('browserdetect_version', $browser_data[1]);
  {{ /php }}
{{ /if }}
```

73. 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

74. 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

75. 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:

```
{{ if $gimme->language == $gimme->default_language }}
...
{{ /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 }}
```

76. 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*.

77. 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

78. 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

79. 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

80. 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

81. 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

82. 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:

```
{{ if !$gimme->url->is_valid }}
 <h3>The requested page was not found.</h3>
{{ set_language name=`$gimme->publication->default_language->english_name` }}
  {{ set_current_issue }}
{{ else }}
    <!-- display content -->
{{ /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

83. 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

84. 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

85. 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:

```
{{ set_topic name="test:en" }}
 {{ list_articles }}
 ...
 {{ /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:

```
{{ set_topic name"test:en" }}
<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".

86. 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".

87. 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:

```
{{ local }}
 <list_of_instructions>
{{ /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

88. LIST 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 }} and {{ /list_articles }} is repeated for every article in the list.

SYNTAX:

```
{{ list_articles [length="<integer_value>"]
                 [columns="<integer_value>"]
                 [ignore_publication="true"]
                 [ignore_issue="true"]
                 [ignore_section="true"]
                 [ignore_language="true"]
                 [location="<longitude, latitude>]
                 [constraints="<list_of_article_constraints>"]
                 [order="<order_condition>"] }}
         <list_of_instructions>
{{ /list_articles }}
```

FILTERS:

- length="<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>" 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 using the current_list object, for example:

```
{{ 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

Constraints

<list_of_article_constraints> can include any of the following constraints:

- 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 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
- type: only articles having the given time 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. The topic name must be written in the format <topic_name>:<language_code> - for example *sports:en* or *health:en*.

  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:

  ```
  {{ set_topic name="sports:en" }}
  {{ list_articles }}
  ...
  {{ /list_articles }}
  ```

  This will list only articles having the topic 'sports'. The topic is automatically appended to the URL parameters so you don't have to set the topic in the current page.

- reads: you can set constraints based on the number of readers that viewed this article since it was published
- author: this works only in Newscoop 3.2.1 and newer versions. 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). 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"

Order

<order_condition> can include any of the following, in descending or ascending order:

- 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
- bycustom.<type>.<fieldname>.<defaultvalue> desc|asc

- byComments instructs 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
- bycustom type can be set in the second part after the dot: case insensitive strings set by

"ci", case sensitive strings set by "cs", or numerical set by "num". The field name is the third part, such as "deck". The fourth part is a default value for articles that do not contain the field name. An example could be: `{{ list_articles ... order="bycustom.ci.deck.aaa desc" constraints="..." }}` Note that the text area (i.e. multi-line text) data types usually contain some hidden html tags, which can affect the ordering, and that custom field ordering does not work for the (complex date) calendar-like fields of events.
- 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. bycustom was introduced in Newscoop 4.0.0.

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.

EXCEPTIONS:

<list_of_instructions> may contain any statement except those listed below. 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

The list_articles statement cannot be used inside list_subtitles, or list_search_results statements.

EXAMPLE:

Taken from theme "The Custodian" (date 2011-03-16) file "article.tpl"

```
{{ list_articles name="topic_articles"
constraints="number not `$gimme->article->number`
`$topic_cond` matchAnyTopic"  ignore_issue=true length=3 }}
  {{ include file="classic/tpl/teaserframe_articlelistright.tpl" }}
  {{ include file="classic/tpl/pagination.tpl" }}
  {{ assign var="number_cond" value="`$number_cond`
     number not `$gimme->article->number`" }}
{{ /list_articles }}
```

89. LIST 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 }} and {{ /list_article_attachments }} is repeated for every attachment in the list.

SYNTAX:

```
{{ list_article_attachments [length="<integer_value>"]
                            [columns="<integer_value>"]
                            [language="current"] }}
   <list_of_instructions>
{{ /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_article_attachments 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:

```
{{ list_article_attachments }}
 {{ if $gimme->current_list->at_beginning }}
  <h4>Downloads:</h4>
 {{ /if }}
  <a href="/attachment/{{ $gimme->attachment->identifier }}">
   {{ $gimme->attachment->file_name }}</a>
  ({{ $gimme->attachment->size_kb }}kb)<br/>
{{ /list_article_attachments }}
```

Here is an example of how to use the attachment list to play a number of MP3 files:

```
{{ list_article_attachments }}
 {{ if $gimme->attachment->extension == "mp3" }}
<object type="application/x-shockwave-flash"
data="/templates/radioactive/apps/player_mp3_maxi.swf" width="200" height="20">
```

```
<param name="movie" value="/templates/radioactive/apps/player_mp3_maxi.swf" />
<param name="bgcolor" value="#444444"/>
<param name="FlashVars" value="mp3={{ uri options="articleattachment" }}" />
<!-- player home: http://flash-mp3-player.net/ -->
</object>
 {{ /if }}
{{ /list_article_attachments }}
```

90. LIST 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:

```
{{ list_article_authors [length="<integer_value>"]
                         [columns="<integer_value>"]
                         [order="<author_order>"] }}
   <list_of_instructions>
{{ /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"

```
{{ list_article_authors }}
 <div id="hidden{{ $gimme->current_list->index }}Content"
class="teammemberinfo" style="display:none">
  <img style="width: 150px; float: left; margin: 0 10px 10px 0"
  src="{{ $gimme->author->picture->imageurl }}" />
   <h2>{{ $gimme->author->name }}</h2>
    <div class="text">
    {{ $gimme->author->biography->text }}
    </div>
</div>
{{ /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.

91. LIST 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:

```
{{ list_article_comments [ignore_language="true|false"]
                         [ignore_article="true|false"]
                         [length="<integer_value>"]
                         [columns="<integer_value>"]
                         [order="<order_condition>"] }}
   <list_of_instructions>
{{ /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:

```
{{ list_article_comments order="byDate desc" }}
    Subject: {{ $gimme->comment->subject }}<br/>
    Posted {{ $gimme->comment->submit_date }}
    by <b>{{ $gimme->comment->reader_email }}</b>
    <br/>
    {{ $gimme->comment->content }}
    <br/>
{{ /list_article_comments }}
```

The following example is taken from the theme "The Custodian" (2011-03-16) file comments.tpl

```
{{ list_article_comments }}
{{ if $gimme->current_list->at_beginning }}
 <a name="commentlist">
  <h4>
  {{ if $gimme->language->name == "English" }}Previous comments
  {{ else }}Los comentarios anteriores
  {{ /if }}
  </h4>
 </a>
{{ /if }}
 <div class="comment" {{ if $gimme->current_list->at_end }}
  id="everlast"{{ /if }}>
   <p><strong>{{ $gimme->comment->nickname }}</strong><br>
   {{ $gimme->comment->content }}</p>
    <p>
     <em>
      {{ $gimme->comment->subject }} |
      <span>{{ $gimme->comment->submit_date|camp_date_format:"%M
      %e, %Y" }}</span>
     </em>
    </p>
 </div>
<!-- /.comment -->
{{ /list_article_comments }}
```

Inside list_article_comments 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

92. LIST 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 {{ list_article_images }} 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.

93. LIST 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 {{ list_article_locations }} 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:

```
{{ list_article_locations [length="<integer_value>"]
                          [columns="<integer_value>"] }}
   <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 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:

```
<?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}}
  {{ list_article_locations }}
    {{ 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>
```

Taken from theme "The Journal" (date 2011-03-15) file "article-cont.tpl"

```
<p>
Location(s):
{{ list_article_locations }}
 {{ if $gimme->location->enabled }}
  {{ $gimme->location->name }}
   {{ if $gimme->current_list->at_end }}{{ else }}, {{ /if }}
 {{ /if }}
{{ /list_article_locations }}
</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.

94. 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:

```
{{ list_article_topics [length="<integer_value>"]
                       [columns="<integer_value>"] }}
  <list_of_instructions>
{{ /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"

```
{{ list_article_topics }}
   {{ assign var="topic_cond" value="`$topic_cond` topic is `$gimme->topic-
>identifier` " }}
{{ /list_article_topics }}
```

Taken from Template Pack "The Custodian" (date 2011-03-16) file "topic-list.tpl"

```
{{ list_article_topics }}
 {{ if $gimme->current_list->at_beginning }}
  {{ if $gimme->language->name == "English" }}Related topics
  {{ else }}Temas relacionados
  {{ /if }}
 {{ /if }}
 : {{$gimme->topic->name }}
 {{ if $gimme->current_list->at_end }}
 {{ /if }}
{{ /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:

```
{{ list_subtopics [length="<integer_value>"]
                  [columns="<integer_value>"]
                  [order="<order_condition>"] }}
    <list_of_instructions>
{{ /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

95. 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"

```
{{ list_issues length="10" order="byNumber desc" }}
 <h2>
  <a href="{{ url options="template classic/archive.tpl" }}">
  {{ $gimme->issue->name }}</a>
 </h2>
  {{ if $gimme->language->name == "English" }}Issue
  {{ else }}Edicin{{ /if }}
   #{{ $gimme->issue->number }} /
  ({{ if $gimme->language->name == "English" }}published
  {{ else }}publicado{{ /if }}
 {{ $gimme->issue->publish_date|camp_date_format:'%M %D, %Y %h:%i:%s' }})
 {{ include file="skins/greenpiece/includes/pagination.tpl" }}
{{ /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:

```
{{ list_sections [length="<integer_value>"]
                 [columns="<integer_value>"]
                 [constraints="<list_of_section_constraints>"] }}
   <list_of_instructions>
{{ /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"

```
{{ list_sections order="bynumber asc" }}
 <h3>
  <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.

96. 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:

```
{{ list_subtitles [length="<integer_value>"]
                  [columns="<integer_value>"] }}
   <list_of_instructions>
{{ /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

97. 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:

```
{{ list_images [length="<integer_value>"]
               [columns="<integer_value>"]
               [<filter>="<filter_value>"]}}
    <list_of_instructions>
{{ /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:

```
{{ list_images photographer="John\ Doe" order="byLastUpdate" }}

  <figure>
    <img src="{{ $gimme->image->thumbnailurl }}" /><br />
    <figcaption>
      <p>{{ $gimme->image->description }}</p>
    </figcaption>
  </figure>

{{ /list_images }}
```

98. 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

99. 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:

```
{{ list_search_results [length="<integer_value>"]
                       [columns="<integer_value>"]
                       [order="<order_condition>"] }}
   <list_of_instructions>
{{ /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

100. 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:

```
{{ search_form [template="<template_name>"] submit_button="<button_name>"
               [html_code="<html_code>"]
               [button_html_code="<html_code>"] }}
   <list_of_instructions>
{{ /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.

101. 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=\"submitbutton\""
}}
  <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:

```
{{ user_form [template="<template_name>"] submit_button="<button_name>"
             [html_code="<html_code>"]
             [button_html_code="<html_code>"] }}
   <list_of_instructions>
{{ /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:

```
{{ camp_edit object="search" attribute="<attribute>"
                             [html_code="<HTML_code>"]
                             [size ="<field_length>"]
                             [columns="<max_columns>"]
                             [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.

102. 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:

```
{{ subscription_form type="<subscription_type>"
                          [template="<template_name>"]
                          submit_button="<button_name>"
                          [total="<total_field_name>"]
                          [html_code="<html_code>"]
                          [button_html_code="<html_code>"] }}
   <list_of_instructions>
{{ /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.

103. 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 }}
```

104. 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

105. 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:

```
{{ camp_select object="<object_name>"
               attribute="<attribute_name>"
               [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:

```
<select name="f_user_country" id="countryId">
...
</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:

```
{{ camp_edit object="<object_name>"
             attribute="<attribute_name>"
             [html_code="<HTML_code>"]
             [size ="<field_length>"]
             [columns="<max_columns>"]
             [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="userNameInput">
```

TEMPLATE REFERENCE - MAPS AND GEOLOCATION

106. 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

```
{{ if $gimme->map->is_enabled }}
    <p>Map Name: {{ $gimme->map->name }}</p>
    <p>Map Provider: {{ $gimme->map->provider }}</p>
{{ /if }}
```

107. 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
//-->
    <div id="side_info"
class="geo_side_info">
      <!-- Map Location Nr. 1 START
//-->
      <div id="poi_seq_0">
        <a class="geomap_poi_name"
href="#"
onClick="geo_hook_on_map_feature_select(geo_object_61_1, 0);
return false;">POI no. 1</a>
        <div
class="geomap_poi_perex"></div>
        <div
class="geomap_poi_center">
          <a href="#"
onClick="geo_object_61_1.center_lonlat(14.753722843736,
48.948841006863); return false;">Center</a>
        </div>
        <div
```

```
class="geomap_poi_spacer">&nbsp;</div>
      </div>
      <!-- Map Location Nr. 1 END
//-->
      <!-- Map Location Nr. 2 START
//-->
      <div id="poi_seq_1">
        <a class="geomap_poi_name"
href="#"
onClick="geo_hook_on_map_feature_select(geo_object_61_1, 1);
return false;">POI Name</a>
        <div
class="geomap_poi_perex">Any text</div>
        <div
class="geomap_poi_center">          <a href="#"
onClick="geo_object_61_1.center_lonlat(1.240539250526,
47.067502513872); return false;">Center</a>
        </div>
        <div
class="geomap_poi_spacer">&nbsp;</div>
      </div>
      <!-- Map Location Nr. 2 END
//-->
    </div>
    <!-- Map Locations List END
//-->
  </div>
  <!-- Map general info and
locations Container START //-->
  <!-- Map Menu (Show Reset Link)
START //-->
  <div class="geomap_menu">
    <a href="#"
onClick="geo_object_61_1.map_showview(); return false;">Gimme
my map</a>
  </div>
  <!-- Map Menu (Show Reset Link)
END //-->
  <!-- Map Render Area START //-->
  <div class="geomap_map">
    <div
id="geo_map_mapcanvas_61_1"></div>
  </div>
  <!-- Map Render Area END //-->
</div>
<div style="clear:both"
></div>
```

Sourcefabric provides some styling definitions as part of the sample templates.

108. LOCATION MULTIMEDIA

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)

109. 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 <div> 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:

```
// going east to west over the 180/-180, and south to north
$p_coordinates = array();
$p_coordinates[] = array("longitude" => "150", "latitude" => "20");
$p_coordinates[] = array("longitude" => "40", "latitude" => "60");

// going directly west to east, and north to south
$p_coordinates = array();
$p_coordinates[] = array("longitude" => "-10", "latitude" => "60");
$p_coordinates[] = array("longitude" => "40", "latitude" => "-20");

$query = Geo_Map::GetGeoSearchSQLQuery($p_coordinates);
echo $query;
```

TEMPLATE REFERENCE - MODIFIERS

110. 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
- %l - 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 }}
```

111. 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

112. 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

113. 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:

```
{{ url [options="<options_list>"] }}
{{ uri [options="<options_list>"] }}
{{ uripath [options="<options_list>"] }}
{{ 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 }}" />
```

NEWSCOOP PLUGINS

114. EXAMPLE PLUGIN

114. EXAMPLE PLUGIN

Example plugin repository is in newscoop organization - https://github.com/newscoop/Plugin-ExamplePluginBundle.

The whole plugin system (installation/management) is based on Composer packages. Packages can live on github.com or your own private git repositories but they must be listed on packagist.org or privately owned composer repositories as long as they are based on satis.

For now we only support this way of managing plugins but we have plans for intoducing installation from zip archives.

The whole management process should be done through our Newscoop\Services\PluginsManagerService class. Importantly this way we allow for developers to react on installation, remove, update events, and more in their plugins.

INSTALLATION

```
php application/console plugins:install "newscoop/example-plugin-bundle"
```

The "Install" command will add your package to your composer.json file and install it. This command will also fire the "plugin.install" event with the parameter plugin_name in event data.

The plugins system is still under development, so for more information check our wiki: https://wiki.sourcefabric.org/display/CS/Plugins+system

NEWSCOOP API

115. INTRODUCTION TO NEWSCOOP API

Newscoop REST API is a basic REST API for Newscoop, which in its first version will allow you to only fetch content. Further on you will be able to give something back to Gimme by pushing data via write services.

For better understanding of what Gimme API is you can have a look at this FAQ.

METHODS OVERVIEW

We use HTTP methods to operate on these resources or collections of resources.

Method	Collection	Resource
GET	fetch resources	fetch resource
POST	create a resource in this collection	
PUT		create or update a resource here
PATCH		update part of a resource
DELETE		delete a resource

ENDPOINTS AND VERSIONS

example.com/api/

API endpoints don't have version mark in main endpoint uri, but for special reasons it can be allowed - ex. for debugging, or removing old API functions.

- example.com/api/articles
- example.com/api/v3/articles => example.com/api/articles

If API is no longer supported, then we redirect user to current API version with Location HTTP header (30x HTTP status codes that indicate redirection - http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.3).

PAGINATION AND SORTING OF COLLECTIONS

Default number of results on the page is **10**.

Allowed parameters:

Parameter name	value	Description
page	integer	results page number
sort	array	array of sort directions indexed by column name e.g. sort[name]=asc
items_per_page	integer	override number of results in the response

```
GET /articles?page={integer}&sort={array}&items_per_page={integer}
```

If response does not contain all of the result resources, we append to the response special pagination object:

PAGINATION

```
"pagination": {
  "itemsPerPage": {integer},
  "currentPage": {integer},
  "nextPageLink": {string}, // /articles/?page=2
  "previousPageLink": {string}, // /articles/?page=1
  "itemsCount": {integer}
}
```

PARTIAL RESPONSE

If client doesn't want whole response object, then he can choose particular fields:

```
GET /articles?fields=title,date,number
```

Parameter name	value	Description
fields	string	Comma separated list of properties of requested objects.

NULL VALUES IN RESPONSE

Keys with **null** value will be removed from response.

RESPONSE FORMAT

We use the **Accept** and **Content-Type** headers to agree what formats to use. But for now we support only **application/json.**

Response codes

Code	Message	Description
200	OK	Success!
304	Not Modified	There was no new data to return.
400	Bad Request	The request was invalid. An accompanying error message will explain why. This is the status code will be returned during rate limiting.
401	Unauthorized	Authentication credentials were missing or incorrect.
403	Forbidden	The request is understood, but it has been refused. An accompanying error message will explain why. This code is used when requests are being denied due to update limits.
404	Not Found	The URI requested is invalid or the resource requested, such as a user, does not exists.
500	Internal Server Error	Something is broken.
502	Bad Gateway	API is down or being upgraded.
503	Service Unavailable	API servers are up, but overloaded with requests. Try again later.
504	Gateway timeout	The API servers are up, but the request couldn't be serviced due to some failure within our stack. Try again later.

ERROR RESPONSE

```
{
  "errors":[
    {
      "message":"Something is broken.",
      "code":500
    }
  ]
}
```

116. NEWSCOOP API SDK'S

Newscoop REST API have two official sdk's:

- Newscoop REST API PHP SDK - https://github.com/sourcefabric/newscoop-api-php-sdk
- Newscoop REST API JS SDK - https://github.com/sourcefabric/newscoop-api-js-sdk

PHP SKD

PHP-SDK is a library that simply allows access to the Newscoop REST API. You can easily install it in any php project with Composer.

There is realy great simple tutorial where you can see and try how to work with this sdk: http://mikolajczuk.tumblr.com/post/34632137832/newscoop-rest-api-sdk-usage

JS SDK

Use Newscoop REST API easly in Your Newscoop theme templates with Java Script.

First you must include small library (you can find newscoop.js file on JS SDK github repository.

```
<script src="newscoop.js"></script>
```

After this you can start use it:

```
var api = new NewscoopRestApi('http://newscoop.dev/api');

api.getResource('/articles', {'type': 'news'})
    .setItemsPerPage(5)
    .setOrder({'number': 'asc'})
    .makeRequest(function(res){
        // callback
    });
```

With this example You will have data (inside makeRequest callback function, under res variable) for 5 latest articles with "news" type.

APPENDIX

117. CREDITS

117. CREDITS

Newscoop is being developed by an international community of developers and designers led by Sourcefabric.

http://newscoop.sourcefabric.org

This version of the manual was written by the Newscoop team, edited by Daniel James and reviewed by Trevor Parsons, using Booktype Pro. Please add your comments and contributions at: http://sourcefabric.booktype.pro/newscoop-42-cookbook/

LICENSE

---

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.

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