Content (CMS)

CMS (15:33)
Play
Part 1: Content Management System Overview

Overview

Content management system (CMS) allows to enrich e-commerce website with business driven content. Generally speaking this enrichment encompasses the following functions:

- Creation of content **pages**
- Creation of content **includes** (chunks of pages to be used in templates of a theme used by shop)

CMS allows to create micro sites as well create navigation menus, promotion banners, customer messaging, which are specific to each shop instance. Each content object can belong only to a single shop.

Whenever a new shop instance is created CMS section of admin is populated with an entry to access the root content object of the specific shop instance. Child content objects can be attached to this root with child content objects of their own to form content hierarchies with unlimited level of depth.

Page

The layout of the content page is determined by theme's content page template. Most commonly these templates include a fixed structure with header, body section and footer. The body of the content object is inserted into the body section. Template is resolved from the **content template** configuration of content object. Default theme defines two of such templates: **content** and **dynocontent** which use content body as **plain HTML** and **HTML with groovy scripts** respectively.

However custom theme implementation can define its own content templates, which could use more complex rendering of pages.

Include

Include refers to a special **content template** configuration **include**. The purpose of include is to specify internal content objects that are private (i.e. not available on storefront).

Main use cases are:

- Grouping content in admin for better management
- Creating a logical root of hierarchy (i.e. include content object can have public content objects children thus creating a public content branch, effectively a microsite section)
- Theme template includes - predefined zones in theme templates that allow configuration of page elements (e.g. meganav, footer, cart page configurable elements)

External CMS

The platform content service engine does not force business users to use built in CMS. Users that have existing CMS (e.g. Alfresco, Adobe

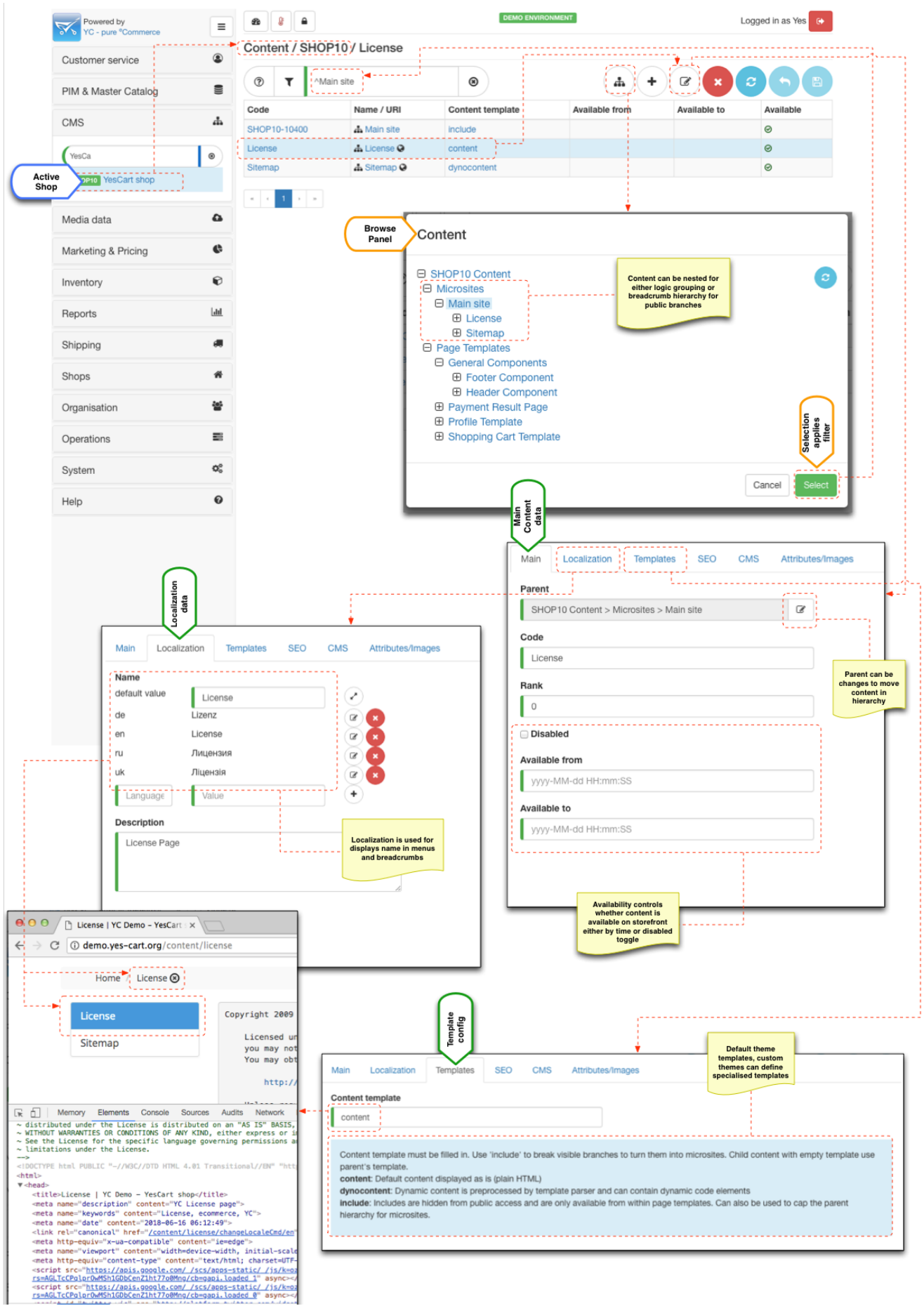
CMS) have two integration options:

- Plug into the platform content service layer (via custom content service adapter) to retrieve content from their systems.
- Use external CMS as their primary website engine and then use REST API as data processing engine bypassing content and themes altogether.

Content management

CMS management is focused at managing content for a specific shop at a time.

There is option of browsing the content hierarchy of a shop or search using keywords. Search filter can also accept special characters that instruct search to behave in a certain way (e.g. prefixing search with ^ would result in displaying content matching the search keyword and its immediate children). Use the help button on filter to see more options available for searching.


Powered by YC - pure eCommerce
DEMO ENVIRONMENT
Logged in as Yes
Content / SHOP10 / License
Customer service
PIM & Master Catalog
CMS
YesCart shop
Active Shop
Media data
Marketing & Pricing
Inventory
Reports
Shipping
Shops
Organisation
Operations
System
Help
^Main site
Code
Name / URI
Content template
Available from
Available to
Available
SHOP10-10400
Main site
include
License
content
Sitemap
dynocontent
Browse Panel
Content
SHOP10 Content
Microsites
Main site
License
Sitemap
Page Templates
General Components
Footer Component
Header Component
Payment Result Page
Profile Template
Shopping Cart Template
Content can be nested for either logic grouping or breadcrumb hierarchy for public branches
Selection applies filter
Cancel
Select
Main Content data
Main
Localization
Templates
SEO
CMS
Attributes/Images
Parent
SHOP10 Content > Microsites > Main site
Code
License
Rank
0
Disabled
Available from
yyyy-MM-dd HH:mm:SS
Available to
yyyy-MM-dd HH:mm:SS
Parent can be changes to move content in hierarchy
Availability controls whether content is available on storefront either by time or disabled toggle
Localization data
Name
default value
License
de
Lizenz
en
License
ru
Лицензия
uk
Ліцензія
Language
Value
Description
License Page
Localization is used for displays name in menus and breadcrumbs
License | YC Demo - YesCart
demo.yes-cart.org/content/license
Home
License
License
Sitemap
Copyright 2009
Memory
Elements
Console
Sources
Audits
Network
Template config
Content template
content
Content template must be filled in. Use 'include' to break visible branches to turn them into microsites. Child content with empty template use parent's template.
content: Default content displayed as is (plain HTML)
dynocontent: Dynamic content is preprocessed by template parser and can contain dynamic code elements
include: Includes are hidden from public access and are only available from within page templates. Can also be used to cap the parent hierarchy for microsites.
Default theme templates, custom themes can define specialised templates

Content object editor has a number of tabs each containing function specific configurations.

Mian tab allows to move content in the hierarchy by changing the parent (more details on setting up hierarchies are in this cookbook). Also content availability options can be set either temporal or by toggling disabled flag. Only available content will be publicly visible on frontend.

Localisation contains settings for content name that is displayed in menus and breadcrumbs.

Templates tab contains content template that defines how the content body is used.

Templates *	Accessible via URL	Rendering
content	✓	Basic layout of web page with content menu. Content body fills the middle section of the page
dynocontent	✓	Dynamic content is enhanced version of "content" that treats content body as template that can contain variables and call custom functions. Variables available to template depend on the theme.
include	✗	Content which is non accessible via public URL used to fill in placeholders in web pages defined by theme. Another usage is splitting content hierarchies into sub hierarchies for purpose of better content management in YUM and creation of distinct microsites.

* Custom themes can include other templates for alternative render process

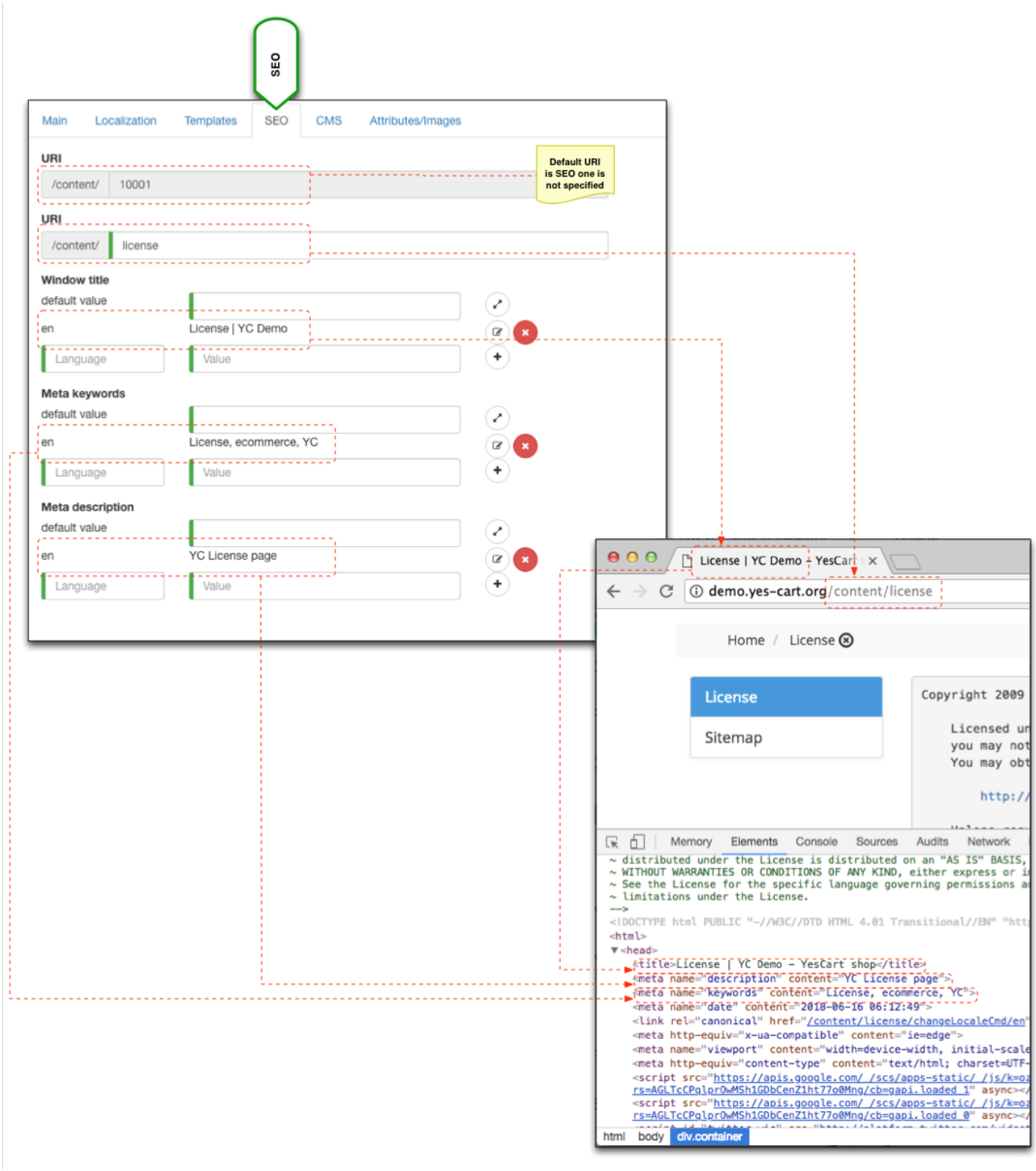
SEO - search optimisation engine configurations.

CMS - actual content that is used according to template configuration

Attributes/Images tag - for additional configurations that can be used by this content object

SEO

SEO tab allows full control over locale specific settings. Options allow to manipulate URI and meta tags.



Content body

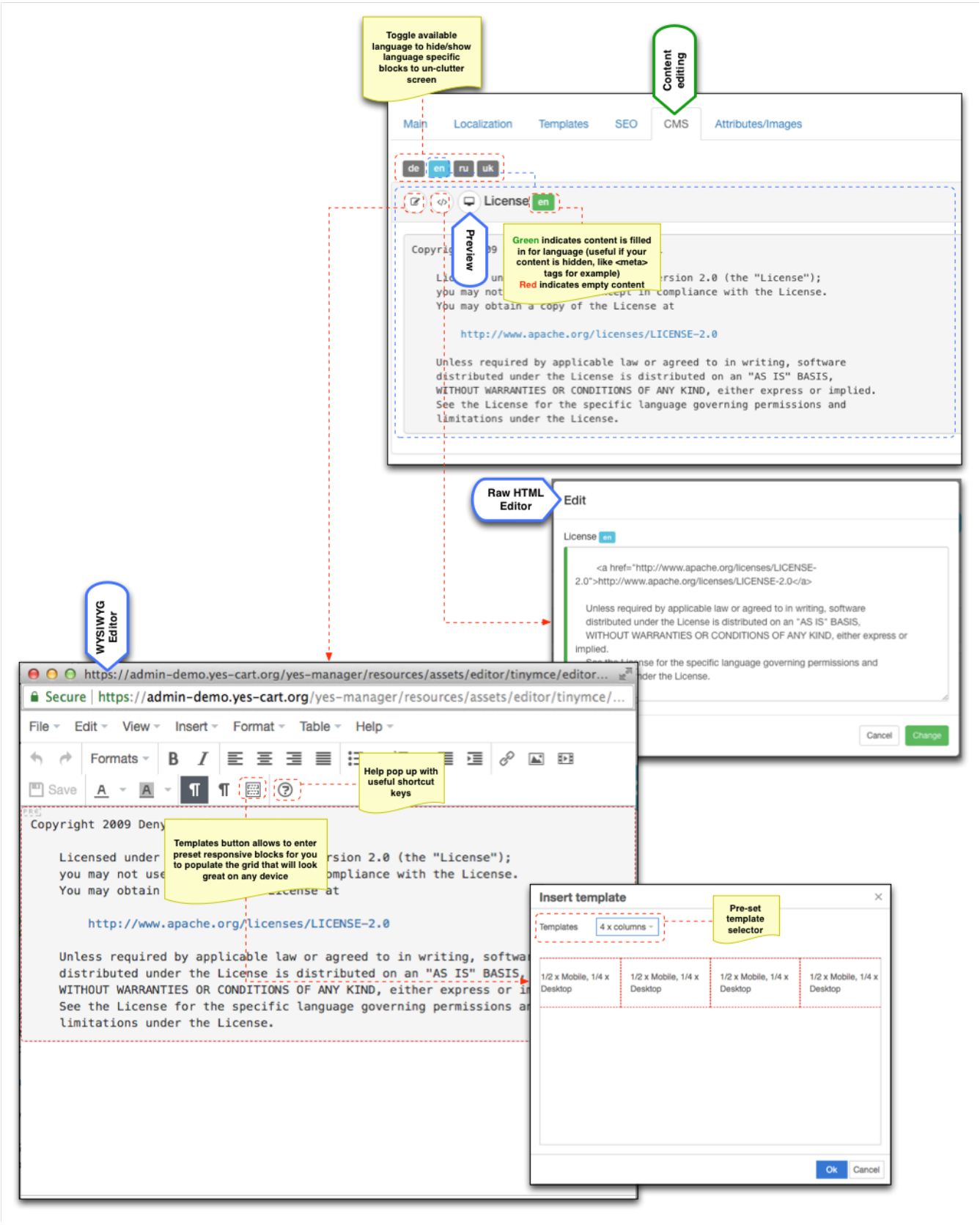
Content body is optional and is reserved only for content objects that render content on storefront. Typically these are content pages and content includes that are defined by templates.

At the top of the tab are locale toggle buttons. These buttons do not disable locales, they are used to show/hide language blocks to make this screen a bit more manageable when working with large content bodies (to enable/disable shop locales see shop configurations).

Each language block has a heading bar with three tools: WYSIWYG editor, raw editor and preview. In addition to this it contains a language

label that is either green (for non empty content body) or red (for empty content body). This is useful when content body does not contain visible element (e.g. scripts or meta tags).

Lower section of the content block shows a preview but it differs from the actual design. In order to see this body as it would be visible in shop you must use **preview** tool. This tool includes the **yc-preview.css** that contains full CSS bundle of the theme and thus will display HTML as closely as possible to the end result. Because preview tool opens in a new window you can also resize it to see responsive design in action.

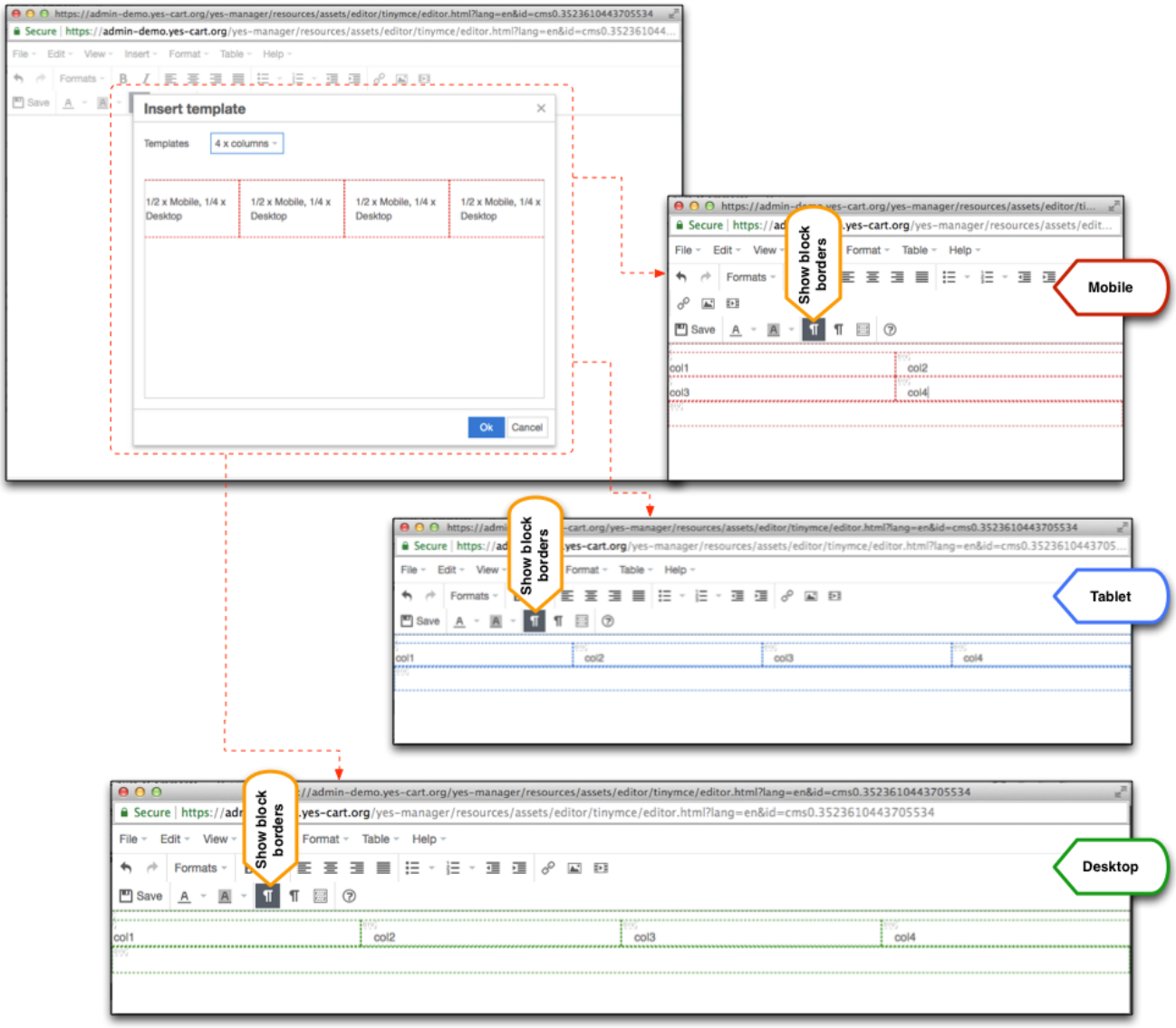


Raw data editor allows raw source manipulation and is intended for use by professional web designers in order to fine tune design and add non standard attributes that may be used by JavaScript functions or other external tools such as Google tag manager or other analytics engines.

WYSIWYG editor is intended for use by business users and has advanced features to allow user friendly web design experience. **Insert template** function allows to insert predefined responsive templates to arrange blocks of HTML. User should familiarise with hot keys and tips that can be viewed using **help button** in the editor.

WYSIWYG

By default this editor has **Show block borders** toggled, which draws a dotted line around blocks of HTML (e.g. div, p, pre). This allows to visualise blocks of HTML and how they behave when size of screen changes. Green indicates desktop size, blue - tables and red - mobile.



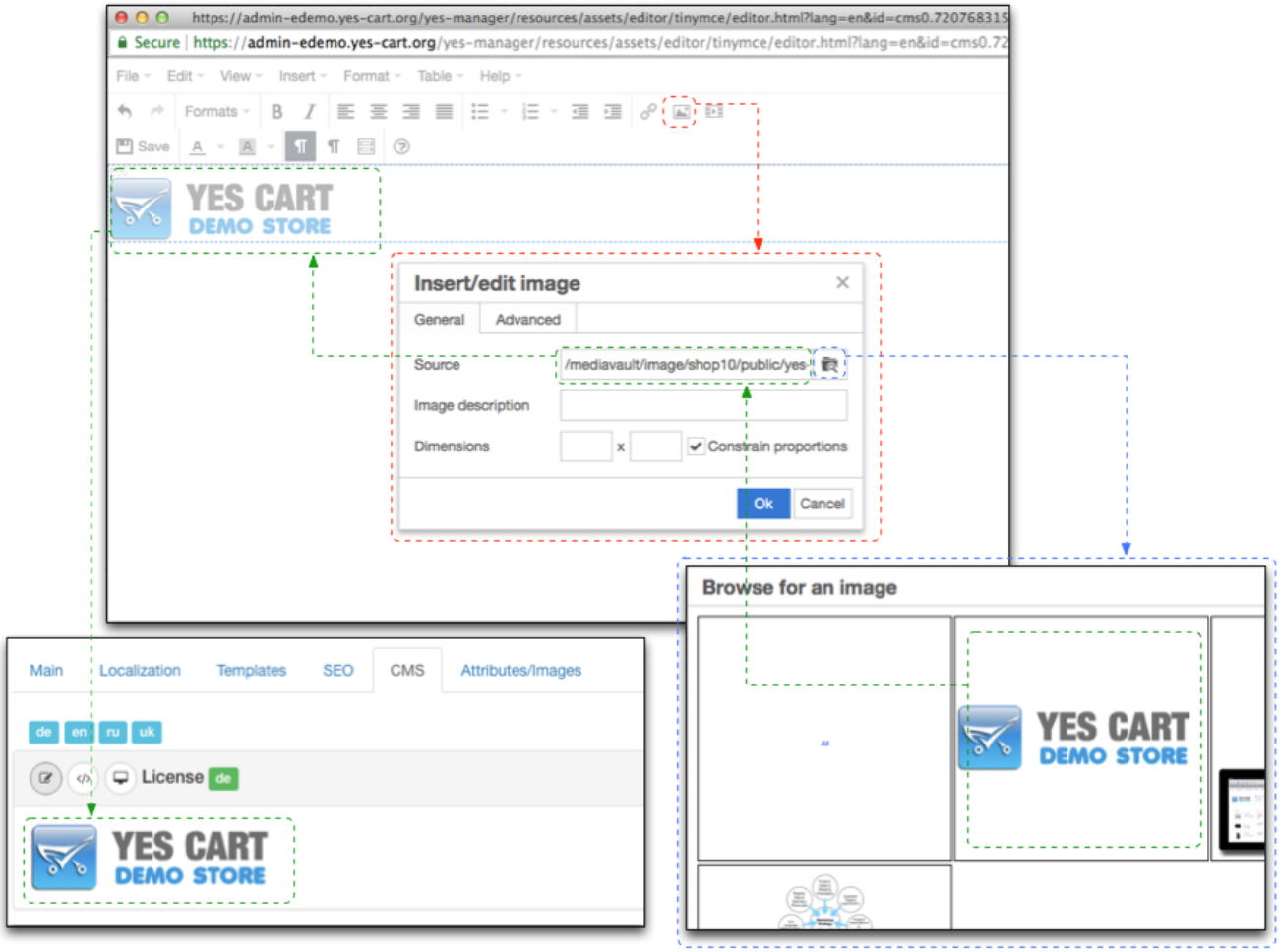
Editor is displayed in a separate window, so it can be resized at any point to visualise how HTML reacts to different view ports.

> Note that working with blocks ENTER key creates a new line within each block. Thus to escape out of the block to add new one use hot key "SHIFT+ENTER". Use help button to view full list of shortcuts and tips.

Media repository YCE 3.5.0+

Media repository is YCE integration for CMS WYSIWYG editor that allows to browse uploaded media files. In open source version only files and images that are attached as attributes to specific content objects can be used by copying generated URI. This integration allows to upload media unrelated to data objects and also browse and select it from WYSIWYG.

CMS in Yes Cart (5:57)
Play
Part 2: Media Management



Using functions

> We strongly recommend using only **raw editor** for content that uses functions or scripts

If you recall in **content template** configurations we had an option of using **dynocontent**. This type of template is an advanced version of **con tent** template that treats the body not just as plain HTML but as a mix of HTML and Groovy script and/or Thymeleaf template 3.6.0+ . This enables use of predefined functions and also some advanced scripting in the content to make the content dynamic (i.e. enriched with server side data, such as product details, category details, cart details etc).

The following built in functions are available in core code:

Version	Function	Param	Example

1.x.x	include	uri	<pre># Groovy style ${include('license')} # Thymeleaf style (3.6.0+) <div th:remove="tag" th:utext="${include.func('license')}"></pre>
1.x.x	contentURL *	uri	<pre># Groovy style License page # Thymeleaf style (3.6.0+) License page</pre>
1.x.x	categoryURL *	uri	<pre># Groovy style Notebooks # Thymeleaf style (3.6.0+) Notebooks</pre>
1.x.x	productURL *	uri { fc, uri } 3.7.0+	<pre># Groovy style TS-231 + 2X ST2000VN001 # Groovy style (3.7.0+) <a href="${productURL(['Main','158'])}">TS-231 + 2X ST2000V # Thymeleaf style (3.6.0+) <a href="[[${productURL.func('158')}]]">TS-231 + 2X ST2000V # Thymeleaf style (3.7.0+) TS-231 + 2 ST2000VN001</pre>

1.x.x	skuURL *	uri
		
		{ fc, uri } 3.7.0+
			`# Groovy style`
			`<a href="${skuURL('82')}">ME181C-A1-WT</a>`
			`# Groovy style (3.7.0+)`
			`<a href="${skuURL(['Main','82'])}">ME181C-A1-WT</a>`
			`# Thymeleaf style (3.6.0+)`
			`<a href="[[${skuURL.func('82')}]]">ME181C-A1-WT</a>`
			`# Thymeleaf style (3.7.0+)`
			`<a href="[[${skuURL.func(['Main','82'])}]]">ME181C-A1-WT</a`

1.x.x	URL *	uri	`# Groovy style`
			`<a href="${URL('')}">Home</a>`
			`# Thymeleaf style (3.6.0+)`
			`<a href="[[${URL.func('')}]]">Home</a>`

3.7.0+	encodeURI	uri	`# Groovy style (3.7.0+)`
			`<a href="${URL('?code=' + encodeURI('unsafe'))}">Home</a>`
			`# Thymeleaf style (3.7.0+)`
			`<a href="[[${URL.func('?code=' + encodeURI.func('unsafe'))}`

3.7.0+	decodeURI	uri	`# Groovy style (3.7.0+)`
			`<span>"${decodeURI('unsafe')}</span>`
			`# Thymeleaf style (3.7.0+)`
			`<span u:text="[[${decodeURI.func('unsafe')}]]"></span>`

3.7.0+
 	
SaaS
	filteredURL	uri	`# Groovy style (3.7.0+)`
			`<span>"${filteredURL('extra/path')}</span>`
			`# Thymeleaf style (3.7.0+)`
			`<span u:text="[[${filteredURL.func('extra/path')}]]"></spar`

* It is not necessary to use functions for links, you can simply use Notebooks instead of Notebooks. However functions become useful when you have multiple environments and some of them are not running from root (e.g. www.mydomain.com) but from a sub root (www.mydomain.com/yes-shop). In such setups above stated functions will automatically resolve "/yes-shop" and prepend it to all generated URLs

For technical users: custom functions can be injected into ContentServiceTemplateSupport through registerFunction so you can add your own functions when customising content service API

URL include SaaS

URL include is an advanced feature of SaaS CMS rendering. It allows to include any arbitrary content by referencing it inside script tag of type **yd-include**. For example:

```
<script
type='yc-include'>/internal/custom/controller/?param1=value1&param2=valu
e2</script>
```

This feature **is not the same as include function from dynocontent template** and serves a multitude of purposes.

- include function runs within scope of content it was called from (i.e. like a script). So all parameters that are passed to included content are exactly the same and thus there is a dependency on the parameter input in included. On the contrast url includes run in a separate server side sub request and allows to pass additional parameters (e.g. param1 and param2 in the example above). There is no limit on the parameters but only simple types can be passed (i.e. not objects) but plain text
- url includes do not depend on dynocontent and thus can be included in any content anywhere thus allowing to embed custom components. For example if you developed an element such as featured product gallery, on internal controller "/internal/productgalery", which accepts a list of product SKU, then this component can be embedded anywhere in content by reference <script type='yc-include'>/internal/productgalery?sku=SKU_A|SKU_B|SKU_C</script> and therefore any such gallery can be placed in any content by simply including one line. See featured products sliders on demo http://edemo.yes-cart.org which use this concept, the same component is included several times with list of SKU to generate different sliders of complete product pods
- lastly url includes allow to fine tune page caching (a feature) or SFG implementation that boosts performance enormously by caching chunks of generated HTML for given URI

Groovy scripts

> We strongly recommend using only **raw editor** for content that uses functions or scripts

Scripting is beyond the scope of this overview, see this cookbook for more technical in-depth details

Thymeleaf templates 3.6.0+

> We strongly recommend using only **raw editor** for content that uses functions or scripts

Thymeleaf templates is beyond the scope of this overview, see official documentation for more technical in-depth details

CMS Customisation

CMS API exposed via service layer (namely ContentService) that provides an interface for all other services that are dealing with views. Therefore it possible to override content service provider in order to customise the CMS whether to use custom implementation or to use this service as proxy to external CMS.

Since 3.6.0+ it is possible to configure CMS provider via configuration. Your custom modules can be injected via extension points and then activated via system configurations (System > Configurations).

For example use of CMS.v3 can be configured like so:

```
CMS.contentService=contentServiceCMS3
CMS.dtoContentService=dtoContentServiceCMS3
CMS.contentFileNameStrategy=contentCMS3FileNameStrategy
CMS.contentImageNameStrategy=contentCMS3ImageNameStrategy
```

Using legacy CMS.v1 can be configured like so:

```
CMS.contentService=contentServiceCMS1
CMS.dtoContentService=dtoContentServiceCMS1
CMS.contentFileNameStrategy=contentCMS1FileNameStrategy
CMS.contentImageNameStrategy=contentCMS1ImageNameStrategy
```