Enonic XP Documentation

Release 6.8.1

Enonic AS

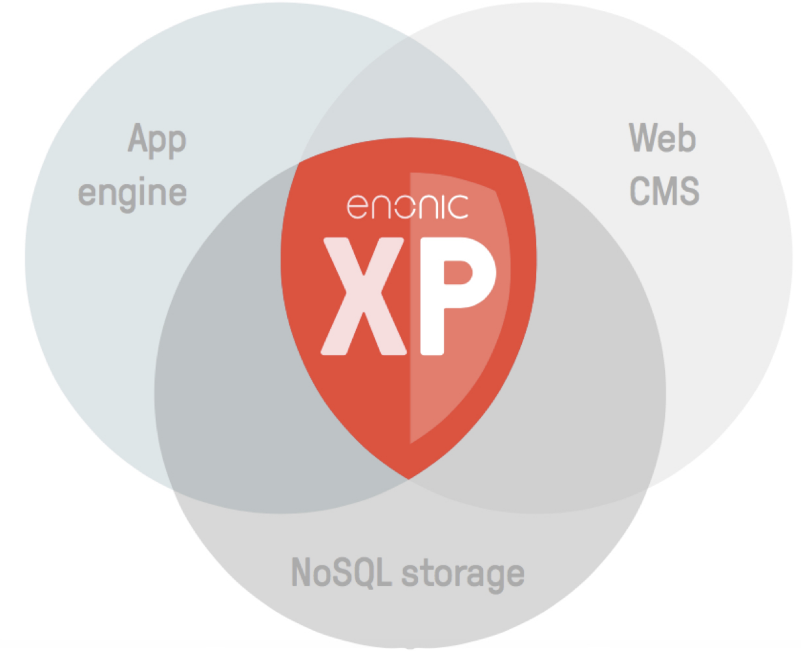
April 20, 2017

Appendix

Enonic XP is a unique and powerful application development stack - in a single runtime.

Lighting fast, and capable of scaling from a single server to large clusters - our mission is to make web development as predictable as building applications for traditional Operating System.

We simply call it the Web Operating System



- NoSQL storage - Distributed content repository built on top of Elasticsearch
- OSGi App Engine - Supports application development with serverside Javascript - using the popular PurpleJS framework
- Powerful embedded Web Content Management - seamlessly blend applications and websites
- Runs on the powerful Java Virtual Machine - can be deployed on just about any infrastructure.

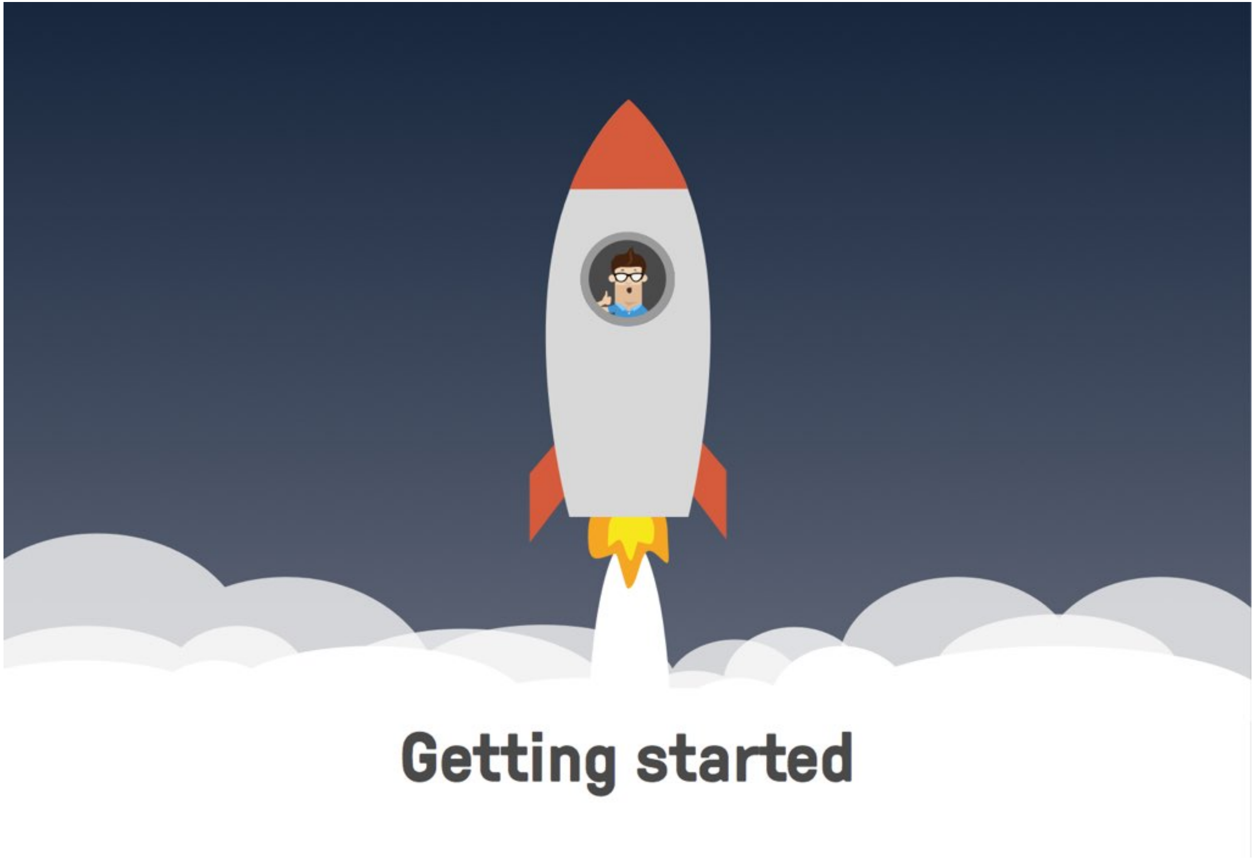
To get started - check out *Getting Started* or move on to the *Tutorials*. The more savvy will probably enjoy our *API and Reference Guide*.

Enjoy! - *The Enonic Development Team*

CHAPTER 1

Getting Started

So - you're looking for the fastest way to get Enonic XP up and running?



Select ONE of the options below to get going:

Enonic Cloud

Enonic Cloud is a one-stop-shop hosted version of Enonic XP, available on demand.

Complete the following steps:

- *Request Free Cloud Trial*
- *Log In*
- *Add Sample Apps*
- *Open Content Studio*
- *Visit Enonic Market*
- *Next Steps*

Request Free Cloud Trial

We're offering a time limited free trial - running your very own cloud instance of Enonic XP. All you need to do is:

- Request a Free Cloud Trial of Enonic XP

Note: Enonic also offers paid cloud instances, ranging from "Developer Cloud" to "Platium Cloud". Check out our offerings

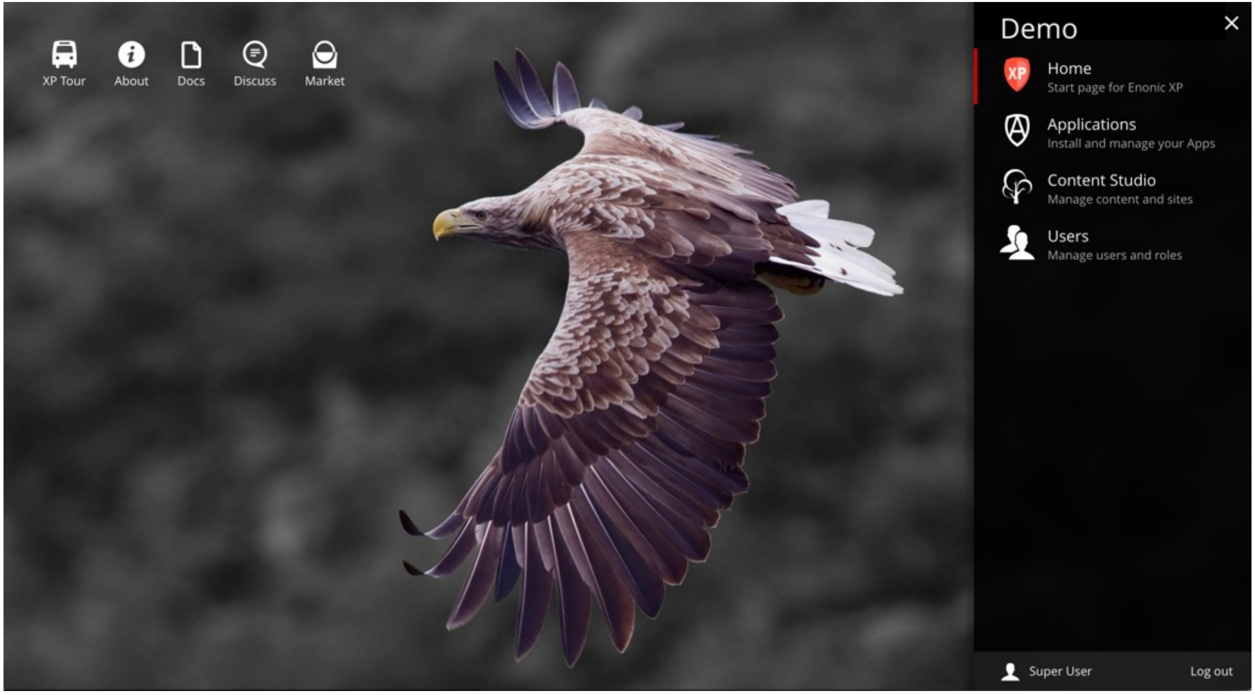
Log In

After requesting a Trial, you should recieve an informative e-mail about your installation.

- Click the link in your e-mail to reach the administrative interface, it should be in the following form: `http://<my-email-com>.tryme.enonic.io`

Log in with username `su` and password `password`.

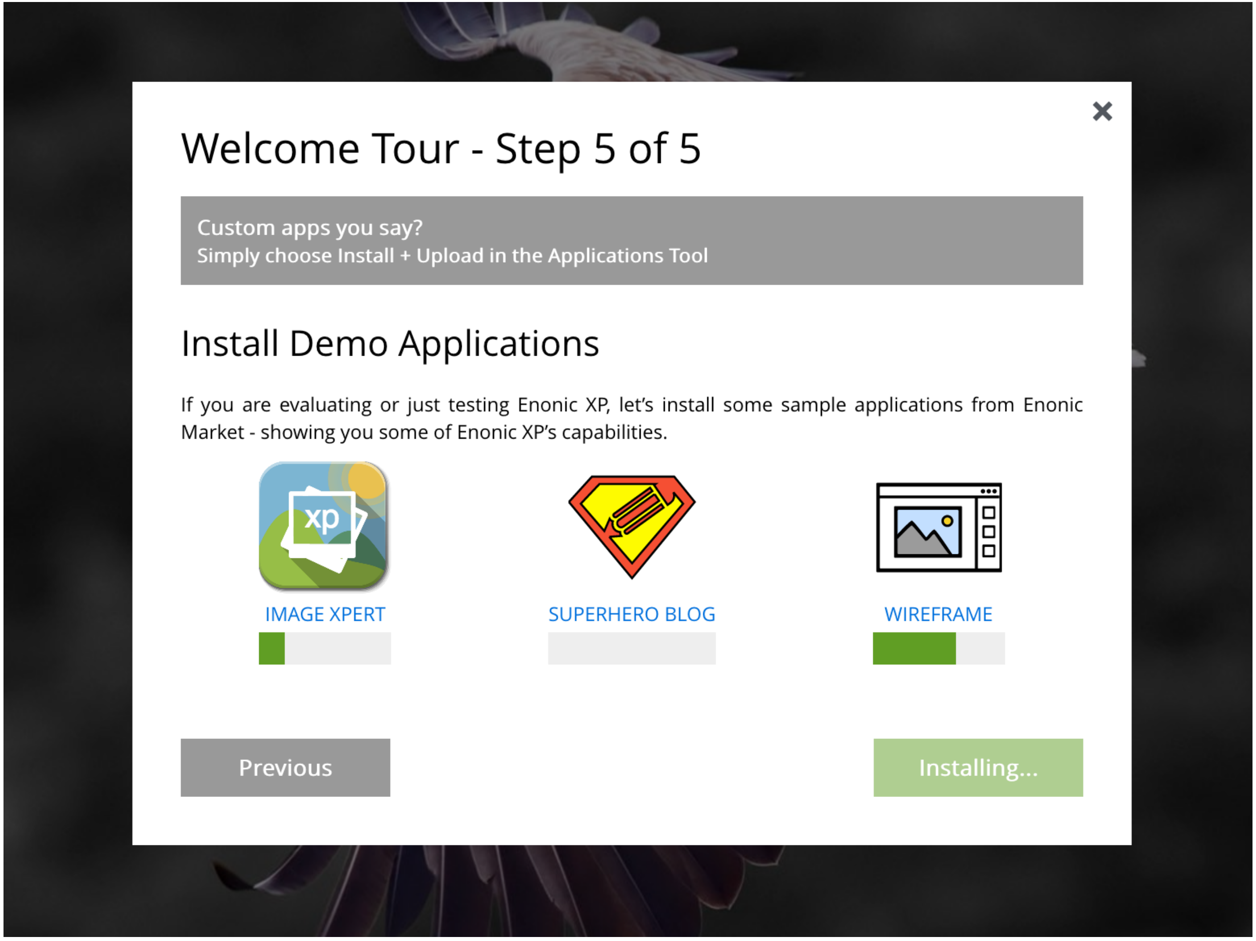
After logging in you should see the following screen, with the launcher panel to the right:



Add Sample Apps

If this is the first time you launch XP - the welcome tour will automatically launch. If it does not start, simply click the tour icon at top left of the home screen.

- Click through each step of the welcome tour
- On the last step, click the *Install* button
- Once the applications are installed, click finish



Welcome Tour - Step 5 of 5

Custom apps you say?
Simply choose Install + Upload in the Applications Tool

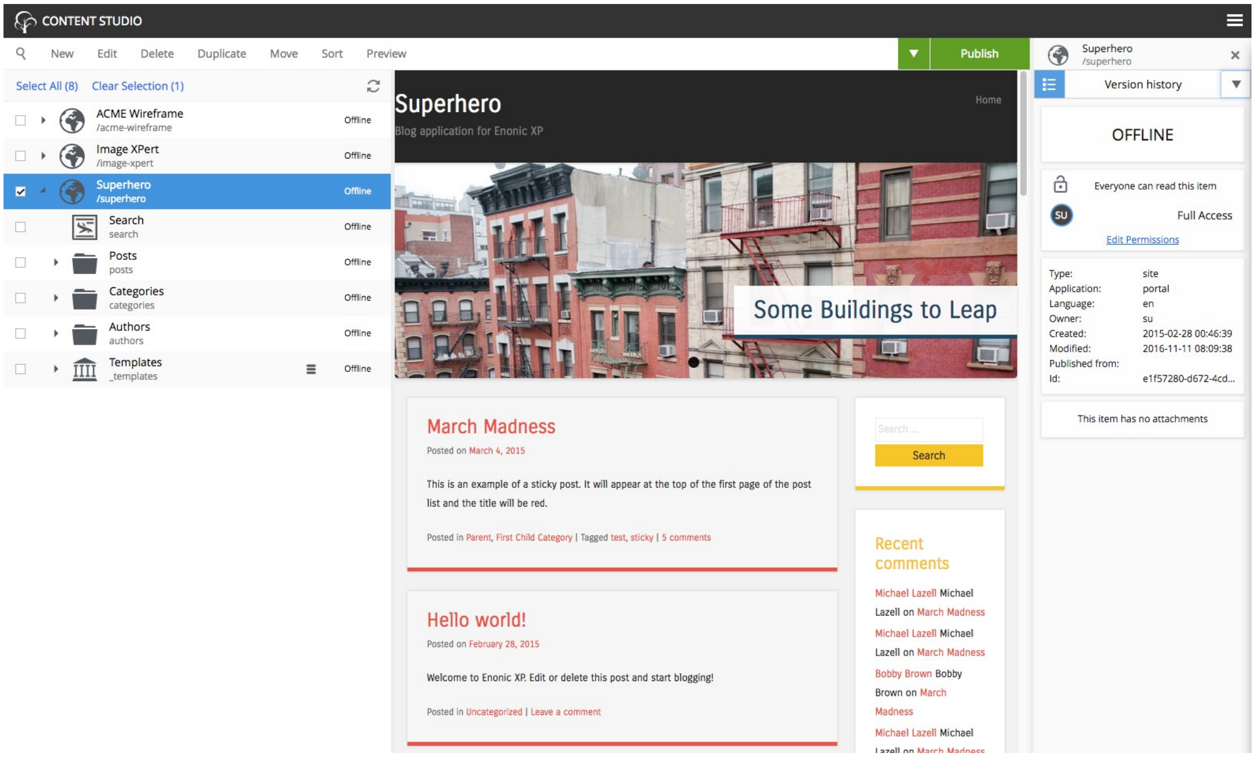
Install Demo Applications

If you are evaluating or just testing Enonic XP, let's install some sample applications from Enonic Market - showing you some of Enonic XP's capabilities.

Open Content Studio

The sample applications automatically create demo sites you can try out.

- Select "Content Studio" from the launcher menu
- Once loaded, you will find the sites in the tree grid
- Select or expand the sites you are interested in for a preview
- Right click or choose actions from the menu to get going



☐	ACME Wireframe /acme-wireframe	Offline
☐	Image XPert /image-xpert	Offline
☑	Superhero /superhero	Offline
☐	Search search	Offline
☐	Posts posts	Offline
☐	Categories categories	Offline
☐	Authors authors	Offline
☐	Templates _templates	Offline

Superhero

Blog application for Enonic XP

Some Buildings to Leap

March Madness

Posted on March 4, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Hello world!

Posted on February 28, 2015

Welcome to Enonic XP. Edit or delete this post and start blogging!

Posted in Uncategorized | Leave a comment

Michael Lazell Michael Lazell on March Madness

Michael Lazell Michael Lazell on March Madness

Bobby Brown Bobby Brown on March Madness

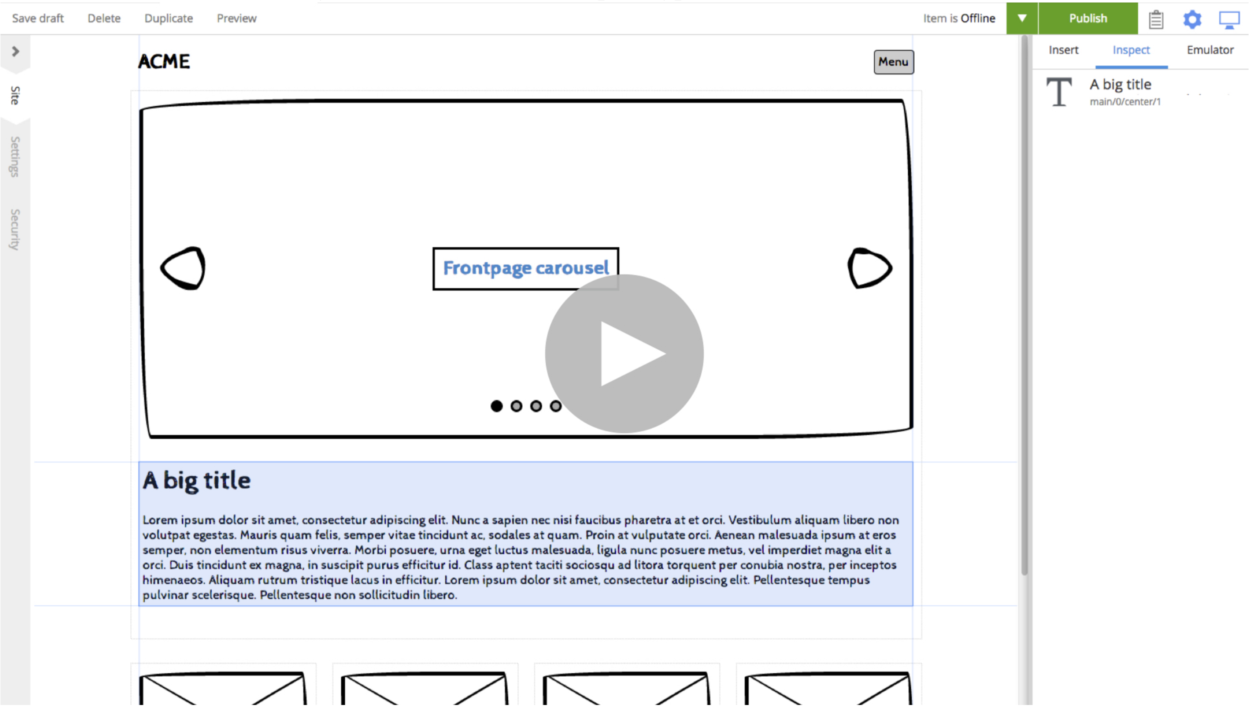
Michael Lazell Michael Lazell on March Madness

Type:	site
Application:	portal
Language:	en
Owner:	su
Created:	2015-02-28 00:46:39
Modified:	2016-11-11 08:09:38
Published from:	
Id:	e1f57280-d672-4cd...

This item has no attachments

Wireframe Prototyping Application

Watch this video to see how you can make interactive prototypes with Enonic XP and learn about Content Studio:



A big title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc a sapien nec nisi faucibus pharetra at et orci. Vestibulum aliquam libero non volutpat egestas. Mauris quam felis, semper vitae tincidunt ac, sodales at quam. Proin at vulputate orci. Aenean malesuada ipsum at eros semper, non elementum risus viverra. Morbi posuere, urna eget luctus malesuada, ligula nunc posuere metus, vel imperdiet magna elit a orci. Duis tincidunt ex magna, in suscipit purus efficitur id. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aliquam rutrum tristique lacus in efficitur. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque tempus pulvinar scelerisque. Pellentesque non sollicitudin libero.

Superhero Blog Application

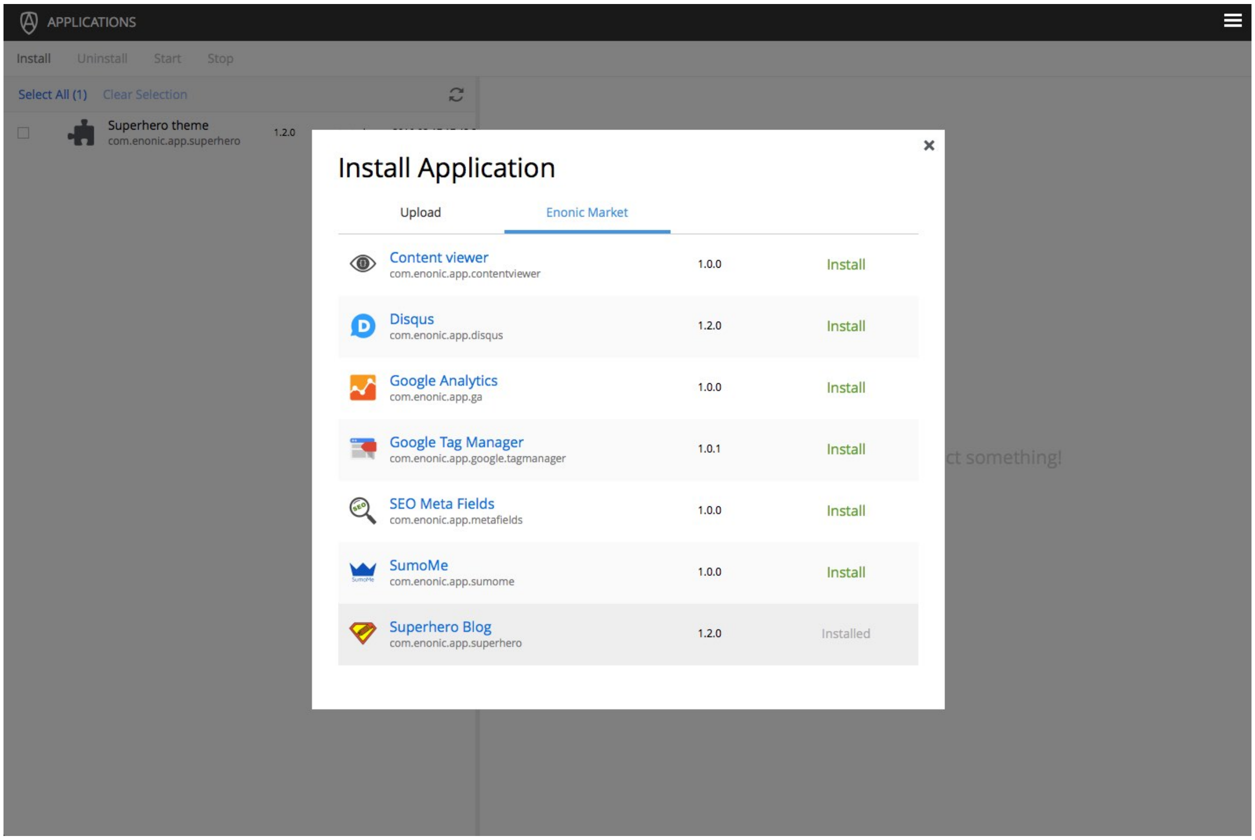
Watch this video to learn how you can use Enonic XP as a blogging platform, and learn more about Content Studio:

Visit Enonic Market

Find more on Enonic Market

If you want to try other applications, follow the steps below:

- Open the *Applications* tool from the launcher panel to the right
- Click *Install* from the menu (top left)
- Browse to find the applications you are looking for and click *Install*



Install Application

Content viewer com.enonic.app.contentviewer	1.0.0	Install
Disqus com.enonic.app.disqus	1.2.0	Install
Google Analytics com.enonic.app.ga	1.0.0	Install
Google Tag Manager com.enonic.app.google.tagmanager	1.0.1	Install
SEO Meta Fields com.enonic.app.metafields	1.0.0	Install
SumoMe com.enonic.app.sumome	1.0.0	Install
Superhero Blog com.enonic.app.superhero	1.2.0	Installed

You may also visit Enonic market directly on https://market.enonic.com

Next Steps

Congratulations on getting started :-)

If you're ready for some more fun, we recommend looking into the following:

- Watch the Platform Video
- Hook up with the community on our forum
- Build your first Application with *My First App*
- Learn more about *Sites*

OSX

This section describes the easiest way to install Enonic XP on a Mac OSX computer.

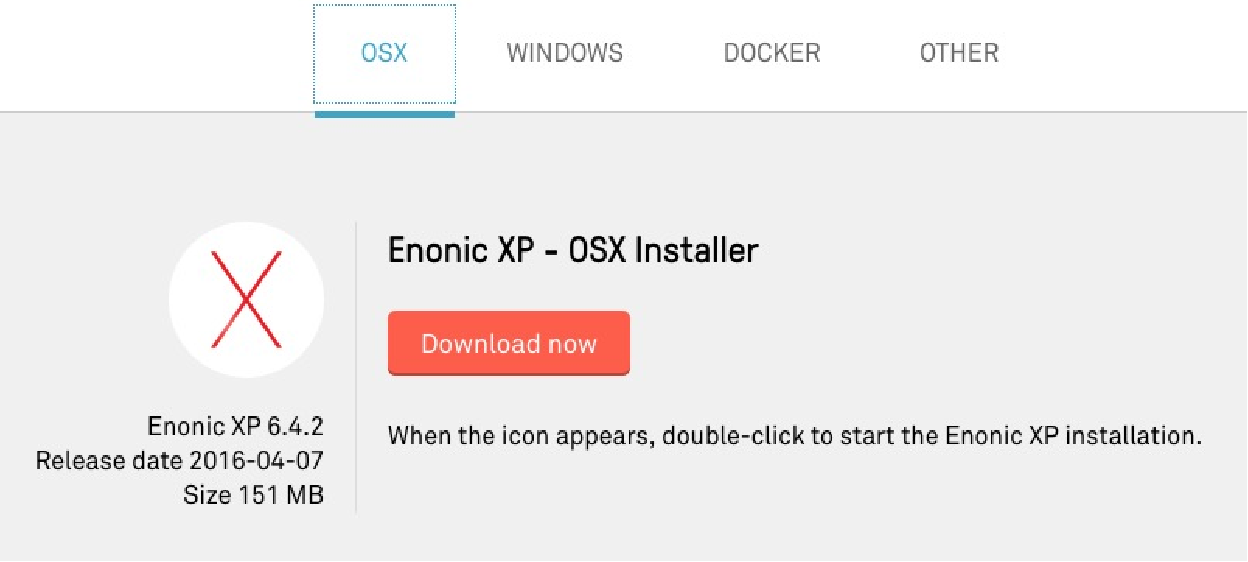
Watch this video

or complete the following steps:

- *Download*
- *Install*
- *Start*
- *Log In*
- *Add Sample Apps*
- *Open Content Studio*
- *Visit Enonic Market*
- *Next Steps*
- *Troubleshooting*

Download

Visit the Enonic XP download page and select the **OSX** tab. Click the **Download now** button and save the .dmg file to a convenient location.



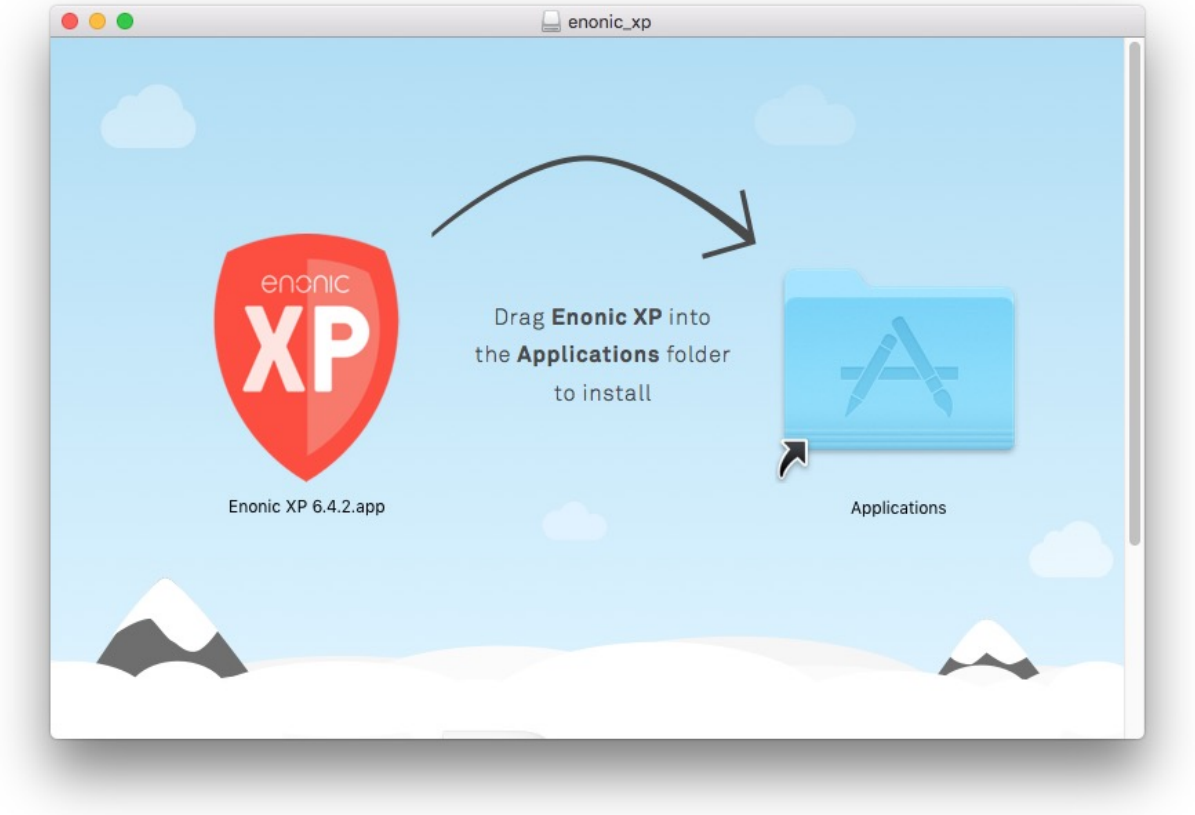
Enonic XP - OSX Installer

When the icon appears, double-click to start the Enonic XP installation.

Enonic XP 6.4.2
Release date 2016-04-07
Size 151 MB

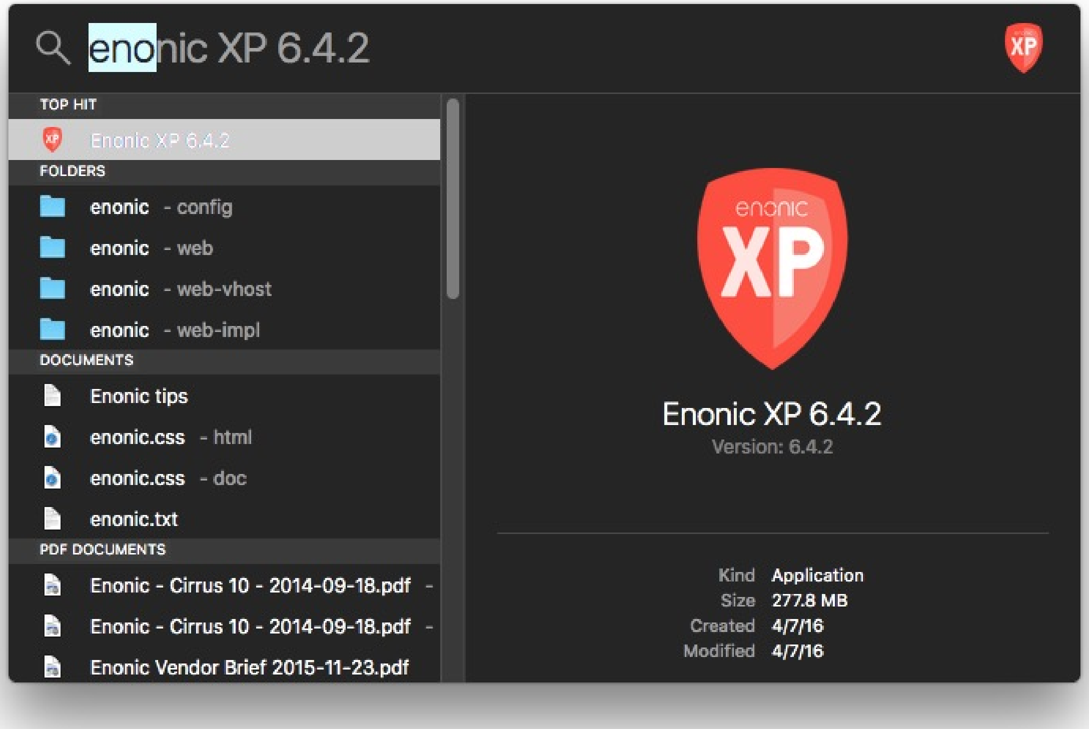
Install

Open the file when it is finished downloading. A window like the one in the image below will appear. Click and drag the Enonic XP logo to the Applications folder.



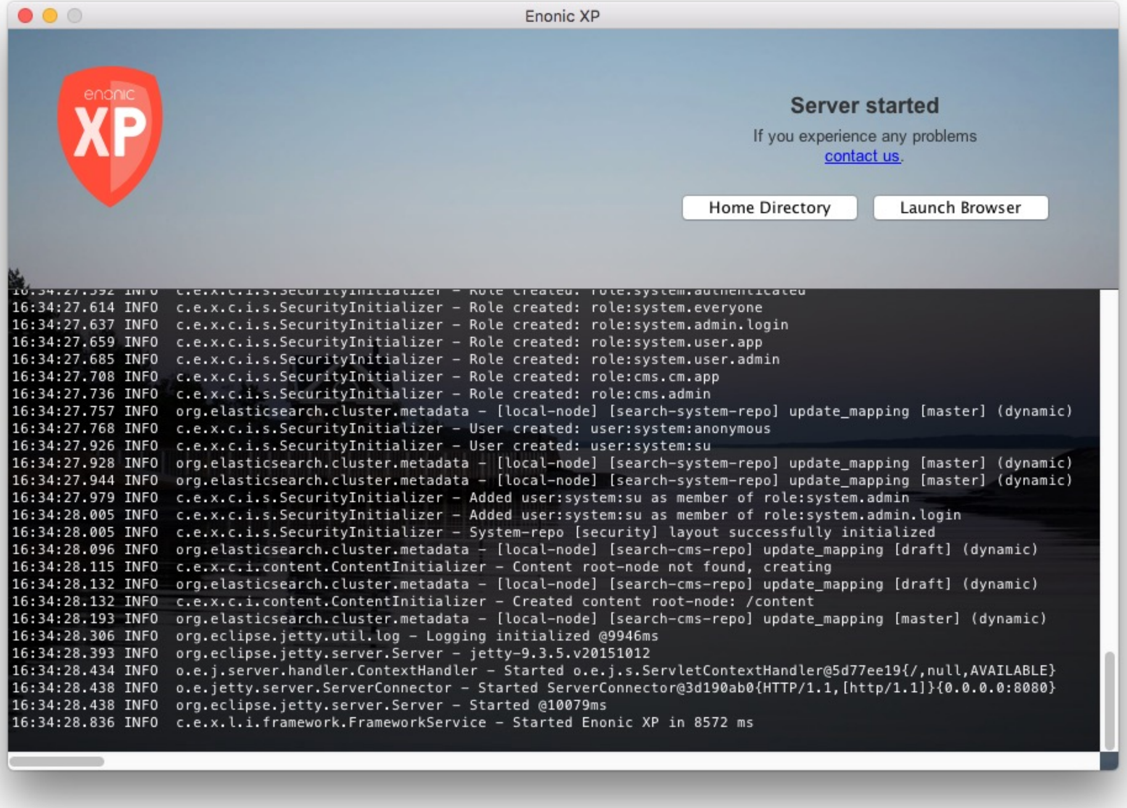
Start

Find the Enonic XP app in the applications folder, or use Spotlight search, and double click to open it.



Kind Application
Size 277.8 MB
Created 4/7/16
Modified 4/7/16

A notice may appear to inform you that the file was downloaded from the Internet. Go ahead and click "Open". Enonic XP will start and a window will open with the log. XP will continue to run while this window is open. The **Home Directory** button will open Finder to the XP home folder where the installation's files can be found. The **Launch Browser** button will open the Enonic XP administration interface in the default browser. The admin UI can also be reached at http://localhost:8080



Server started

If you experience any problems contact us.

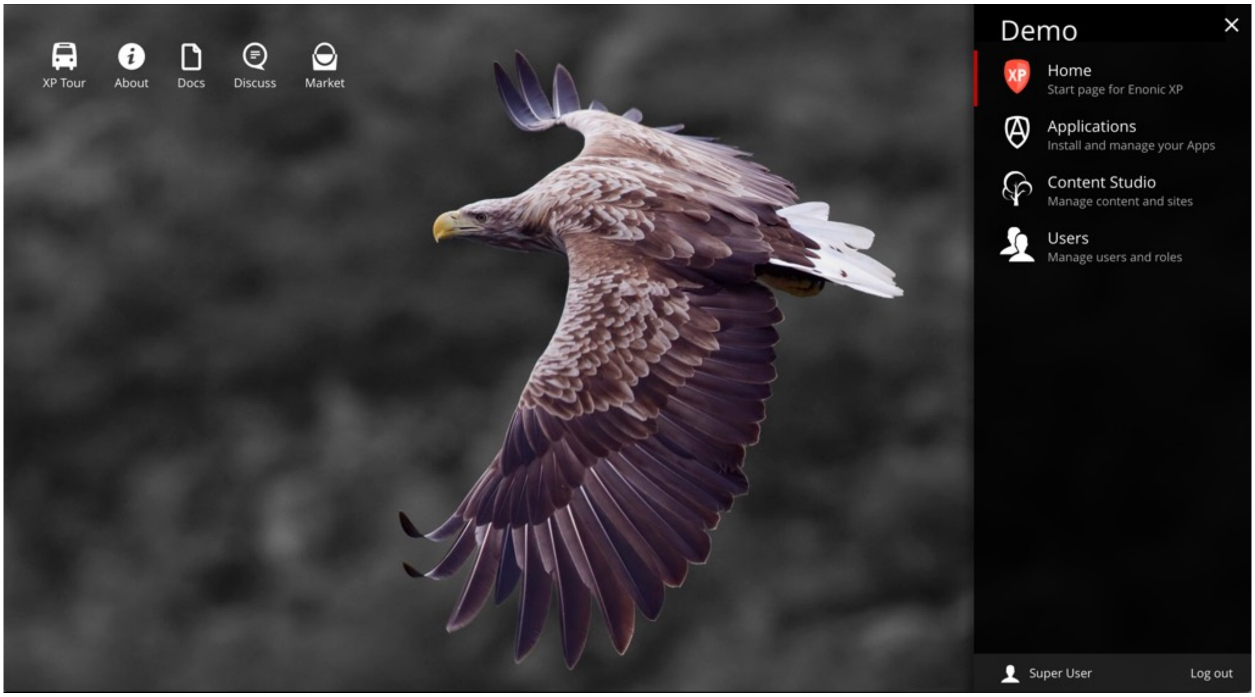
```
16:34:27.592 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:system.authenticated
16:34:27.614 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:system.everyone
16:34:27.637 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:system.admin.login
16:34:27.659 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:system.user.app
16:34:27.685 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:system.user.admin
16:34:27.708 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:cms.cm.app
16:34:27.736 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:cms.admin
16:34:27.757 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-system-repo] update_mapping [master] (dynamic)
16:34:27.768 INFO  c.e.x.c.i.s.SecurityInitializer - User created: user:system:anonymous
16:34:27.926 INFO  c.e.x.c.i.s.SecurityInitializer - User created: user:system:su
16:34:27.928 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-system-repo] update_mapping [master] (dynamic)
16:34:27.944 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-system-repo] update_mapping [master] (dynamic)
16:34:27.979 INFO  c.e.x.c.i.s.SecurityInitializer - Added user:system:su as member of role:system.admin
16:34:28.005 INFO  c.e.x.c.i.s.SecurityInitializer - Added user:system:su as member of role:system.admin.login
16:34:28.005 INFO  c.e.x.c.i.s.SecurityInitializer - System-repo [security] layout successfully initialized
16:34:28.096 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-cms-repo] update_mapping [draft] (dynamic)
16:34:28.115 INFO  c.e.x.c.i.content.ContentInitializer - Content root-node not found, creating
16:34:28.132 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-cms-repo] update_mapping [draft] (dynamic)
16:34:28.132 INFO  c.e.x.c.i.content.ContentInitializer - Created content root-node: /content
16:34:28.193 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-cms-repo] update_mapping [master] (dynamic)
16:34:28.306 INFO  org.eclipse.jetty.util.log - Logging initialized @9946ms
16:34:28.393 INFO  org.eclipse.jetty.server.Server - jetty-9.3.5.v20151012
16:34:28.434 INFO  o.e.j.server.handler.ContextHandler - Started o.e.j.s.ServletContextHandler@5d77ee19{/,null,AVAILABLE}
16:34:28.438 INFO  o.e.jetty.server.ServerConnector - Started ServerConnector@3d190ab0{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
16:34:28.438 INFO  org.eclipse.jetty.server.Server - Started @10079ms
16:34:28.836 INFO  c.e.x.l.i.framework.FrameworkService - Started Enonic XP in 8572 ms
```

Log In

Click the **Launch Browser** button or point your browser to `http://localhost:8080`

Log in with username `su` and password `password`.

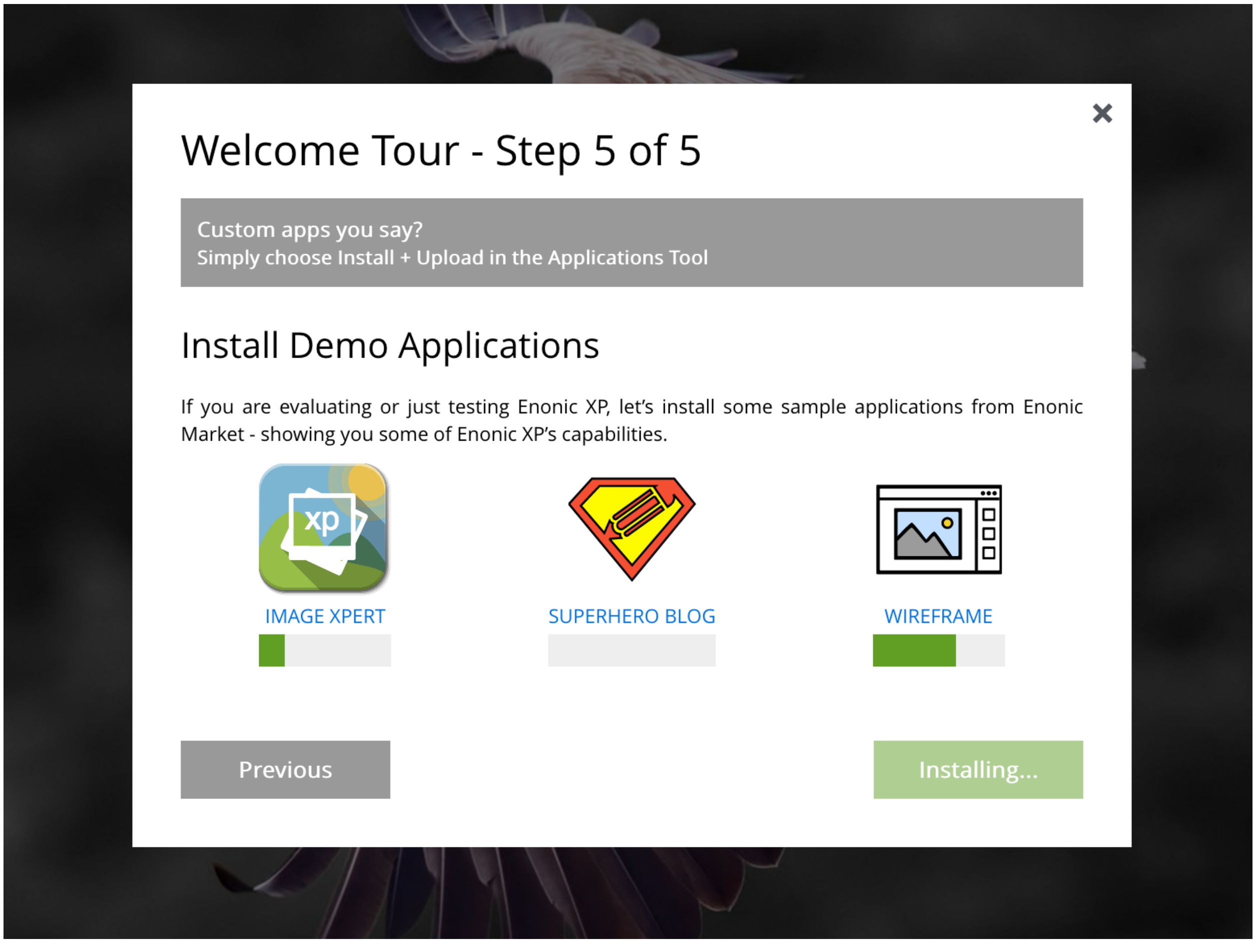
After logging in you should see the following screen, with the launcher panel to the right:



Add Sample Apps

If this is the first time you launch XP - the welcome tour will automatically launch. If it does not start, simply click the tour icon at top left of the home screen.

- Click through each step of the welcome tour
- On the last step, click the *Install* button
- Once the applications are installed, click finish



Welcome Tour - Step 5 of 5

Custom apps you say?
Simply choose Install + Upload in the Applications Tool

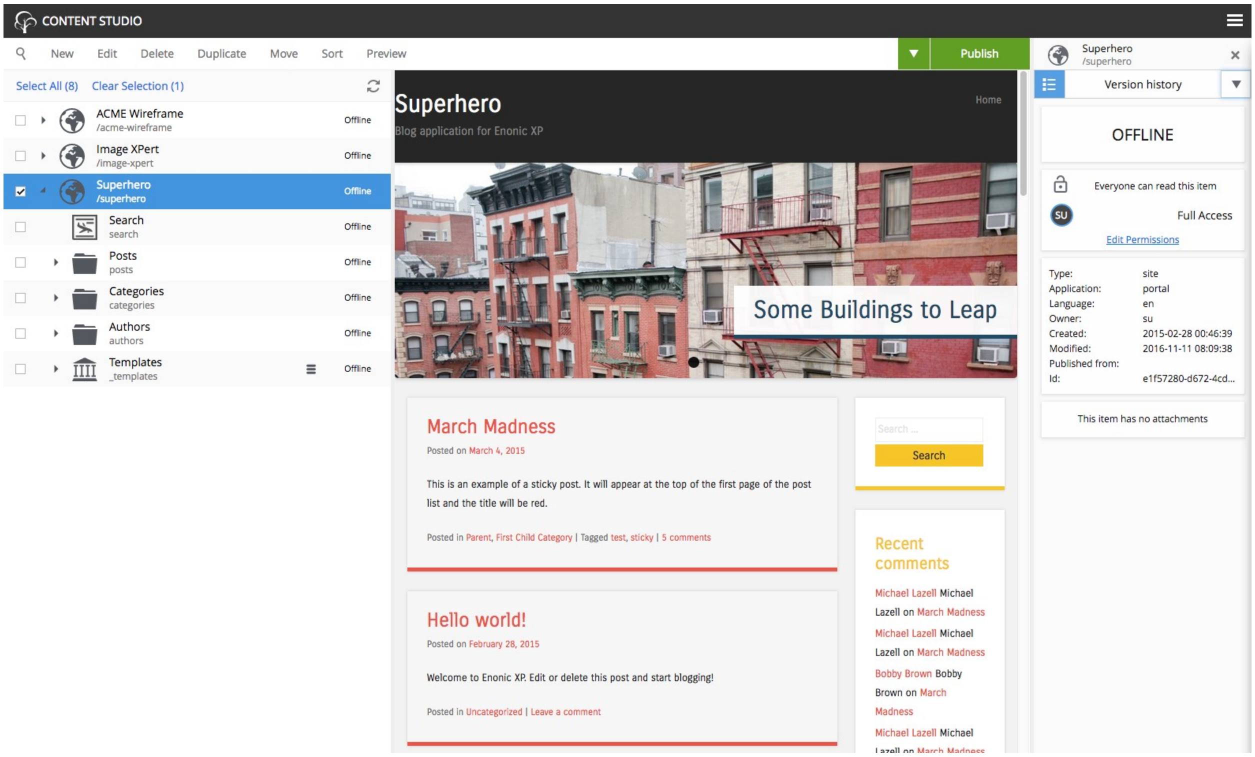
Install Demo Applications

If you are evaluating or just testing Enonic XP, let's install some sample applications from Enonic Market - showing you some of Enonic XP's capabilities.

Open Content Studio

The sample applications automatically create demo sites you can try out.

- Select "Content Studio" from the launcher menu
- Once loaded, you will find the sites in the tree grid
- Select or expand the sites you are interested in for a preview
- Right click or choose actions from the menu to get going



ACME Wireframe	/acme-wireframe	Offline
Image XPert	/image-xpert	Offline
Superhero	/superhero	Offline
Search	search	Offline
Posts	posts	Offline
Categories	categories	Offline
Authors	authors	Offline
Templates	_templates	Offline

Superhero

Blog application for Enonic XP

Some Buildings to Leap

March Madness

Posted on March 4, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Hello world!

Posted on February 28, 2015

Welcome to Enonic XP. Edit or delete this post and start blogging!

Posted in Uncategorized | Leave a comment

Michael Lazell Michael Lazell on March Madness

Michael Lazell Michael Lazell on March Madness

Bobby Brown Bobby Brown on March Madness

Michael Lazell Michael Lazell on March Madness

Superhero
/superhero

OFFLINE

Everyone can read this item

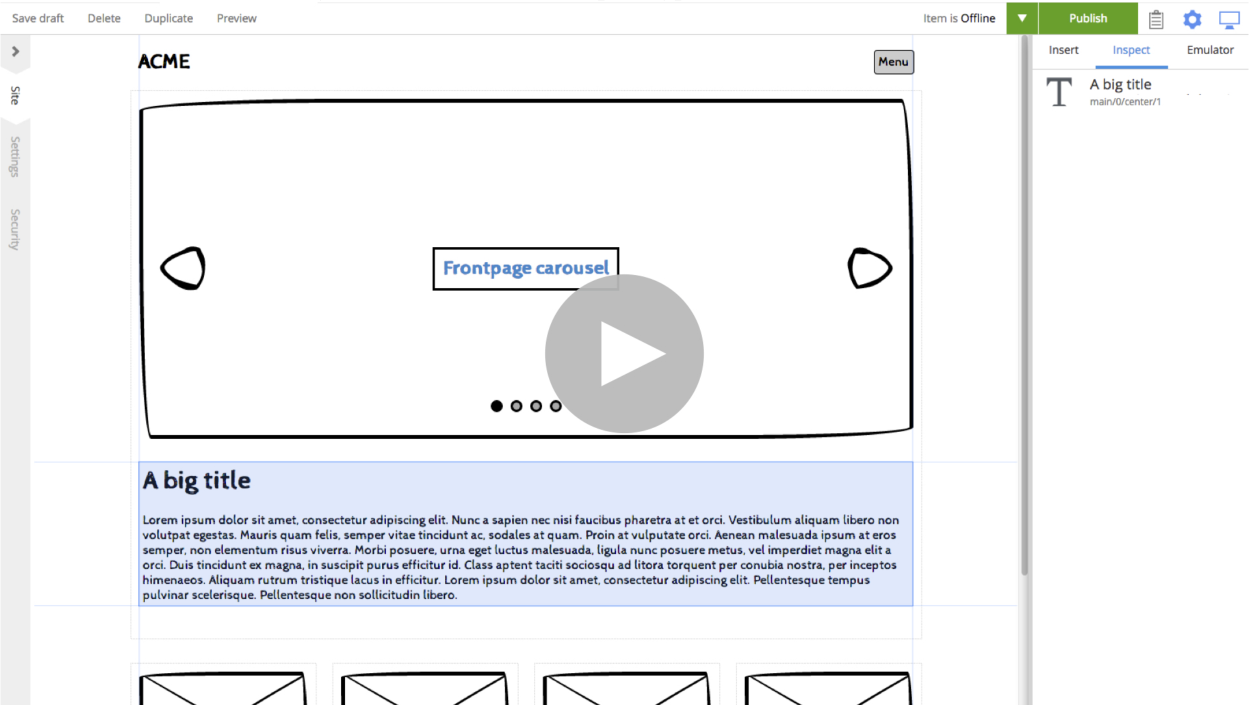
SU Full Access

Type:	site
Application:	portal
Language:	en
Owner:	su
Created:	2015-02-28 00:46:39
Modified:	2016-11-11 08:09:38
Published from:	
Id:	e1f57280-d672-4cd...

This item has no attachments

Wireframe Prototyping Application

Watch this video to see how you can make interactive prototypes with Enonic XP and learn about Content Studio:



A big title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc a sapien nec nisi faucibus pharetra at et orci. Vestibulum aliquam libero non volutpat egestas. Mauris quam felis, semper vitae tincidunt ac, sodales at quam. Proin at vulputate orci. Aenean malesuada ipsum at eros semper, non elementum risus viverra. Morbi posuere, urna eget luctus malesuada, ligula nunc posuere metus, vel imperdiet magna elit a orci. Duis tincidunt ex magna, in suscipit purus efficitur id. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aliquam rutrum tristique lacus in efficitur. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque tempus pulvinar scelerisque. Pellentesque non sollicitudin libero.

Superhero Blog Application

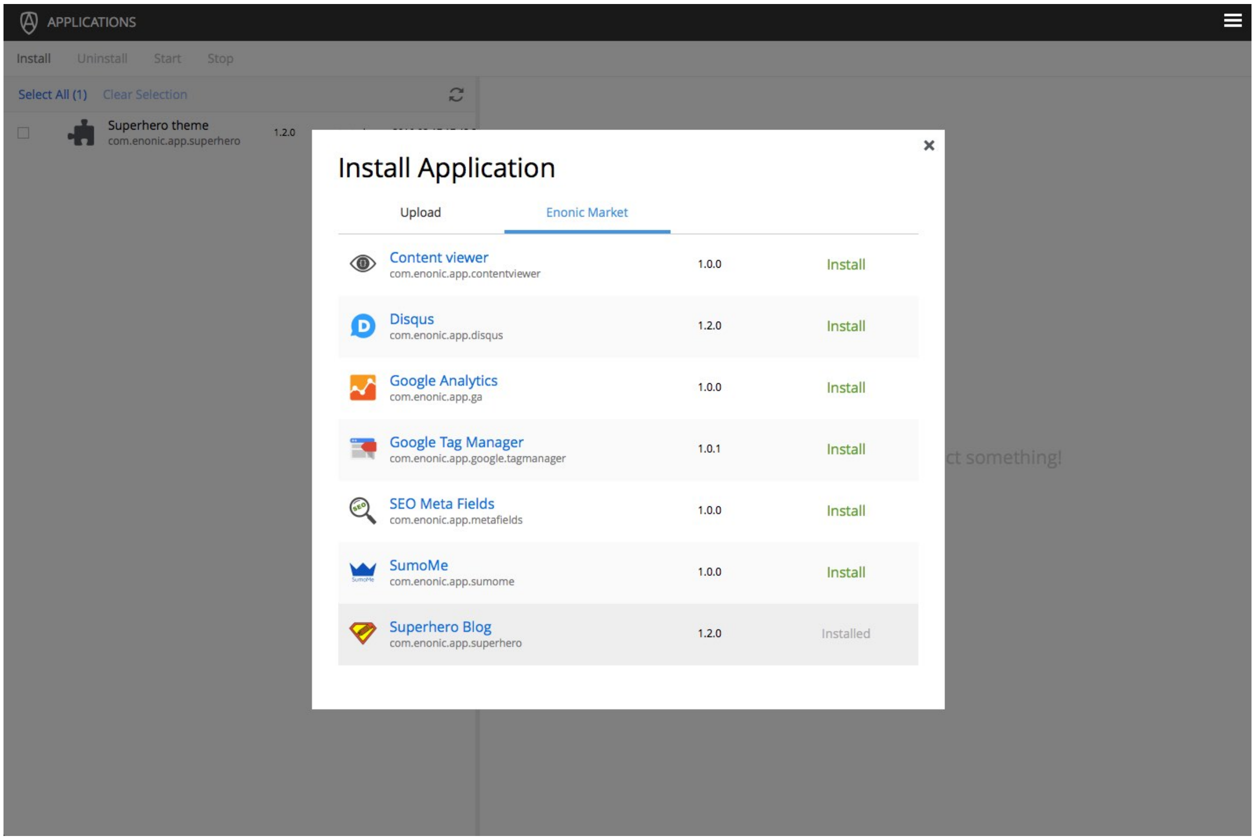
Watch this video to learn how you can use Enonic XP as a blogging platform, and learn more about Content Studio:

Visit Enonic Market

Find more on Enonic Market

If you want to try other applications, follow the steps below:

- Open the *Applications* tool from the launcher panel to the right
- Click *Install* from the menu (top left)
- Browse to find the applications you are looking for and click *Install*



Install Application

Content viewer com.enonic.app.contentviewer	1.0.0	Install
Disqus com.enonic.app.disqus	1.2.0	Install
Google Analytics com.enonic.app.ga	1.0.0	Install
Google Tag Manager com.enonic.app.google.tagmanager	1.0.1	Install
SEO Meta Fields com.enonic.app.metafields	1.0.0	Install
SumoMe com.enonic.app.sumome	1.0.0	Install
Superhero Blog com.enonic.app.superhero	1.2.0	Installed

You may also visit Enonic market directly on https://market.enonic.com

Next Steps

Congratulations on getting started :-)

If you're ready for some more fun, we recommend looking into the following:

- Watch the Platform Video
- Hook up with the community on our forum
- Build your first Application with *My First App*
- Learn more about *Sites*

Troubleshooting

Verify that you comply with the minimum requirements for running XP on your local machine

Note: General Requirements

- MacOS 10.9 or newer
- At least 1 GB of available memory
- HTTP port 8080 should be available (this can be changed if needed, see *Configuration*)

Windows

This section describes the easiest way to install Enonic XP on a Windows computer.

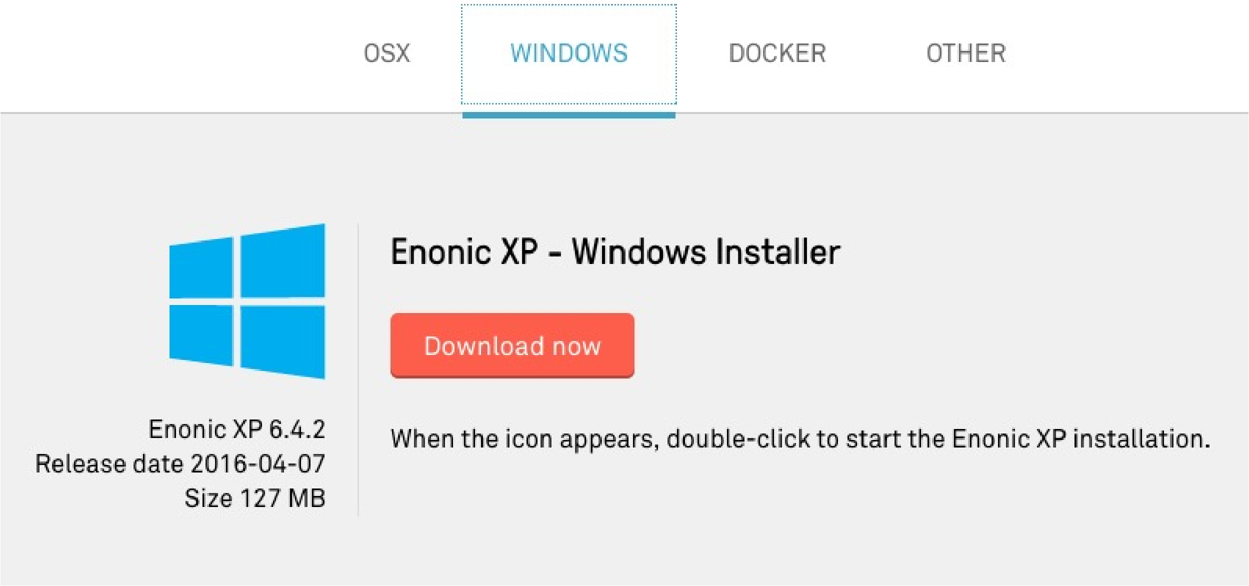
Watch this video

or complete the steps below:

- *Download*
- *Install*
- *Start*
- *Log In*
- *Add Sample Apps*
- *Open Content Studio*
- *Visit Enonic Market*
- *Next Steps*
- *Troubleshooting*

Download

Visit the Enonic XP download page and select the **WINDOWS** tab. Click the **Download now** button and save the .exe file to a convenient location.



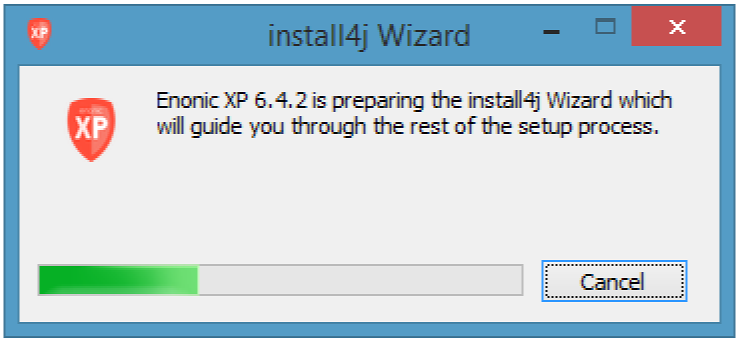
Enonic XP - Windows Installer

Enonic XP 6.4.2
Release date 2016-04-07
Size 127 MB

When the icon appears, double-click to start the Enonic XP installation.

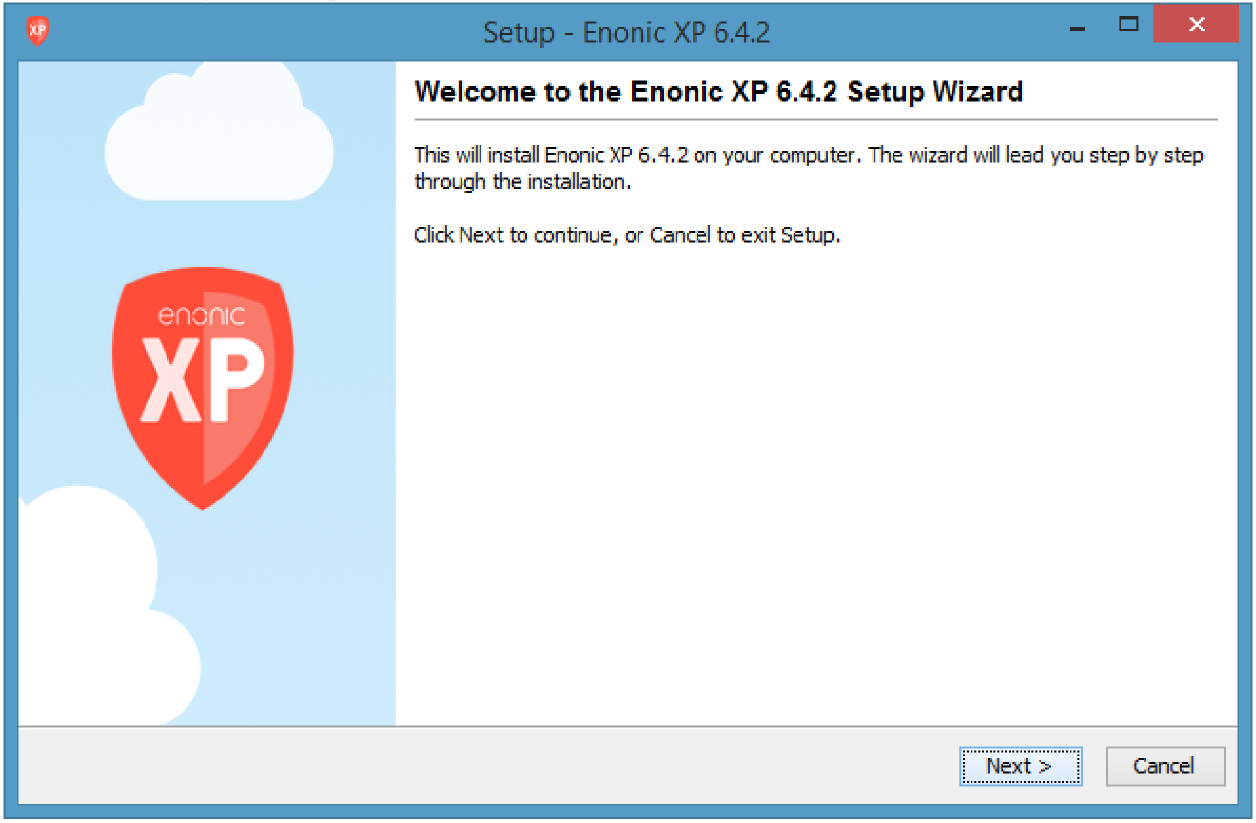
Install

Open the file when it is finished downloading and follow the instructions to install Enonic XP.



Enonic XP 6.4.2 is preparing the install4j Wizard which will guide you through the rest of the setup process.

Click "Next" to begin the Setup Wizard.

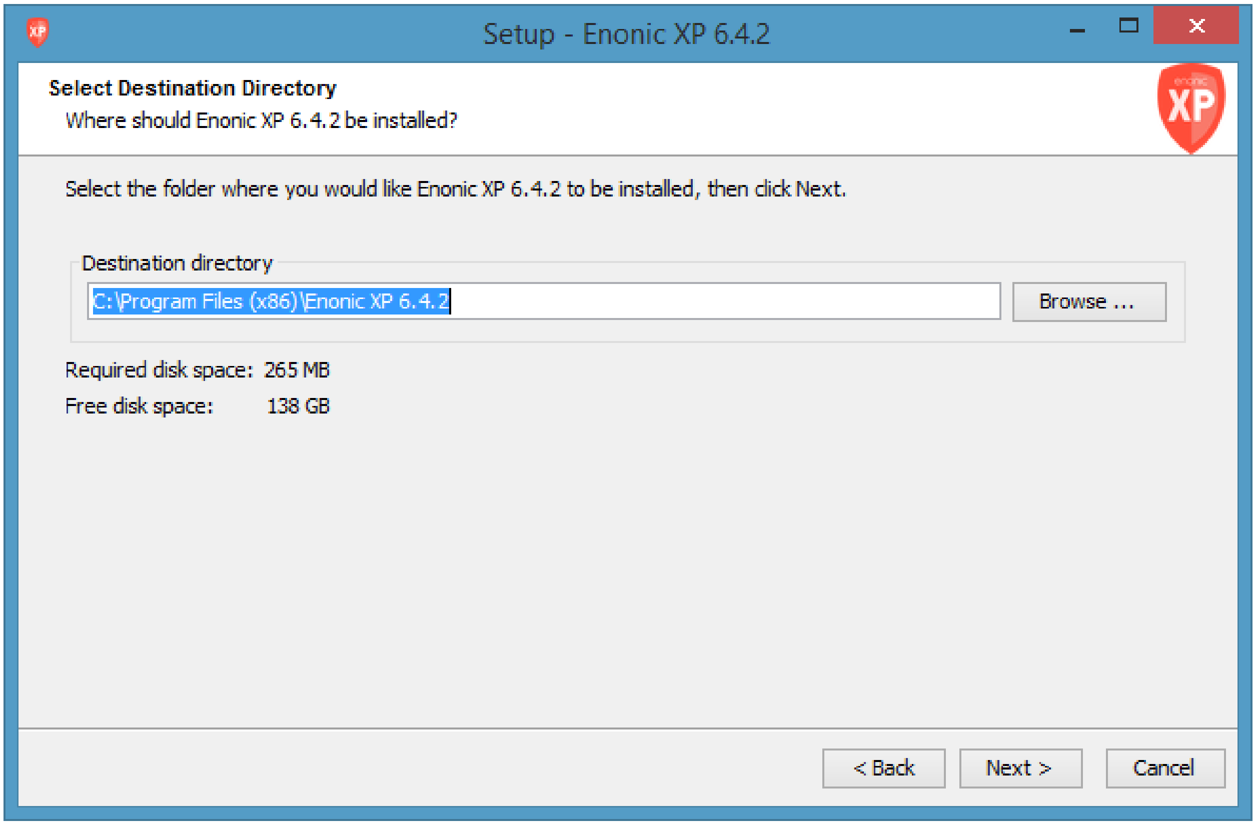


Welcome to the Enonic XP 6.4.2 Setup Wizard

This will install Enonic XP 6.4.2 on your computer. The wizard will lead you step by step through the installation.

Click Next to continue, or Cancel to exit Setup.

Select a convenient location to install Enonic XP. The default location is in `C:\Program Files (x86)\`



Setup - Enonic XP 6.4.2

Select Destination Directory

Where should Enonic XP 6.4.2 be installed?

Select the folder where you would like Enonic XP 6.4.2 to be installed, then click Next.

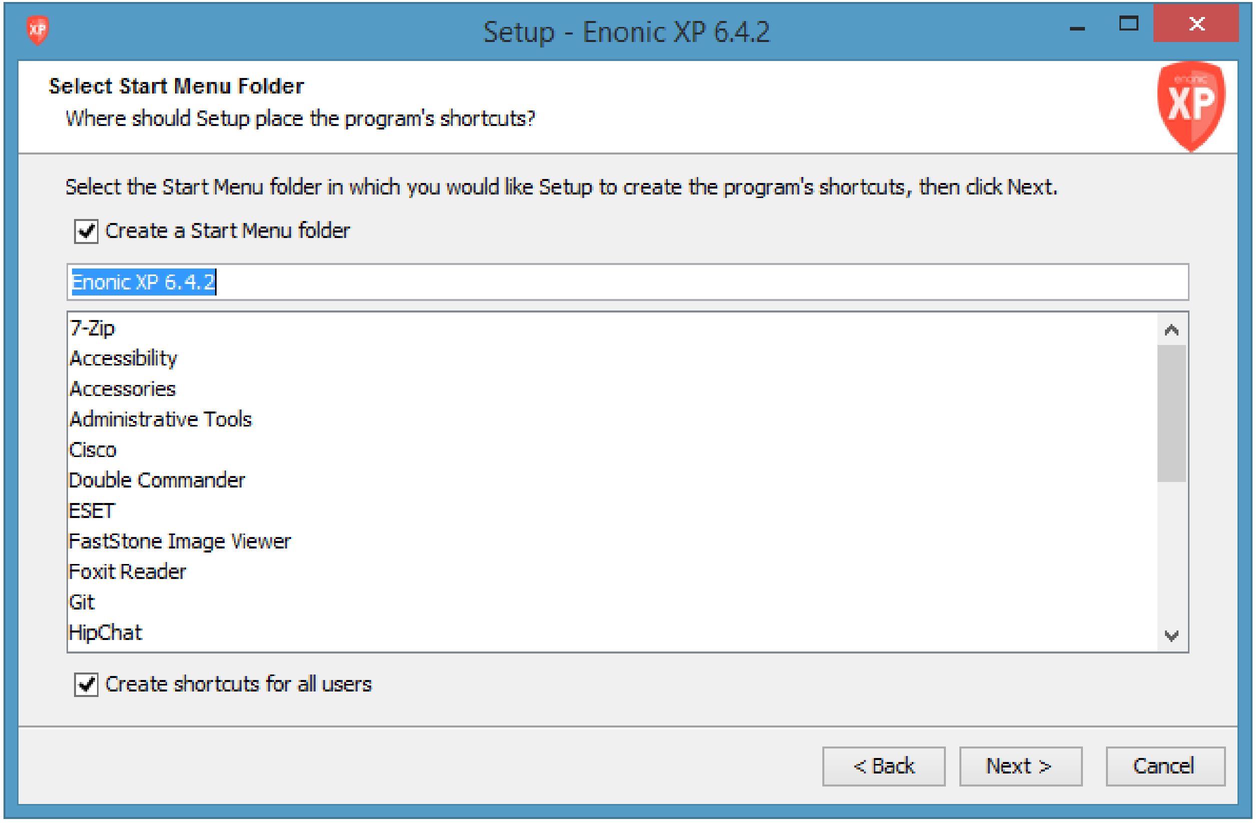
Destination directory

C:\Program Files (x86)\Enonic XP 6.4.2

Required disk space: 265 MB

Free disk space: 138 GB

Make your Start Menu folder and shortcuts selections. Leave the boxes checked for the defaults.



Setup - Enonic XP 6.4.2

Select Start Menu Folder

Where should Setup place the program's shortcuts?

Select the Start Menu folder in which you would like Setup to create the program's shortcuts, then click Next.

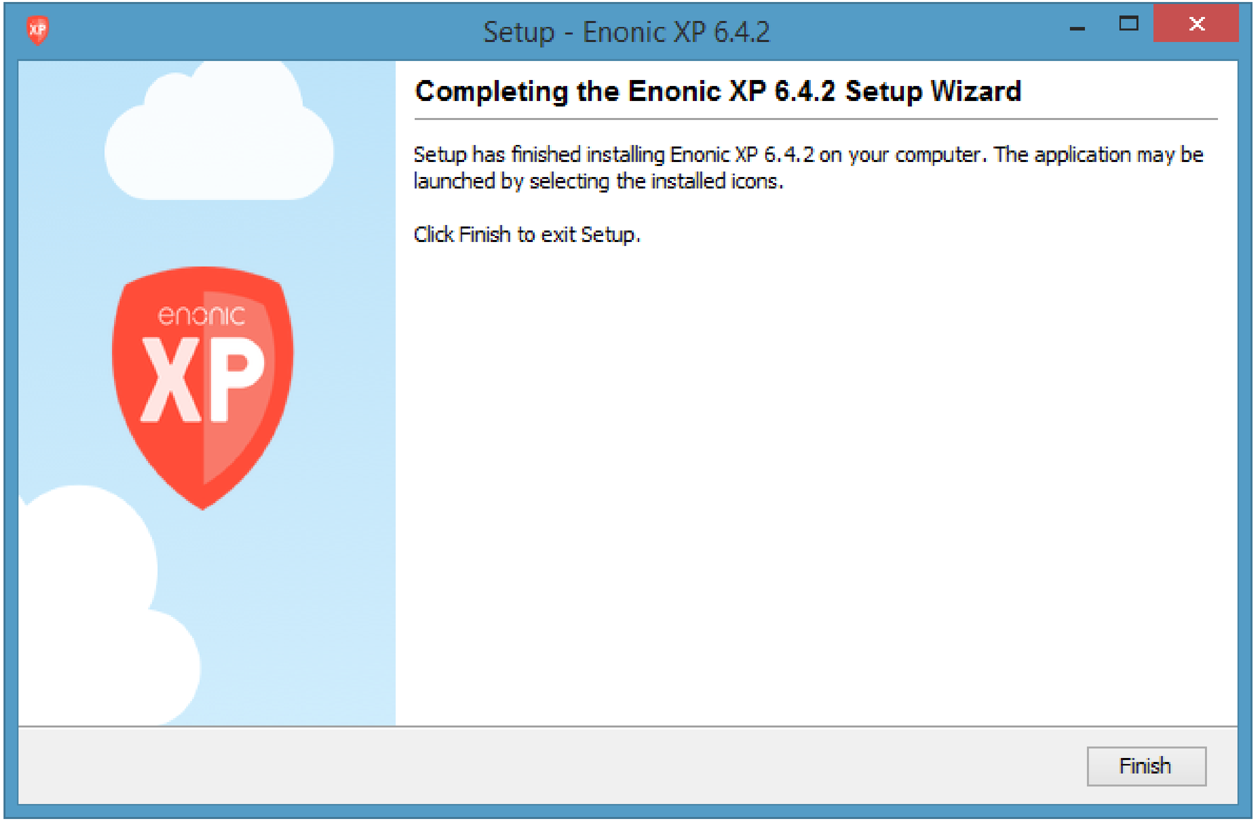
☑ Create a Start Menu folder

Enonic XP 6.4.2

7-Zip
Accessibility
Accessories
Administrative Tools
Cisco
Double Commander
ESET
FastStone Image Viewer
Foxit Reader
Git
HipChat

☑ Create shortcuts for all users

Click “Finish” to close the installer.



Completing the Enonic XP 6.4.2 Setup Wizard

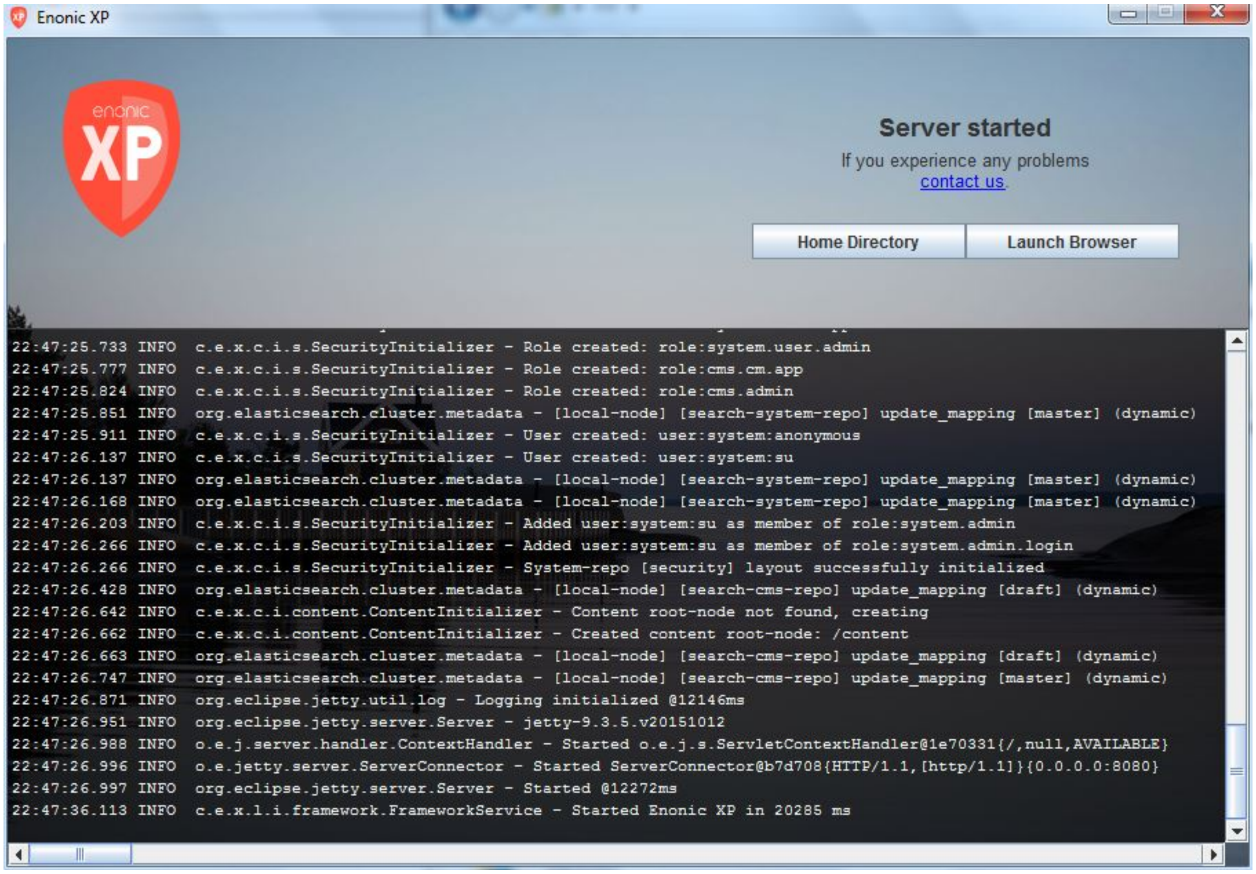
Setup has finished installing Enonic XP 6.4.2 on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

Start

Now find the Enonic XP Start Menu shortcut and click to start it.

A window will open with the log and some buttons. XP will continue to run while this window is open. The **Home Directory** button will open the XP home folder where the installation’s files can be found. The **Launch Browser** button will open the Enonic XP administration interface in the default browser. The admin UI can also be reached at http://localhost:8080



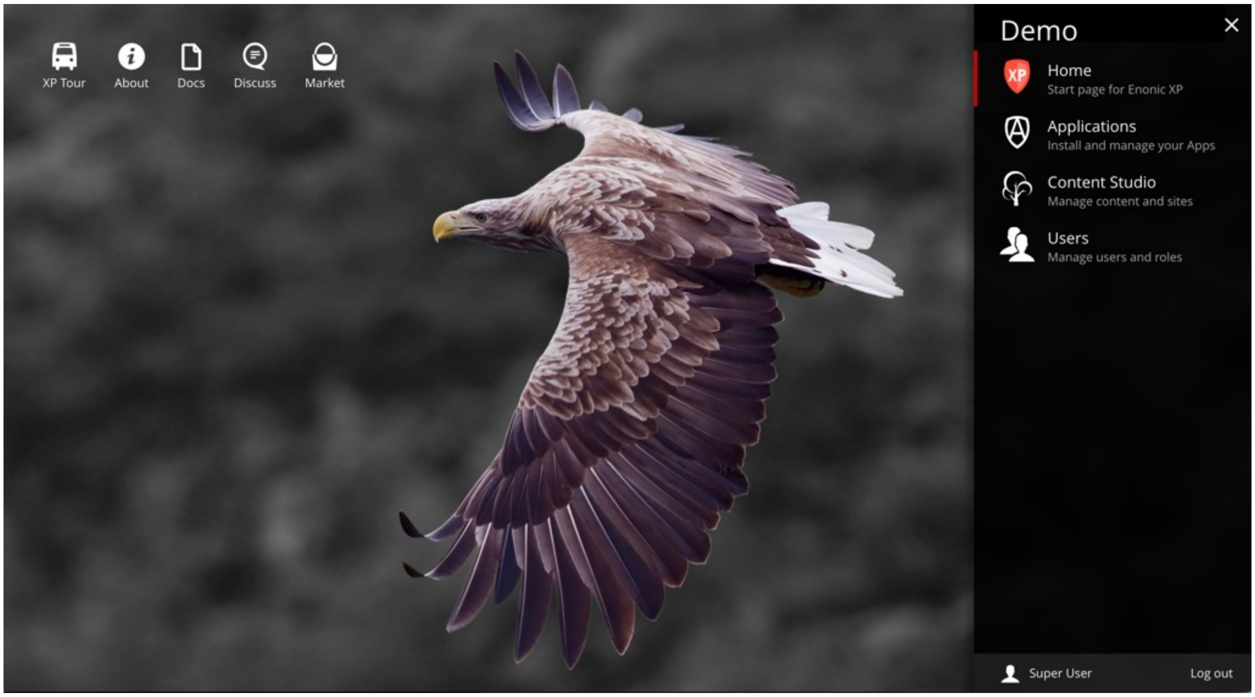
```
22:47:25.733 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:system.user.admin
22:47:25.777 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:cms.cm.app
22:47:25.824 INFO  c.e.x.c.i.s.SecurityInitializer - Role created: role:cms.admin
22:47:25.851 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-system-repo] update_mapping [master] (dynamic)
22:47:25.911 INFO  c.e.x.c.i.s.SecurityInitializer - User created: user:system:anonymous
22:47:26.137 INFO  c.e.x.c.i.s.SecurityInitializer - User created: user:system:su
22:47:26.137 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-system-repo] update_mapping [master] (dynamic)
22:47:26.168 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-system-repo] update_mapping [master] (dynamic)
22:47:26.203 INFO  c.e.x.c.i.s.SecurityInitializer - Added user:system:su as member of role:system.admin
22:47:26.266 INFO  c.e.x.c.i.s.SecurityInitializer - Added user:system:su as member of role:system.admin.login
22:47:26.266 INFO  c.e.x.c.i.s.SecurityInitializer - System-repo [security] layout successfully initialized
22:47:26.428 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-cms-repo] update_mapping [draft] (dynamic)
22:47:26.642 INFO  c.e.x.c.i.content.ContentInitializer - Content root-node not found, creating
22:47:26.662 INFO  c.e.x.c.i.content.ContentInitializer - Created content root-node: /content
22:47:26.663 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-cms-repo] update_mapping [draft] (dynamic)
22:47:26.747 INFO  org.elasticsearch.cluster.metadata - [local-node] [search-cms-repo] update_mapping [master] (dynamic)
22:47:26.871 INFO  org.eclipse.jetty.util.log - Logging initialized @12146ms
22:47:26.951 INFO  org.eclipse.jetty.server.Server - jetty-9.3.5.v20151012
22:47:26.988 INFO  o.e.j.server.handler.ContextHandler - Started o.e.j.s.ServletContextHandler@1e70331{/,null,AVAILABLE}
22:47:26.996 INFO  o.e.jetty.server.ServerConnector - Started ServerConnector@b7d708{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
22:47:26.997 INFO  org.eclipse.jetty.server.Server - Started @12272ms
22:47:36.113 INFO  c.e.x.l.i.framework.FrameworkService - Started Enonic XP in 20285 ms
```

Log In

Click the **Launch Browser** button or point your browser to `http://localhost:8080`

Log in with username `su` and password `password`.

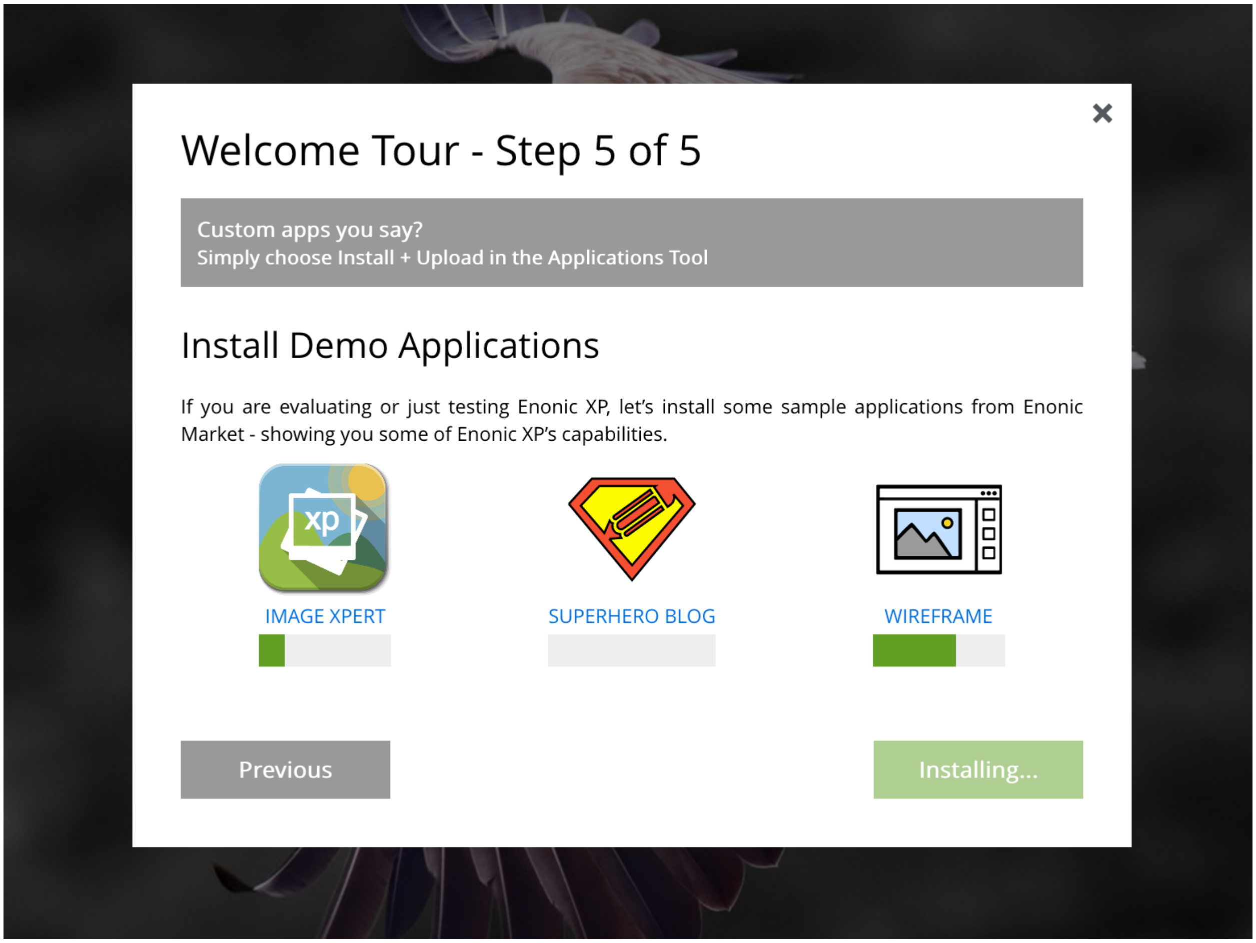
After logging in you should see the following screen, with the launcher panel to the right:



Add Sample Apps

If this is the first time you launch XP - the welcome tour will automatically launch. If it does not start, simply click the tour icon at top left of the home screen.

- Click through each step of the welcome tour
- On the last step, click the *Install* button
- Once the applications are installed, click finish



Welcome Tour - Step 5 of 5

Custom apps you say?
Simply choose Install + Upload in the Applications Tool

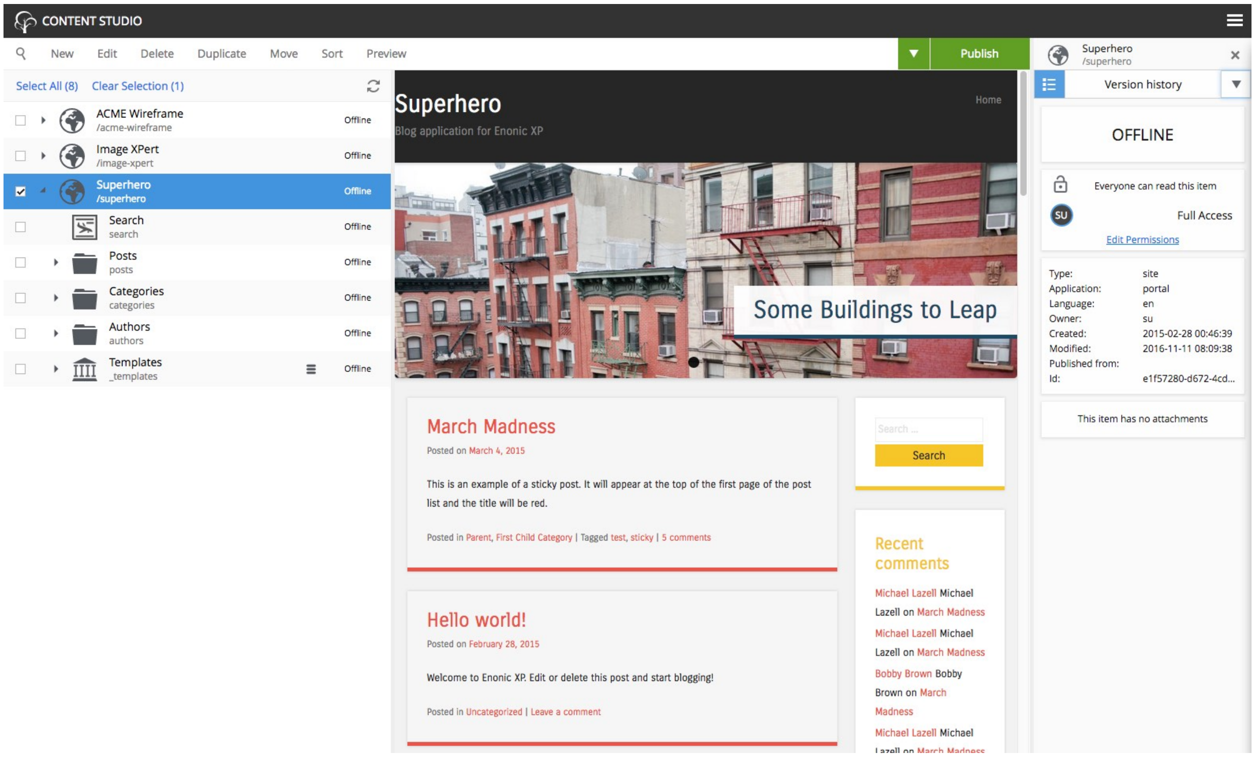
Install Demo Applications

If you are evaluating or just testing Enonic XP, let's install some sample applications from Enonic Market - showing you some of Enonic XP's capabilities.

Open Content Studio

The sample applications automatically create demo sites you can try out.

- Select "Content Studio" from the launcher menu
- Once loaded, you will find the sites in the tree grid
- Select or expand the sites you are interested in for a preview
- Right click or choose actions from the menu to get going



□	ACME Wireframe /acme-wireframe	Offline
□	Image XPert /image-xpert	Offline
☑	Superhero /superhero	Offline
□	Search search	Offline
□	Posts posts	Offline
□	Categories categories	Offline
□	Authors authors	Offline
□	Templates _templates	Offline

Superhero

Blog application for Enonic XP

March Madness

Posted on March 4, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Hello world!

Posted on February 28, 2015

Welcome to Enonic XP. Edit or delete this post and start blogging!

Posted in Uncategorized | Leave a comment

Michael Lazell Michael Lazell on March Madness

Michael Lazell Michael Lazell on March Madness

Bobby Brown Bobby Brown on March Madness

Michael Lazell Michael Lazell on March Madness

OFFLINE

Everyone can read this item

SU Full Access

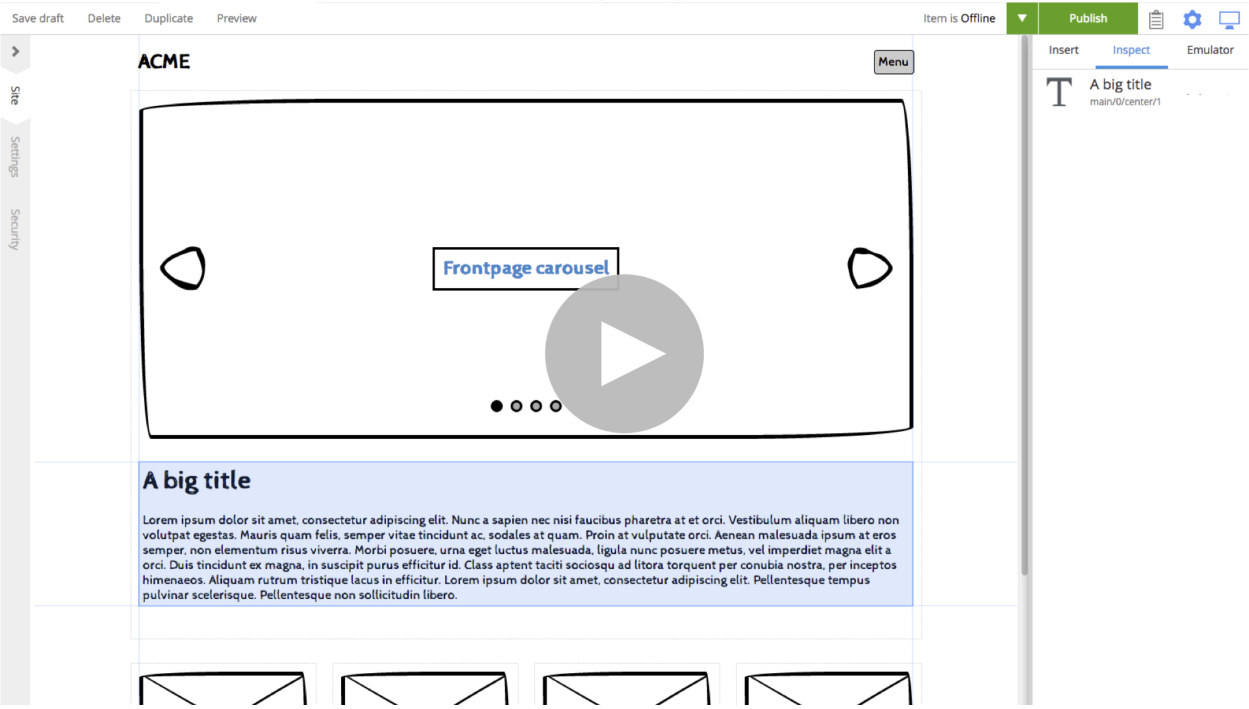
Edit Permissions

Type:	site
Application:	portal
Language:	en
Owner:	su
Created:	2015-02-28 00:46:39
Modified:	2016-11-11 08:09:38
Published from:	
Id:	e1f57280-d672-4cd...

This item has no attachments

Wireframe Prototyping Application

Watch this video to see how you can make interactive prototypes with Enonic XP and learn about Content Studio:



A big title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc a sapien nec nisi faucibus pharetra at et orci. Vestibulum aliquam libero non volutpat egestas. Mauris quam felis, semper vitae tincidunt ac, sodales at quam. Proin at vulputate orci. Aenean malesuada ipsum at eros semper, non elementum risus viverra. Morbi posuere, urna eget luctus malesuada, ligula nunc posuere metus, vel imperdiet magna elit a orci. Duis tincidunt ex magna, in suscipit purus efficitur id. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aliquam rutrum tristique lacus in efficitur. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque tempus pulvinar scelerisque. Pellentesque non sollicitudin libero.

Superhero Blog Application

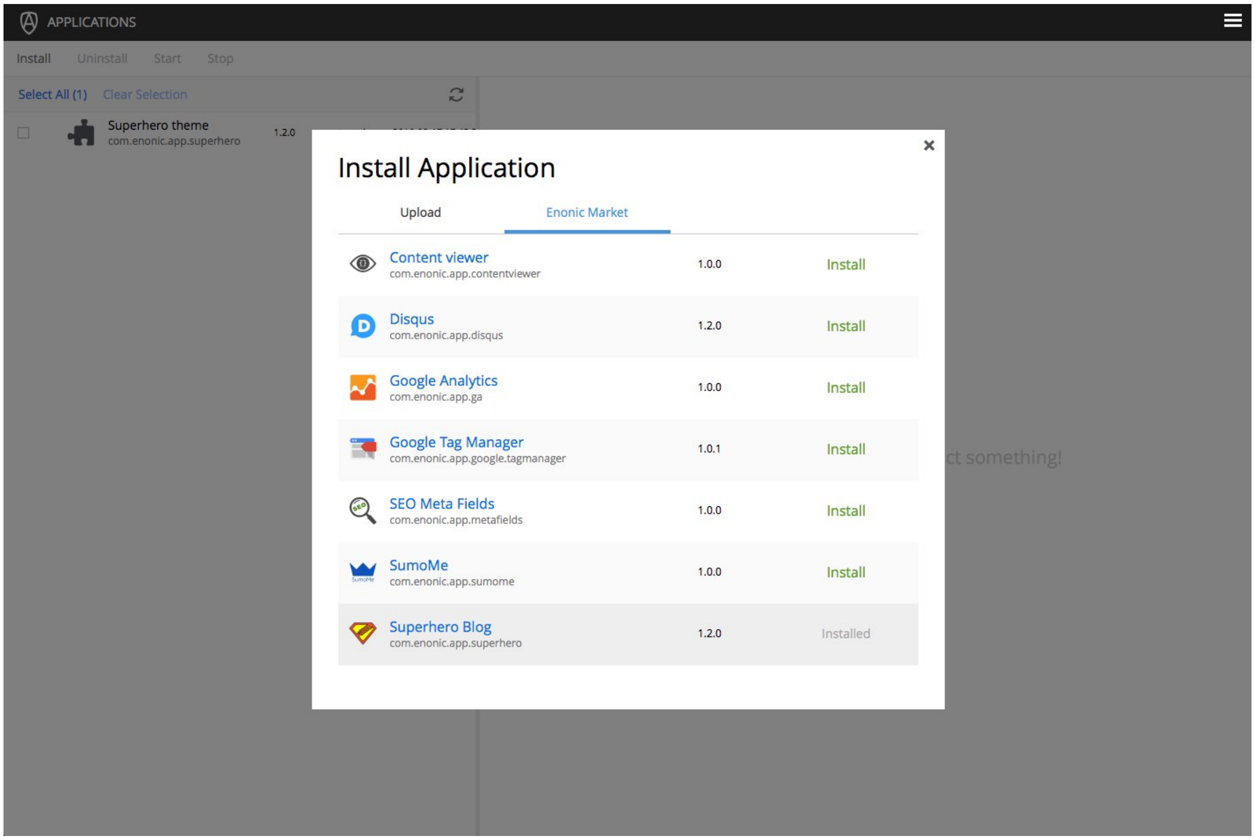
Watch this video to learn how you can use Enonic XP as a blogging platform, and learn more about Content Studio:

Visit Enonic Market

Find more on Enonic Market

If you want to try other applications, follow the steps below:

- Open the *Applications* tool from the launcher panel to the right
- Click *Install* from the menu (top left)
- Browse to find the applications you are looking for and click *Install*



Content viewer com.enonic.app.contentviewer	1.0.0	Install
Disqus com.enonic.app.disqus	1.2.0	Install
Google Analytics com.enonic.app.ga	1.0.0	Install
Google Tag Manager com.enonic.app.google.tagmanager	1.0.1	Install
SEO Meta Fields com.enonic.app.metafields	1.0.0	Install
SumoMe com.enonic.app.sumome	1.0.0	Install
Superhero Blog com.enonic.app.superhero	1.2.0	Installed

You may also visit Enonic market directly on https://market.enonic.com

Next Steps

Congratulations on getting started :-)

If you're ready for some more fun, we recommend looking into the following:

- Watch the Platform Video
- Hook up with the community on our forum
- Build your first Application with *My First App*
- Learn more about *Sites*

Troubleshooting

Verify that you comply with the minimum requirements for running XP on your local machine

Note: General Requirements

- Windows 7 or newer
- At least 1 GB of available memory
- HTTP port 8080 should be available (this can be changed if needed, see *Configuration*)

Docker

We´re huge devop fans - and devops love Docker. Docker is the most popular application container platform in the world. For your convenience, we build Docker images of every Enonic XP release.

Complete the following steps:

Install Docker

Running Enonic XP with Docker actually requires access to a Docker container - now, that's a surprise!

Note: Docker version 1.8.1 or newer is required to complete this guide

If you don't already have a Docker up and running, we recommend reading the brilliant documentation on how to get started with Docker:

- Docker for Windows
- Docker for OSX
- Docker for Linux

Start Server

Launch Enonic XP on Docker

With Docker up and running, installing Enonic XP is as smooth as baby skin. Execute the commands below in your terminal/shell to get going.

- Create a storage container for configuration files, applications and data (XP_HOME)

```
docker run -it --name xp-home enonic/xp-home
```

- Install and start Enonic XP, mounting the xp-home volume

```
docker run -d -p 8080:8080 --volumes-from xp-home --name xp-app enonic/xp-app
```

This will download the latest stable Enonic XP image, start it, and map it to port `8080` on your docker-host. You can optionally add :<versionnumber> at the end of the command to launch a specific version of Enonic XP - i.e.

```
docker run -d -p 8080:8080 --volumes-from xp-home --name xp-app enonic/xp-app:6.8.1
```

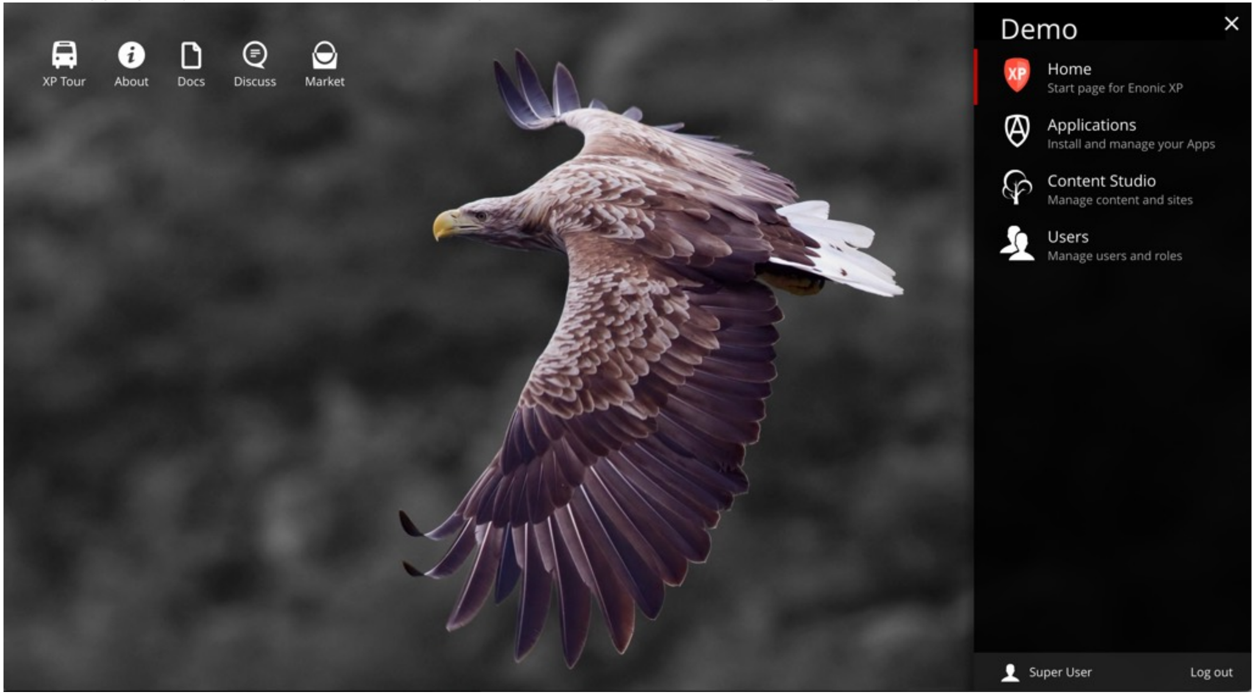
Check out our Project page at Docker Hub for more info.

Log In

Start by pointing your browser to `http://<mydockercontainer>:8080`

Log in with username `su` and password `password`.

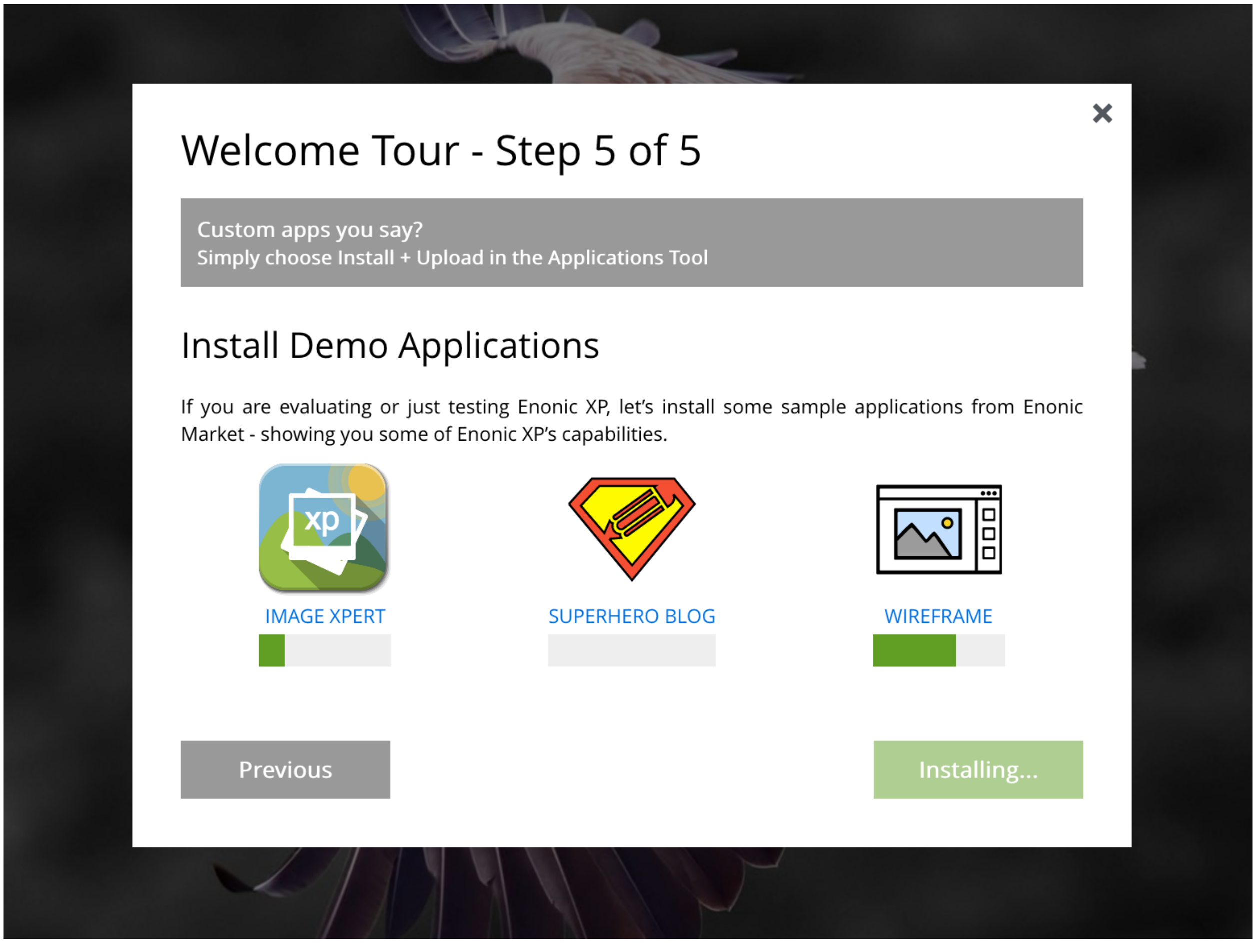
After logging in you should see the following screen, with the launcher panel to the right:



Add Sample Apps

If this is the first time you launch XP - the welcome tour will automatically launch. If it does not start, simply click the tour icon at top left of the home screen.

- Click through each step of the welcome tour
- On the last step, click the *Install* button
- Once the applications are installed, click finish



Welcome Tour - Step 5 of 5

Custom apps you say?
Simply choose Install + Upload in the Applications Tool

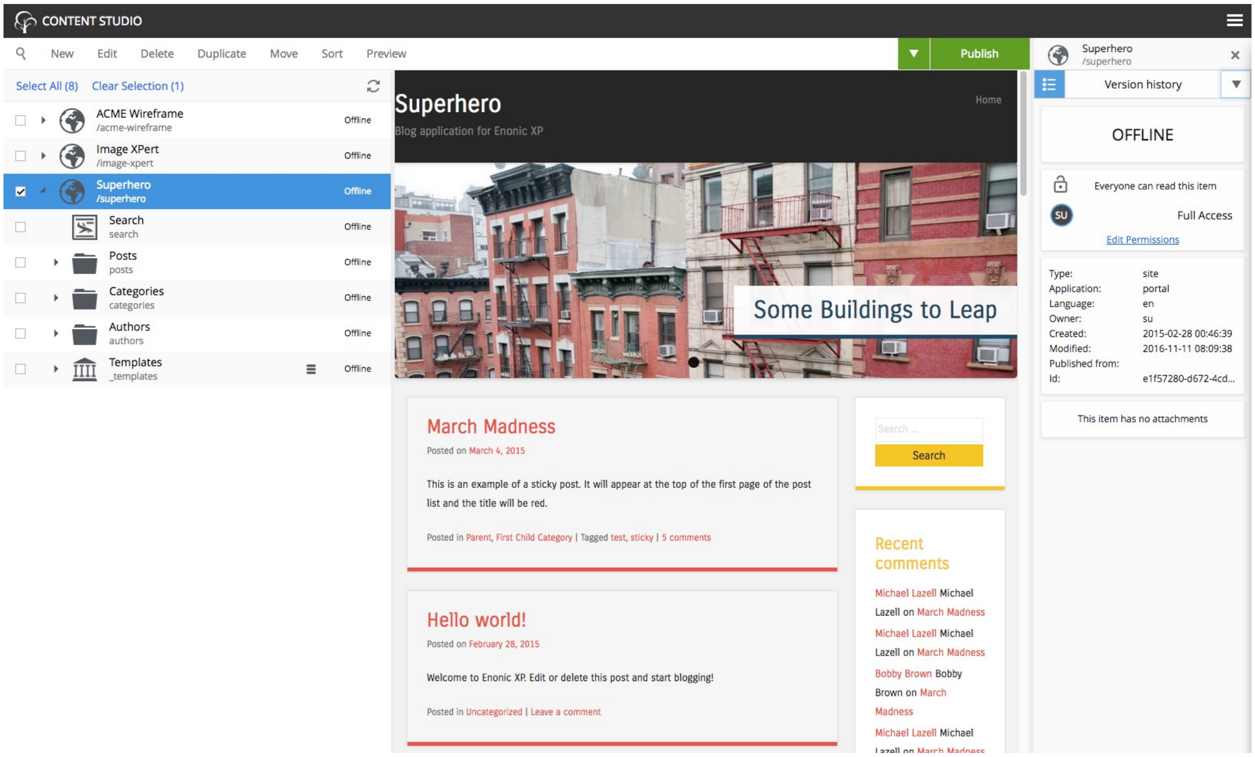
Install Demo Applications

If you are evaluating or just testing Enonic XP, let's install some sample applications from Enonic Market - showing you some of Enonic XP's capabilities.

Open Content Studio

The sample applications automatically create demo sites you can try out.

- Select "Content Studio" from the launcher menu
- Once loaded, you will find the sites in the tree grid
- Select or expand the sites you are interested in for a preview
- Right click or choose actions from the menu to get going



☐	ACME Wireframe /acme-wireframe	Offline
☐	Image XPert /image-xpert	Offline
☑	Superhero /superhero	Offline
☐	Search search	Offline
☐	Posts posts	Offline
☐	Categories categories	Offline
☐	Authors authors	Offline
☐	Templates _templates	Offline

Superhero

Blog application for Enonic XP

Some Buildings to Leap

March Madness

Posted on March 4, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Hello world!

Posted on February 28, 2015

Welcome to Enonic XP. Edit or delete this post and start blogging!

Posted in Uncategorized | Leave a comment

Michael Lazell Michael Lazell on March Madness

Michael Lazell Michael Lazell on March Madness

Bobby Brown Bobby Brown on March Madness

Michael Lazell Michael Lazell on March Madness

Superhero
/superhero

OFFLINE

Everyone can read this item

SU Full Access

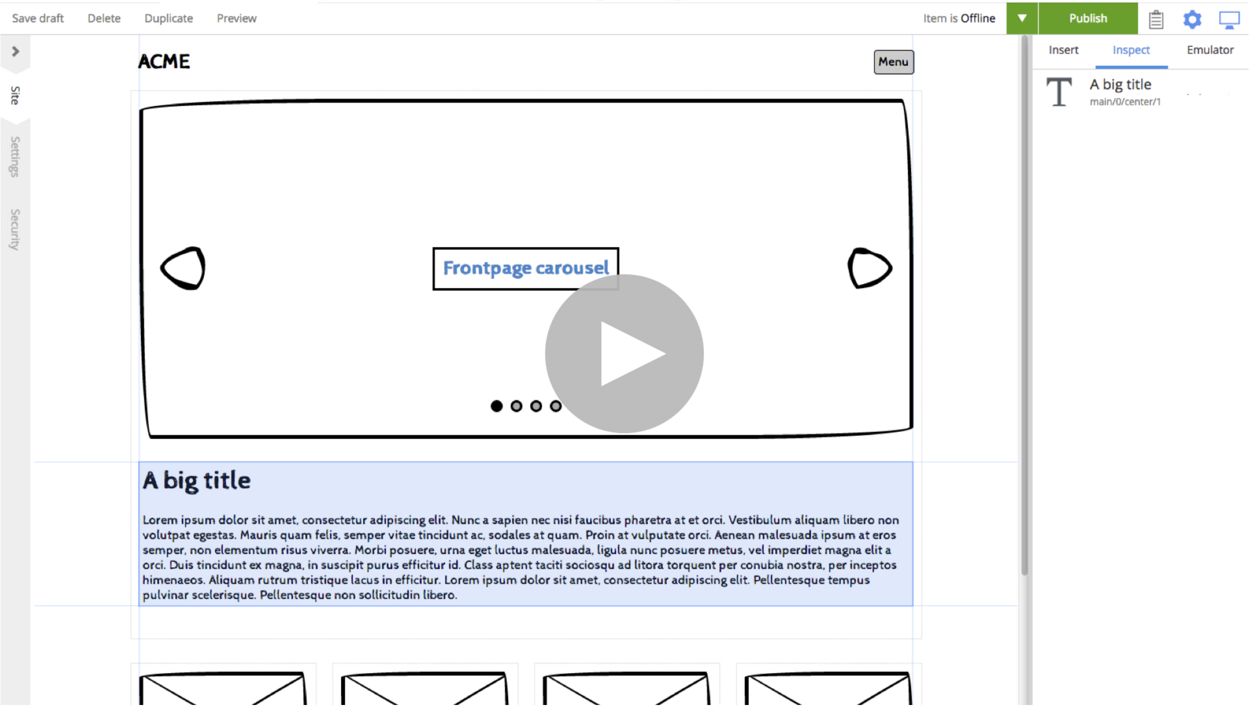
Edit Permissions

Type:	site
Application:	portal
Language:	en
Owner:	su
Created:	2015-02-28 00:46:39
Modified:	2016-11-11 08:09:38
Published from:	
Id:	e1f57280-d672-4cd...

This item has no attachments

Wireframe Prototyping Application

Watch this video to see how you can make interactive prototypes with Enonic XP and learn about Content Studio:



A big title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc a sapien nec nisi faucibus pharetra at et orci. Vestibulum aliquam libero non volutpat egestas. Mauris quam felis, semper vitae tincidunt ac, sodales at quam. Proin at vulputate orci. Aenean malesuada ipsum at eros semper, non elementum risus viverra. Morbi posuere, urna eget luctus malesuada, ligula nunc posuere metus, vel imperdiet magna elit a orci. Duis tincidunt ex magna, in suscipit purus efficitur id. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aliquam rutrum tristique lacus in efficitur. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque tempus pulvinar scelerisque. Pellentesque non sollicitudin libero.

Superhero Blog Application

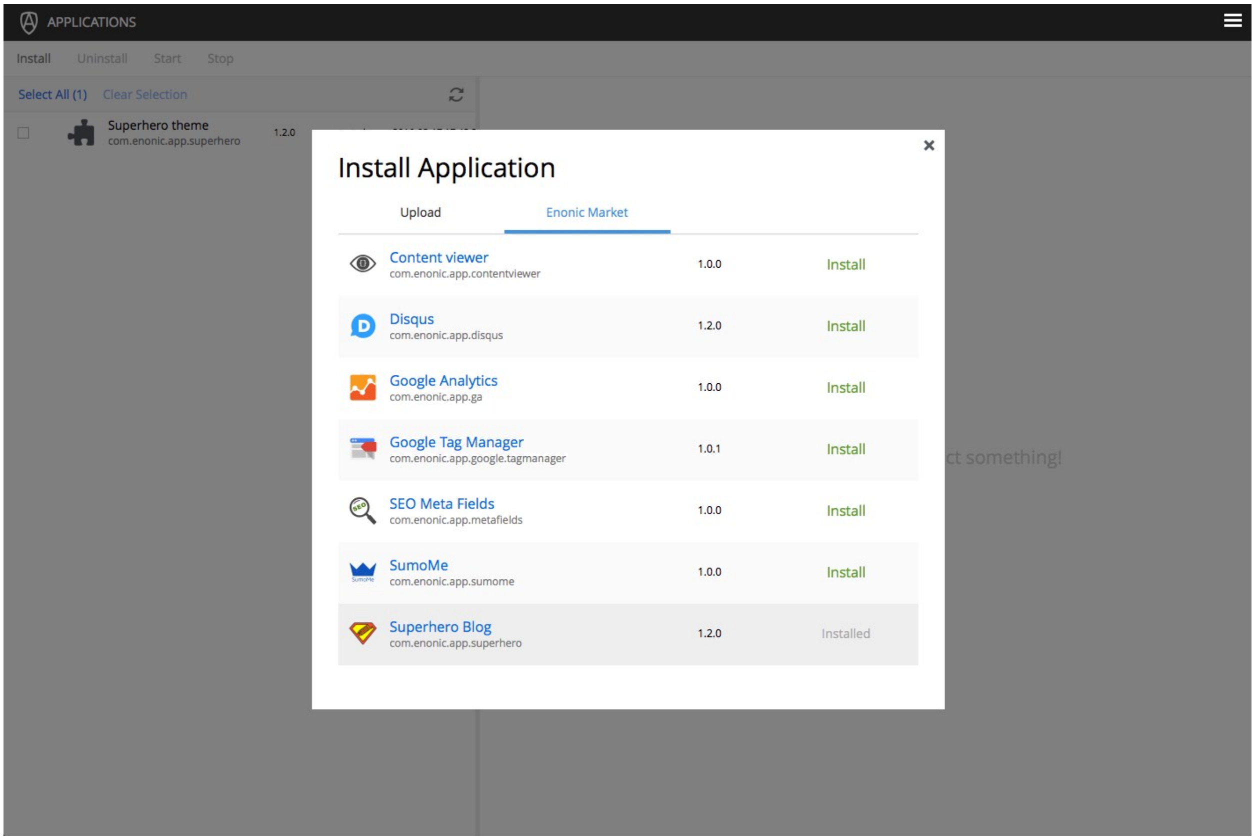
Watch this video to learn how you can use Enonic XP as a blogging platform, and learn more about Content Studio:

Visit Enonic Market

Find more on Enonic Market

If you want to try other applications, follow the steps below:

- Open the *Applications* tool from the launcher panel to the right
- Click *Install* from the menu (top left)
- Browse to find the applications you are looking for and click *Install*



Content viewer com.enonic.app.contentviewer	1.0.0	Install
Disqus com.enonic.app.disqus	1.2.0	Install
Google Analytics com.enonic.app.ga	1.0.0	Install
Google Tag Manager com.enonic.app.google.tagmanager	1.0.1	Install
SEO Meta Fields com.enonic.app.metafields	1.0.0	Install
SumoMe com.enonic.app.sumome	1.0.0	Install
Superhero Blog com.enonic.app.superhero	1.2.0	Installed

You may also visit Enonic market directly on https://market.enonic.com

Next Steps

Congratulations on getting started :-)

If you're ready for some more fun, we recommend looking into the following:

- Watch the Platform Video
- Hook up with the community on our forum
- Build your first Application with *My First App*
- Learn more about *Sites*

For Developers

This section describes how to install Enonic XP on any operating system. This is also the recommended approach for setting up a development environment. If you have any problems, please look at our *Troubleshooting* section.

Watch this video

or complete the following steps:

Install Java

Warning: To run Enonic XP, you need Java Development Kit (JDK) 1.8.92 JDK or newer.

Check JDK Version

If you're not sure what JDK version you have (or even if you have one), run the following in your terminal/shell:

```
javac -version
```

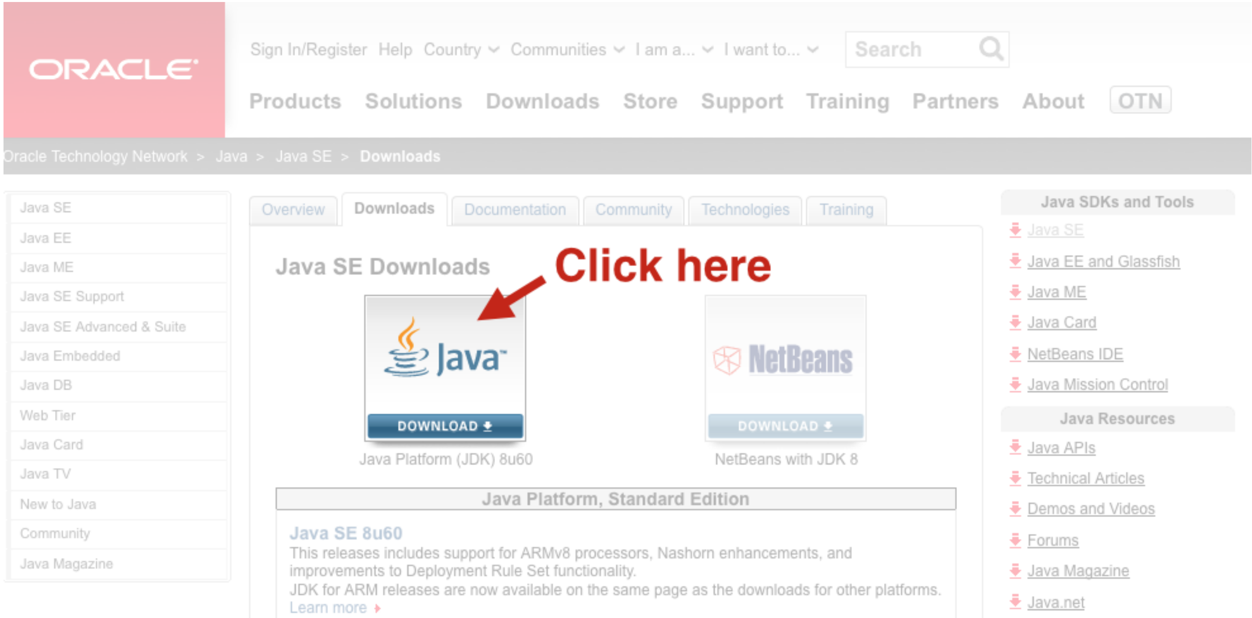
This should produce a response such as: "javac 1.8.0_112"

Having problems with your existing Java installation? Check out our *Troubleshooting Java* documentation.

Optionally Install Java

If it turns out you're on the wrong java version, follow these steps

- Download it from http://www.oracle.com/technetwork/java/javase/downloads/index.html



Java SE 8u60

This releases includes support for ARMv8 processors, Nashorn enhancements, and improvements to Deployment Rule Set functionality.

JDK for ARM releases are now available on the same page as the downloads for other platforms.

Learn more

- Follow the instructions for your respective operating system

Download Enonic XP

Enonic XP is available in a simple universal distribution file - running on all plaforms (Windows, Linux, OSX etc)

- Download Enonic XP distribution
- Unzip the file to a suitable location

Command-line version (OSX/Linux only):

```
curl -O http://repo.enonic.com/public/com/enonic/xp/distro/6.8.1/distro-6.8.1.zip
unzip distro-6.8.1.zip
cd enonic-xp-6.8.1
```

Next - let’s get the server started

Start the server

Now that the software has been downloaded, you’re ready to start the server - start the respective file from command line.

Linux and OS X:

```
[XP Installation Folder]/bin/server.sh
```

Windows:

```
[XP Installation Folder]\bin\server.bat
```

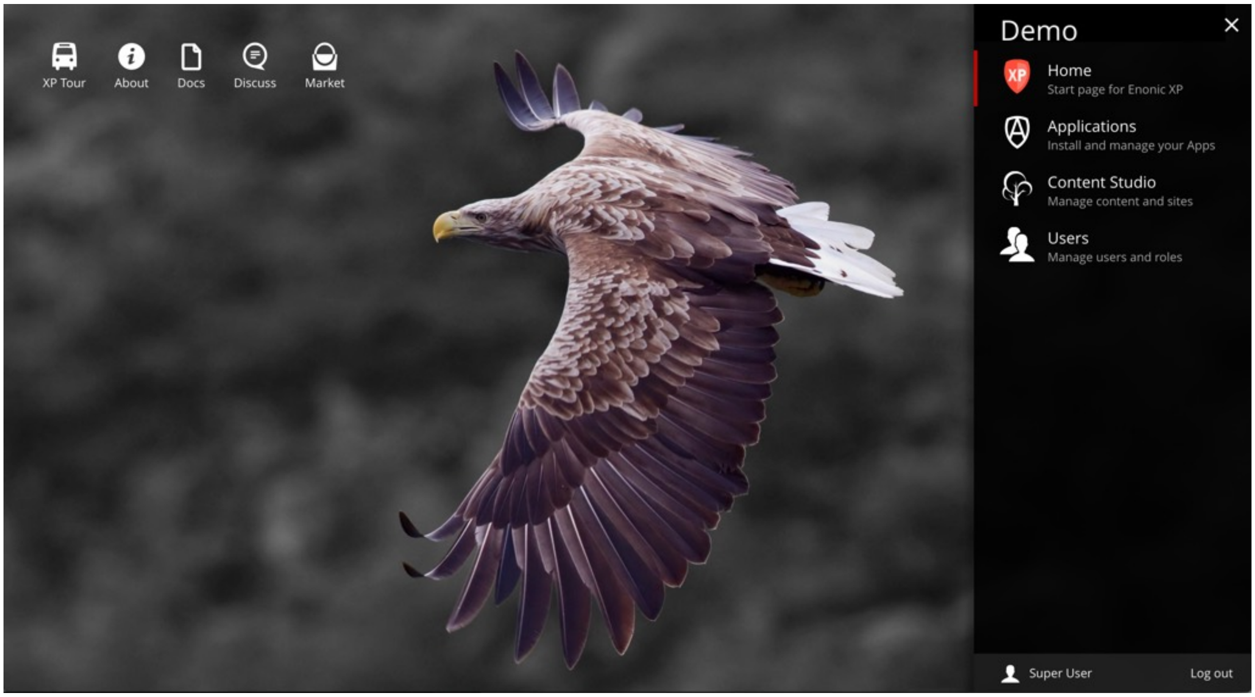
This will start Enonic XP. When successfully started, the following will appear at the end of the log:

```
12:53:14.302 INFO  c.e.x.l.framework.FrameworkService - Started Enonic XP in 7378 ms
```

Log In

Log in with username `su` and password `password`.

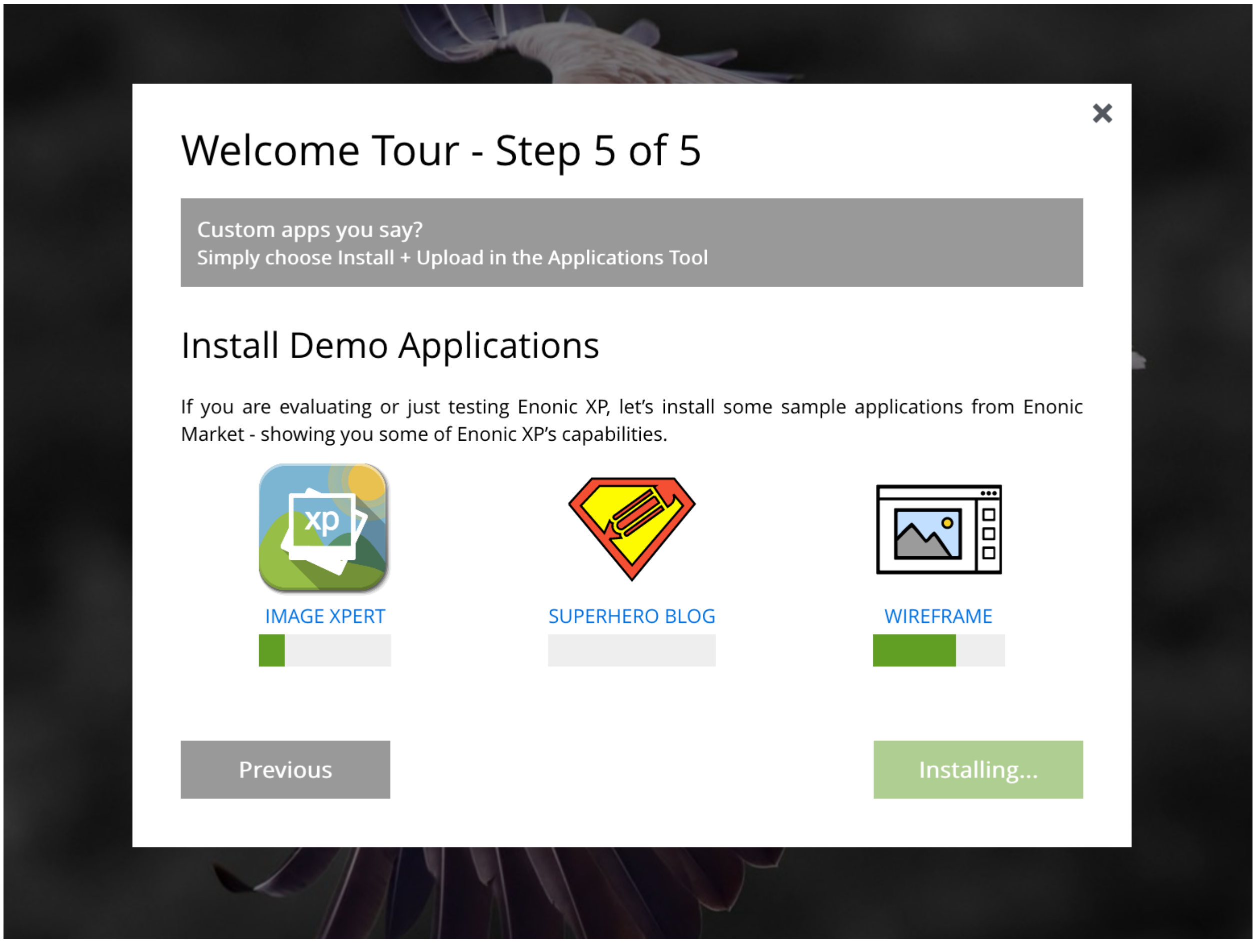
After logging in you should see the following screen, with the launcher panel to the right:



Add Sample Apps

If this is the first time you launch XP - the welcome tour will automatically launch. If it does not start, simply click the tour icon at top left of the home screen.

- Click through each step of the welcome tour
- On the last step, click the *Install* button
- Once the applications are installed, click finish



Welcome Tour - Step 5 of 5

Custom apps you say?
Simply choose Install + Upload in the Applications Tool

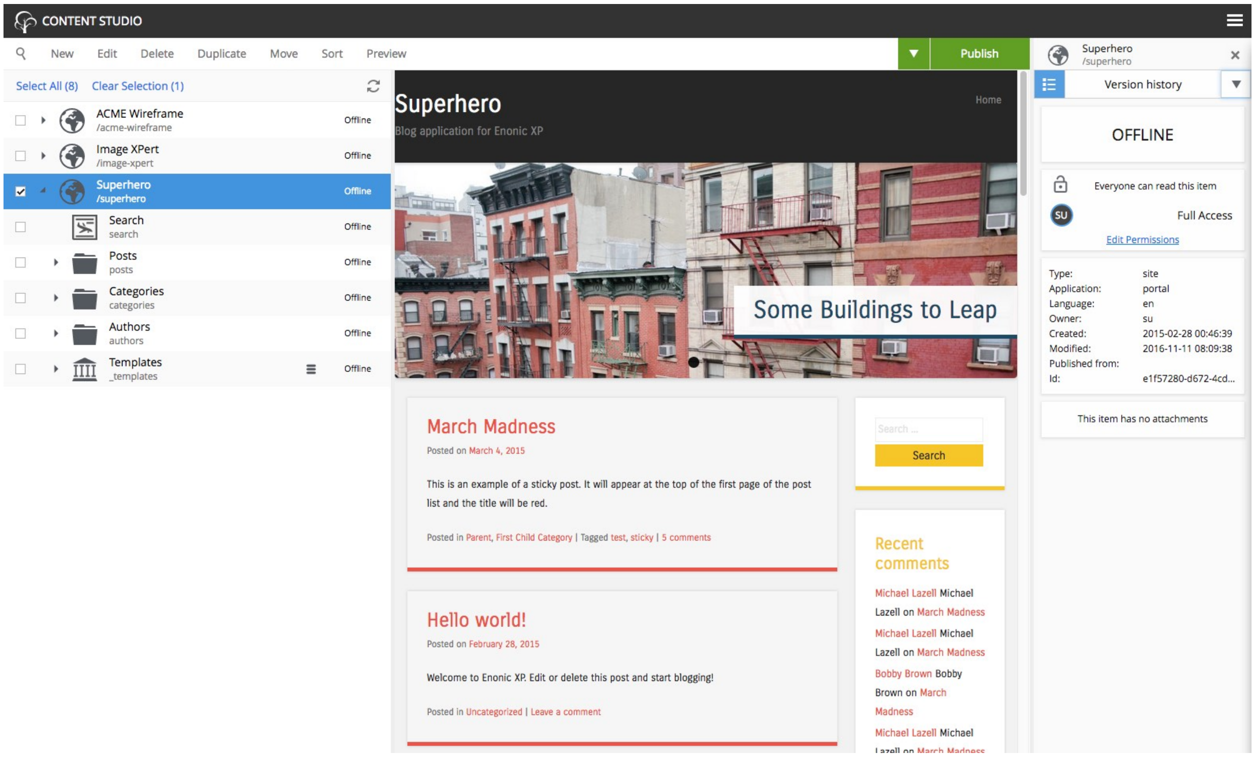
Install Demo Applications

If you are evaluating or just testing Enonic XP, let's install some sample applications from Enonic Market - showing you some of Enonic XP's capabilities.

Open Content Studio

The sample applications automatically create demo sites you can try out.

- Select "Content Studio" from the launcher menu
- Once loaded, you will find the sites in the tree grid
- Select or expand the sites you are interested in for a preview
- Right click or choose actions from the menu to get going



Superhero

Blog application for Enonic XP

March Madness

Posted on March 4, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Hello world!

Posted on February 28, 2015

Welcome to Enonic XP. Edit or delete this post and start blogging!

Posted in Uncategorized | Leave a comment

Michael Lazell Michael Lazell on March Madness

Michael Lazell Michael Lazell on March Madness

Bobby Brown Bobby Brown on March Madness

Michael Lazell Michael Lazell on March Madness

Everyone can read this item

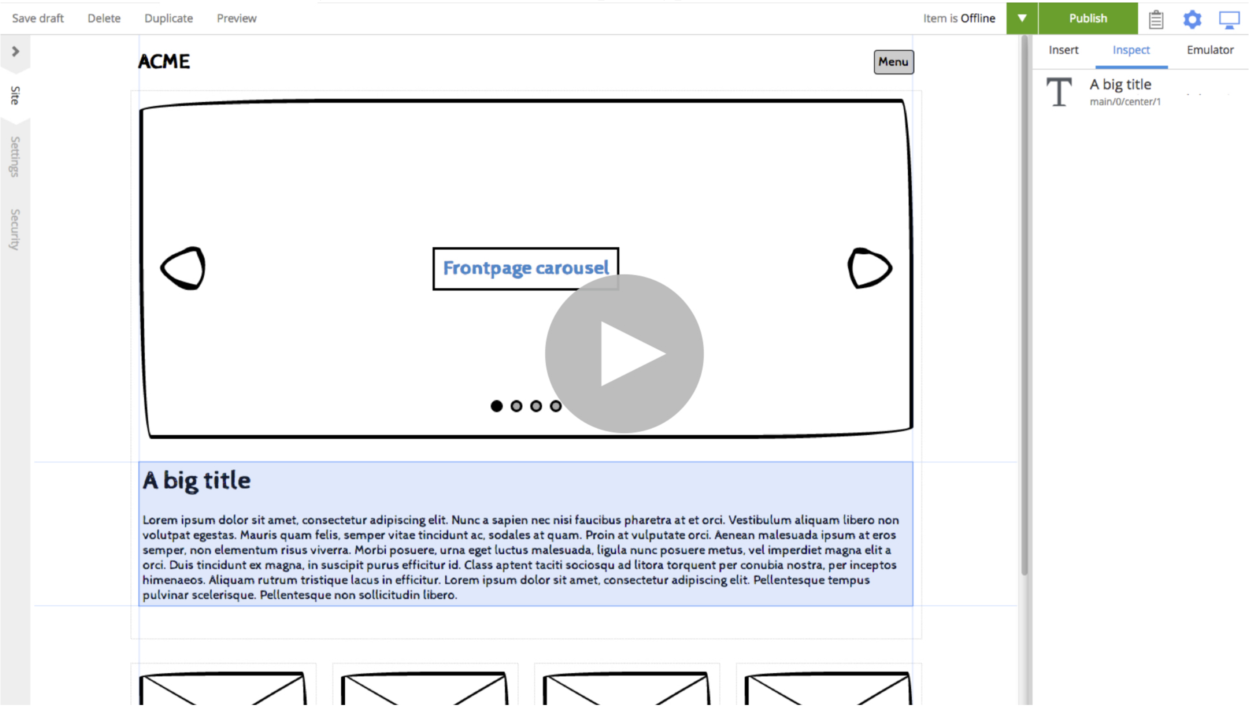
Full Access

Type: site
Application: portal
Language: en
Owner: su
Created: 2015-02-28 00:46:39
Modified: 2016-11-11 08:09:38
Published from:
Id: e1f57280-d672-4cd...

This item has no attachments

Wireframe Prototyping Application

Watch this video to see how you can make interactive prototypes with Enonic XP and learn about Content Studio:



A big title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc a sapien nec nisi faucibus pharetra at et orci. Vestibulum aliquam libero non volutpat egestas. Mauris quam felis, semper vitae tincidunt ac, sodales at quam. Proin at vulputate orci. Aenean malesuada ipsum at eros semper, non elementum risus viverra. Morbi posuere, urna eget luctus malesuada, ligula nunc posuere metus, vel imperdiet magna elit a orci. Duis tincidunt ex magna, in suscipit purus efficitur id. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aliquam rutrum tristique lacus in efficitur. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque tempus pulvinar scelerisque. Pellentesque non sollicitudin libero.

Superhero Blog Application

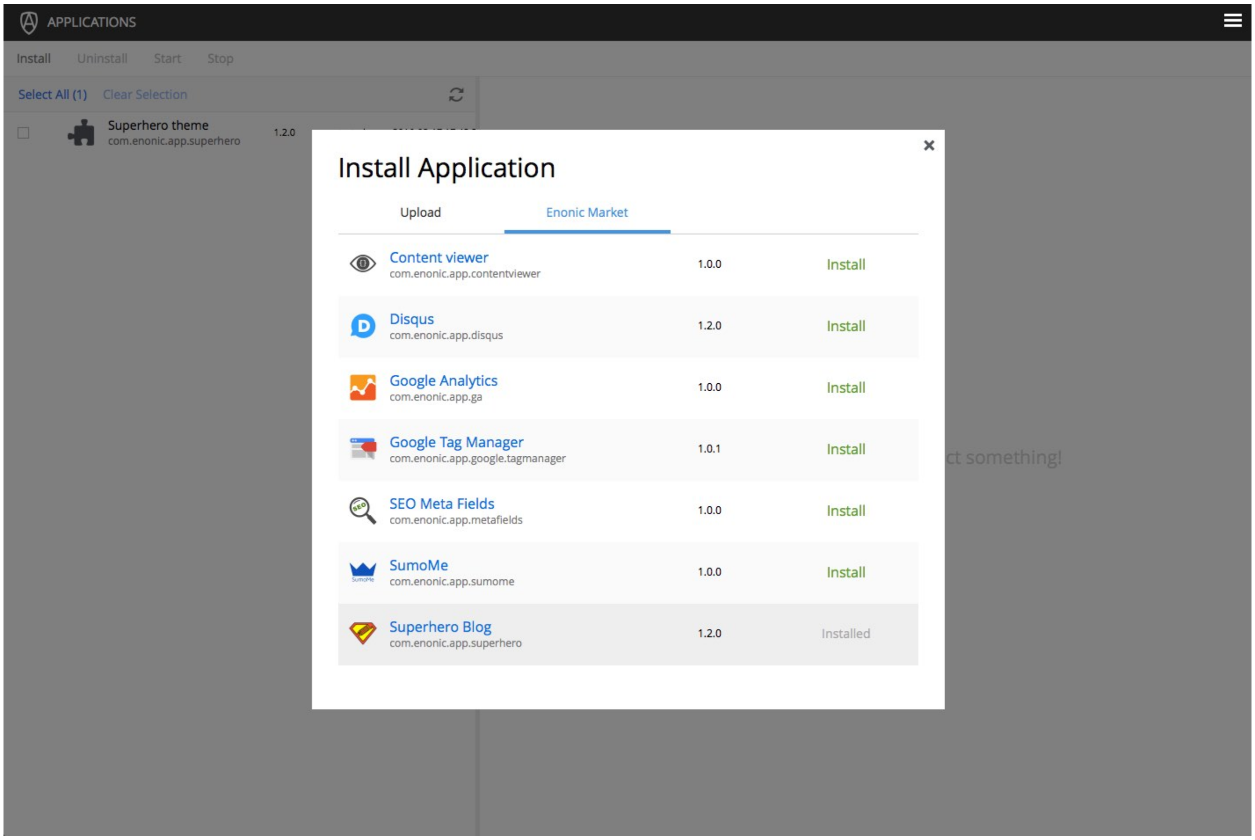
Watch this video to learn how you can use Enonic XP as a blogging platform, and learn more about Content Studio:

Visit Enonic Market

Find more on Enonic Market

If you want to try other applications, follow the steps below:

- Open the *Applications* tool from the launcher panel to the right
- Click *Install* from the menu (top left)
- Browse to find the applications you are looking for and click *Install*



Content viewer com.enonic.app.contentviewer	1.0.0	Install
Disqus com.enonic.app.disqus	1.2.0	Install
Google Analytics com.enonic.app.ga	1.0.0	Install
Google Tag Manager com.enonic.app.google.tagmanager	1.0.1	Install
SEO Meta Fields com.enonic.app.metafields	1.0.0	Install
SumoMe com.enonic.app.sumome	1.0.0	Install
Superhero Blog com.enonic.app.superhero	1.2.0	Installed

You may also visit Enonic market directly on https://market.enonic.com

Next Steps

Congratulations on getting started :-)

If you're ready for some more fun, we recommend looking into the following:

- Watch the Platform Video
- Hook up with the community on our forum
- Build your first Application with *My First App*
- Learn more about *Sites*

Troubleshooting

Verify that you comply with the minimum requirements for running XP on your local machine

Note: General Requirements

- Any OS supporting the required version of Java (Mac, Linux, Windows etc)
- At least 1 GB of available memory
- HTTP port 8080 should be available (this can be changed if needed, see *Configuration*)

CHAPTER 2

Tutorials

Ready to learn something new? Well you've come to the right place!

You will find our tutorials below.

Project Init (Video)

This video demonstrates how to initialize a new application project using existing project or starter kits

Here are some additional resources related to application development

- *Developer Guide*
- *Projects*
- *Libraries*

Javascript MVC (Video)

This video demonstrates how to build site html parts using Javascript as controller and Thymeleaf as view technology.

Here are some resources related to site development

- *Sites*
- *Content Types*
- *Views*
- *Part*

My First App

This guide will lead you through the required steps to build the "Hello World" app for Enonic XP.

In this tutorial, you will learn how to initialize new application projects and deploy them. We will create a simple website that displays a list of countries and cities with a Google map of each city. Upon completion, you will be familiar with content types, page and part components, page templates, regions, and the Content Studio app. You won't be writing any code - just copy/paste from the examples.

The screen-shots below show the final product of this tutorial.

Note: To complete this tutorial, you will need a local running installation (see *For Developers*) of Enonic XP and a text editor of your choice. All terminal actions assume you're using OSX or Linux.

Initialize project

Enonic XP includes the *Toolbox CLI* which can perform several useful operations. The *init-project* operation will clone an existing project from a repository source, such as GitHub. The starter-vanilla project will initialize a new application with the standard structures required (see *Projects*).

1. Create a new folder at a suitable location on your filesystem for the application project files. e.g. `/Users/<username>/projects/myapp` This will be the project root.
2. Change directory in the terminal to this project root.
3. Run the following command, replacing [$XP_INSTALL] with the path to your unzipped XP installation:

```
[$XP_INSTALL]/toolbox/toolbox.sh init-project -n com.company.myapp -r starter-vanilla -c 1.0.0
```

Tip: Only basic characters (a-z, 0-9 and .) must be used for application names, and the name must be globally unique. We recommend following standard Java package naming conventions such as com.mycompany.myapp.

Your project folder will now be filled with the standard folder structure for developing an app.

Build and Deploy

Now that we have set up a project, we should test that it builds and deploys successfully. But before deploying the app, the `$XP_HOME` environment variable must be set to the path of the home folder of the XP installation.

1. Run the following command in the terminal, replacing [$XP_INSTALL] with your installation location (no brackets):

Linux and OSX:

```
export XP_HOME=[XP Installation Folder]/home
```

Windows:

```
set XP_HOME=[XP Installation Folder]/home
```

2. Execute the following command (from the project root directory):

Linux and OSX:

```
./gradlew deploy
```

Windows:

```
gradlew deploy
```

The included Gradle wrapper will build the app and then attempt to deploy it to your installation.

The deployment step simply moves the result of the build (the application JAR file) into the `$XP_HOME/deploy` directory. From there, Enonic XP will detect, install and start the application automatically.

You will need to access the Administrative console to check that the app has installed and started.

3. Log in to the Administrative console (http://localhost:8080) with the Administrative user credentials (userid **su** and password **password**).
4. Navigate to the Applications Tool. The application you just deployed should be listed here.
5. Click the app called "Myapp" to see information about it and confirm that it has started.

Note: You can change the display name of the application by editing the gradle.properties file.

Create the Hello World Site

Our next goal is to set up a "Hello World" site in Content Studio, but first we must add some initial configuration to our project.

Site descriptor

An application can serve many purposes and building sites is just one of them. The `site.xml` file is the descriptor that will let Enonic XP know that this app can be added to a site. Site-wide configurations can be defined in this file but we will leave the config element empty for now (see *Site Descriptors*).

A basic site.xml file was automatically created by the init-project script:

```
[project-root]/src/main/resources/site/site.xml
```

Note: All of the files we will be working with are below the "site" directory in the project folder - src/main/resources/site. All file paths from now on will begin with "site/".

Page Component

Page components are the most basic building blocks of websites in Enonic XP (see *Page*). They require a JavaScript controller and optionally an XML descriptor and an HTML view. This first example does not need a descriptor file.

A page controller (see *Page*) is a JavaScript file that handles requests such as GET and POST. Controllers usually pass data in the form of a JavaScript object to be dynamically rendered in an HTML view. No data is passed in the example below, but the view file is specified and rendered as static HTML.

1. Create a folder called `hello` inside the `site/pages` directory.
2. Create the page controller and page view files specified below inside the `hello` folder:

Listing 2.1: Hello page controller - site/pages/hello/hello.js

```
var thymeleaf = require('/lib/xp/thymeleaf'); // Import the thymeleaf render function

// Handle the GET request
exports.get = function(req) {

    // Specify the view file to use
    var view = resolve('hello.html');

    // Render HTML from the view file
    var body = thymeleaf.render(view, {});
    // Return the response object
    return {
        body: body
    }
};
```

The *view* below is a simple HTML file. This file will be updated later to handle dynamic content.

Listing 2.2: Hello page view - site/pages/hello/hello.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Hello world</title>
    </head>
    <body data-portal-component-type="page">
        <h1>Hello world</h1>
    </body>
</html>
```

3. Once these files are in place, redeploy the app from the terminal with `./gradlew deploy`.

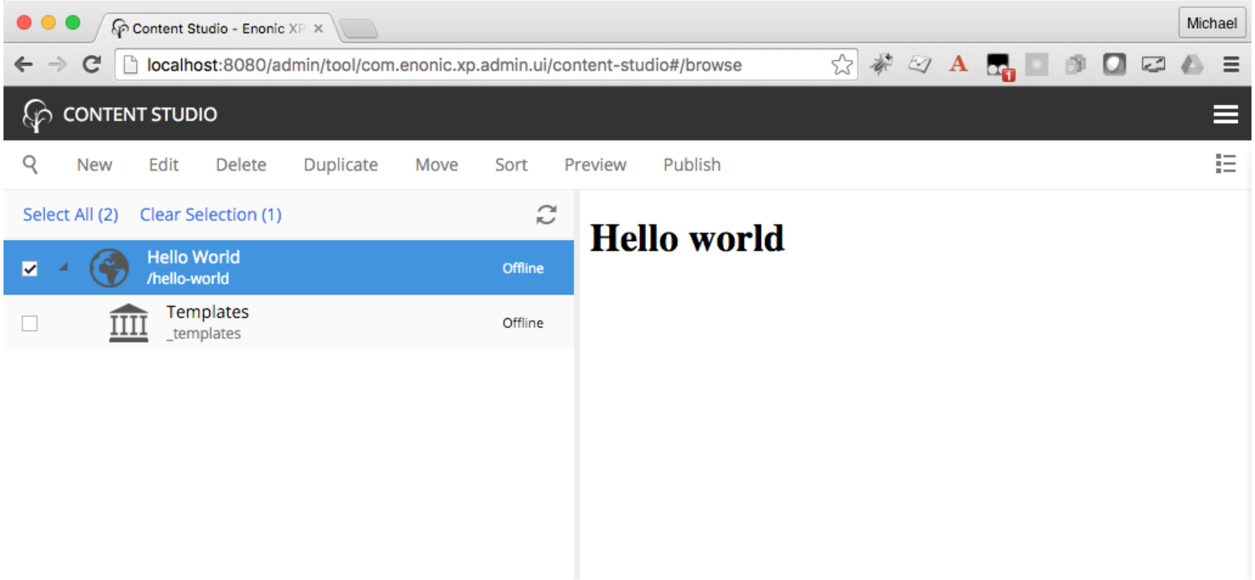
Tip: Each page controller must reside in its own folder under the `site/pages` directory. The name of the controller JavaScript file must be the same as the directory that contains it. The HTML view file can reside anywhere in the project and have any valid file name. This allows view files to be shared between components.

Create Site

Now that the files are in place, we can create the site in a browser using the Content Studio admin tool. Switch between different tools by clicking the menu icon ☰ (top right) to open the Launcher panel.

1. In your browser, navigate to the Content Studio tool. (Use the menu icon at the top right)
2. Click "New" and select "Site" from the list of content types (Opens a tab for editing the new site).
3. Fill in the form with Display Name: "Hello World".
4. Select your "MyApp" application in the "Applications" dropdown.
5. If you don't see a blue area on the right of the page then click this button 🖥 in the toolbar to open the Page Editor.
6. Use the dropdown in the Page Editor (blue area) to select the "hello" page.
7. Click the "Save draft" button in the toolbar (top-left).
8. Now close the "Hello World" site editor tab to see the content pane.

When you click on the "Hello World" site content, the preview should look something like this:



Add some Countries

In order to make our "World" slightly more interesting, we need some data - or more specifically countries.

To add structured data (such as countries), we need so-called *Content Types*. The content type defines the form (and underlying schema) of items you manage.

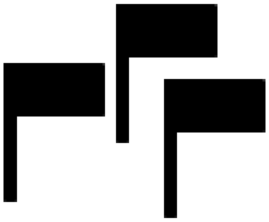
1. Create a folder called "country" inside the "content-types" folder of your project.
2. Add the Country content type file below to this folder.

Listing 2.3: Country content type - site/content-types/country/country.xml

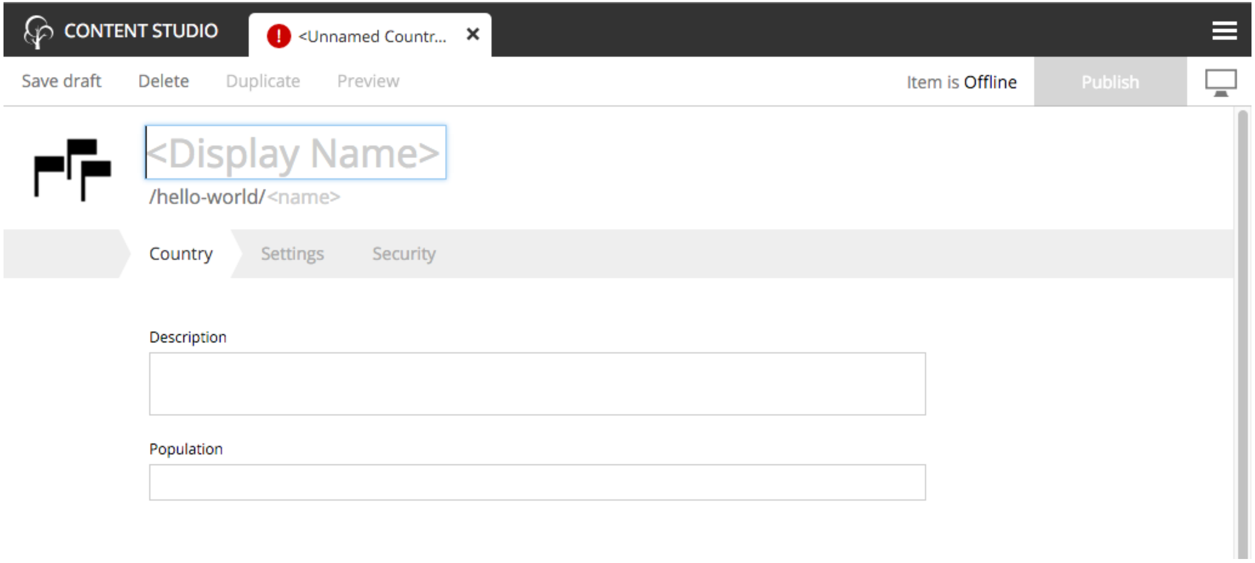
```
<content-type>
  <display-name>Country</display-name>
  <super-type>base:structured</super-type>
  <form>
    <input type="TextArea" name="description">
      <label>Description</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
    <input type="TextLine" name="population">
      <label>Population</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
  </form>
</content-type>
```

Each content type can have a custom icon that will be visible in the Content Studio interface. Though not required, content icons can be helpful for content editors.

3. Copy the image below to the the same folder (content-types/country) with the name *country.png*.



This content type defines form inputs for **description** and **population**. Every content has a built-in field for **Display Name**. When the app is redeployed, this content type will produce the form seen below in the Content Studio app.



Tip: Each content type must reside in its own folder under the `site/content-types` directory. The name of the content type XML file and the icon PNG file must be the same as the directory that contains them.

Create the Country Part

We also need a way to present a country - because every country wants to be seen. This time, rather than just making another page controller, we will create a *Part* component. Parts are reusable components that can be added to pages containing "regions" - more on this later.

1. Create a folder called "country" inside the "parts" folder in your project.
2. Add the part **controller** and **view** files below to the "country" folder:

Listing 2.4: Country part controller - site/parts/country/country.js

```
var portal = require('/lib/xp/portal'); // Import the portal functions
var thymeleaf = require('/lib/xp/thymeleaf'); // Import the Thymeleaf rendering function

// Handle the GET request
exports.get = function(req) {

    // Get the country content as a JSON object
    var content = portal.getContent();
    // Prepare the model object with the needed data from the content
    var model = {
        name: content.displayName,
        description: content.data.description,
        population: content.data.population
    };

    // Specify the view file to use
    var view = resolve('country.html');

    // Return the merged view and model in the response object
    return {
        body: thymeleaf.render(view, model)
    }
};
```

The part controller file above handles the GET request and passes the country content data to the view file which is shown below.

Listing 2.5: Country part view - site/parts/country/country.html

```
<div>
    <h3 data-th-text="${name}"></h3>
    <div data-th-if="${population}" data-th-text="'Population: ' + ${population}"></div>
    <div data-th-if="${description}" data-th-text="${description}"></div>
</div>
```

The Hello Region Page

Parts start to make sense when placed into a *region*. Regions are “slots” contained within pages or layouts. Pages and layouts may contain multiple regions, and each region must have a unique name.

Let’s create a new page component with a single region called “Main”. We will later place the “Country” part into this region.

The benefit of a region (see *Regions*) is that a page component can be re-used across multiple different pages by simply adding different parts to them as needed.

1. Create a folder called “hello-region” in your project’s `site/pages/` folder.
2. Add the “Hello region” page descriptor, controller and view files:

Listing 2.6: Page descriptor - site/pages/hello-region/hello-region.xml

```
<page>
  <display-name>Hello Region</display-name>
  <config/>
  <regions>
    <region name="main"/>
  </regions>
</page>
```

The XML file above is a *Descriptor*. Regions and page configurations can be defined here.

Listing 2.7: Page controller - site/pages/hello-region/hello-region.js

```
var portal = require('/lib/xp/portal'); // Import the portal functions
var thymeleaf = require('/lib/xp/thymeleaf'); // Import the Thymeleaf rendering function

// Handle the GET request
exports.get = function(req) {

    // Get the content that is using the page
    var content = portal.getContent();

    // Extract the main region which contains component parts
    var mainRegion = content.page.regions.main;

    // Prepare the model that will be passed to the view
    var model = {
        mainRegion: mainRegion
    }

    // Specify the view file to use
    var view = resolve('hello-region.html');

    // Render the dynamic HTML with values from the model
    var body = thymeleaf.render(view, model);

    // Return the response object
    return {
        body: body
    }
};
```

This page controller uses a portal library (see `lib-portal` in *Javascript Libraries*) to get the content and extract the "main" region which was defined in the descriptor XML file.

Listing 2.8: Page view - site/pages/hello-region/hello-region.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello world</title>
</head>
<body data-portal-component-type="page">
    <h1>Country</h1>
    <div data-portal-region="main">
        <div data-th-if="${mainRegion}" data-th-each="component : ${mainRegion.components}" data-th-r
            <div data-portal-component="${component.path}" data-th-remove="tag"></div>
        </div>
    </div>
</body>
</html>
```

The view file above defines the place on the page where the region will render component parts that are dragged and dropped in the Page Editor.

3. When done - redeploy your app once again!

```
./gradlew deploy
```

Tip: You can restart XP in *Development mode* and then the app won't have to be redeployed after making changes.

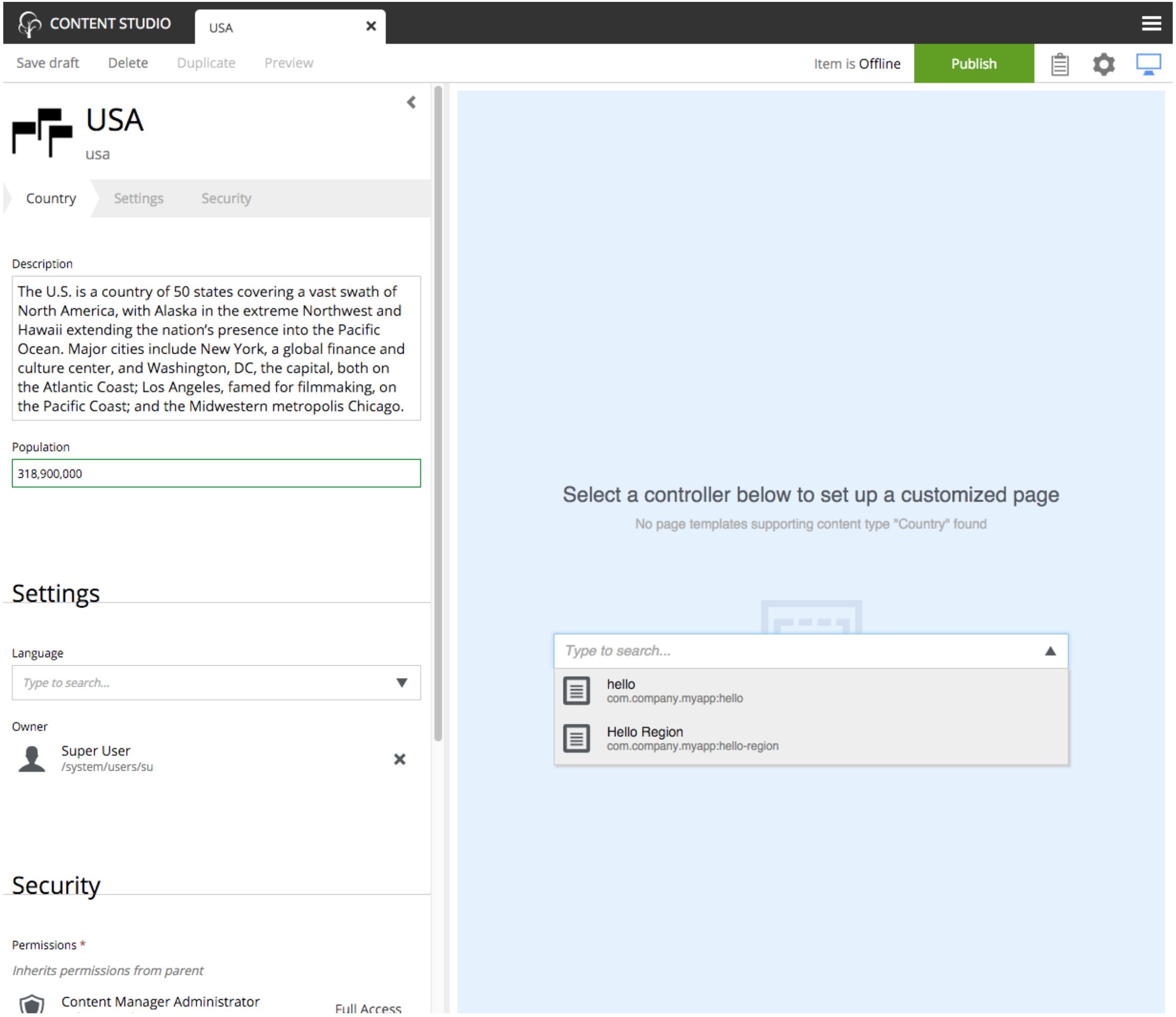
Add your favorite country

Now that the "Country" content type is installed (and we have a part to display them), we can create new countries using the Content Studio interface.

1. Right-click on the "Hello World" site from the navigation tree and select "New". The "Create Content" dialogue will open.
2. Click "Country" from the list of content types.
3. Fill in the form with the details of your favorite country.

Similar to the site, we must also configure a view for the country

4. In the toolbar, in the top right corner of the page, click the button with the monitor icon 🖥 to activate the Page Editor (blue background).



Description: The U.S. is a country of 50 states covering a vast swath of North America, with Alaska in the extreme Northwest and Hawaii extending the nation's presence into the Pacific Ocean. Major cities include New York, a global finance and culture center, and Washington, DC, the capital, both on the Atlantic Coast; Los Angeles, famed for filmmaking, on the Pacific Coast; and the Midwestern metropolis Chicago.

Population: 318,900,000

Select a controller below to set up a customized page

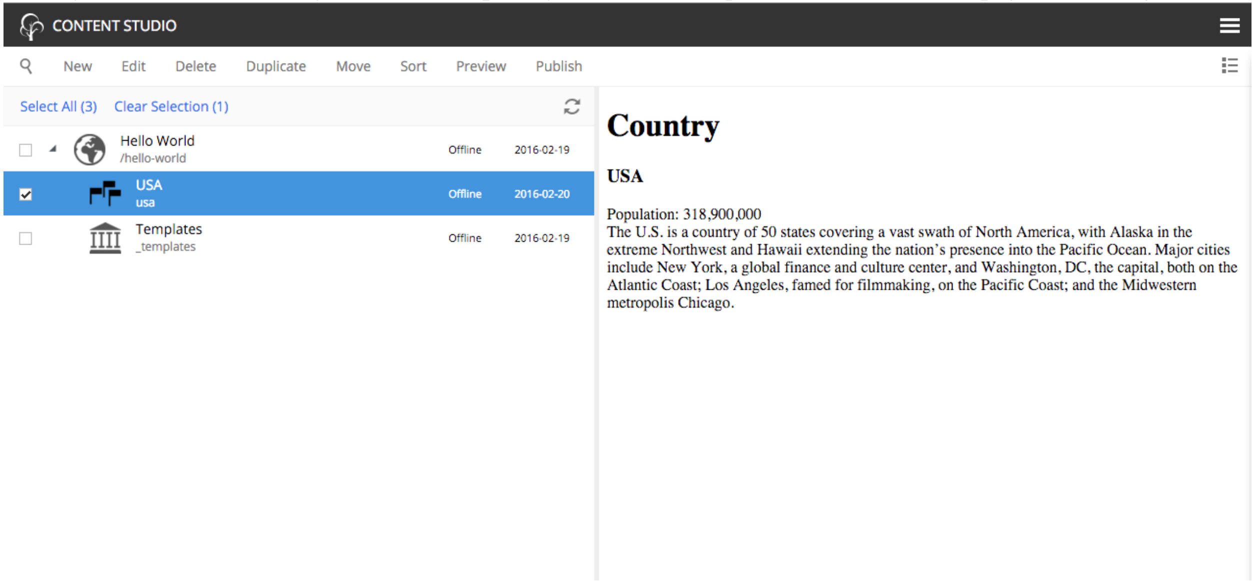
No page templates supporting content type "Country" found

hello
com.company.myapp:hello

Hello Region
com.company.myapp:hello-region

5. In the Page Editor, select “Hello Region” from the template selector dropdown. If the dropdown arrow is not visible, double-click inside the option field or start typing “hello world” in it to see the options.
6. Click the cog button to open the Inspection Panel (far right).
7. In the Inspection Panel, click the “Insert” tab. This reveals a list of default components that can be placed into regions.
8. Click and drag the “Part” into the box on the page.
9. A new dropdown option will appear. Select the “country” part.
10. Save draft and close the content edit tab.

When you click on the country in the content pane, you should see a preview of the rendered page, something like this:



□	Hello World /hello-world	Offline	2016-02-19
☑	USA usa	Offline	2016-02-20
□	Templates _templates	Offline	2016-02-19

Country

USA

Population: 318,900,000
The U.S. is a country of 50 states covering a vast swath of North America, with Alaska in the extreme Northwest and Hawaii extending the nation’s presence into the Pacific Ocean. Major cities include New York, a global finance and culture center, and Washington, DC, the capital, both on the Atlantic Coast; Los Angeles, famed for filmmaking, on the Pacific Coast; and the Midwestern metropolis Chicago.

The Country Page Template

With our current solution, sadly, we would have to create a new page for every country we add. As this is not a very effective way of working with large data sets, we will create a page template that will automatically render all country content.

1. Select the Templates item located below the “Hello World” site in the content pane.
2. Click “New” and select “Page Template”.
3. Fill in the form as follows:

- Display Name: “Country”
- Supports: “Country” (selected from the list of content types)

4. If the blue Page Editor panel is not displayed on the right, click the button in the toolbar.
5. Select the “Hello Region” controller with the dropdown in the blue Page Editor panel.

6. Open the Inspection Panel (activated from the cog button in the toolbar).
7. Under the "Insert" tab, drag and drop a "Part" into the empty region where it says "Drop here".
8. Select the "country" part from the dropdown.
9. Click "Save draft" in the toolbar and close the tab.

Every "Country" content you create will now use this template by default.

Tip: The "Support" property is the key. A page template will support rendering of the content types specified here.

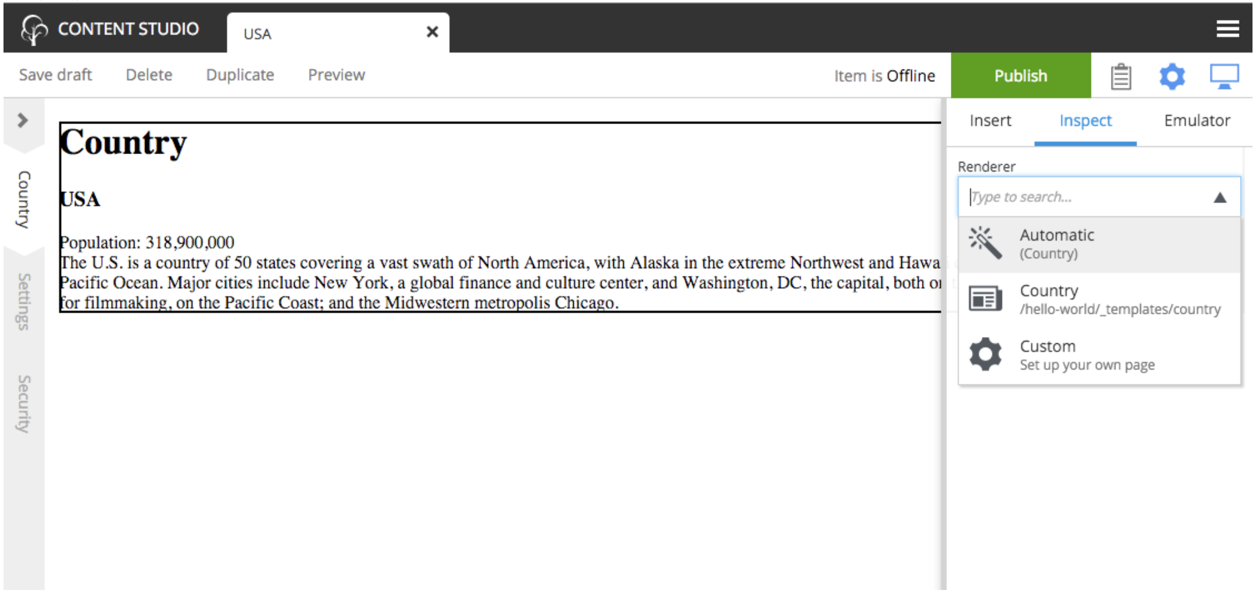
Try this out by creating a few new countries in your site. Be aware that every content you create will be a child of the content that was selected in the content pane, so make sure you select the "Hello World" site before clicking "New" in the toolbar. Or get in the habit of right-clicking the parent content and selecting "New" from the context menu. This way you will never accidentally create a content in the wrong place.

Extra task

Make your Favorite Country use the page template too!

You might remember that your favorite country was "hardcoded" - so let's change it to use templates as well.

1. In the Content pane, double click the country content to edit it.
2. Open the Inspection Panel and select the "Inspect" tab if it's not already selected.
3. You should see a label for "Rnederer" with "Custom" selected and a label for "Page controller" with "Hello Region" selected. If you see a label for "Part" instead then click on the page above the country name to select the page. Then click the "Inspect" tab. (See image below)
4. Now select "Automatic" from under the "Renderer" label in the "Inspect" tab.
5. Save draft and close the tab.



Country

USA

Population: 318,900,000
The U.S. is a country of 50 states covering a vast swath of North America, with Alaska in the extreme Northwest and Hawa
Pacific Ocean. Major cities include New York, a global finance and culture center, and Washington, DC, the capital, both o
for filmmaking, on the Pacific Coast; and the Midwestern metropolis Chicago.

You can select another *Page template* at any time, or even customize the presentation of a single content.

The Country List Part

Each country content can now be viewed on a page. But the site home page is still a bit empty. This section will have you alter the “hello” page controller and view files to list all of the country contents.

1. Edit the “hello” page controller `site/pages/hello/hello.js` and replace the file’s contents with the code below:

```
var thymeleaf = require('/lib/xp/thymeleaf'); // Import the thymeleaf render function
var contentLib = require('/lib/xp/content'); // Import the content service functions
var portal = require('/lib/xp/portal'); // Import the portal functions

// Handle the GET request
exports.get = function(req) {
    var model = {};

    var nearestSite = portal.getSite();

    // Get all the country contents (in the current site)
    var result = contentLib.query({
        start: 0,
        count: 100,
        contentTypes: [
            app.name + ':country'
        ],
        "query": "_path LIKE '/content" + nearestSite._path + "/*'"
    });

    var hits = result.hits;
    var countries = [];

    // Loop through the contents and extract the needed data
    for(var i = 0; i < hits.length; i++) {

        var country = {};
        country.name = hits[i].displayName;
        country.contentUrl = portal.pageUrl({
            id: hits[i]._id
        });
        countries.push(country);
    }

    // Add the country data to the model
    model.countries = countries;

    // Specify the view file to use
    var view = resolve('hello.html');

    // Compile HTML from the view with dynamic data from the model
    var body = thymeleaf.render(view, model);

    // Return the response object
    return {
        body: body
    }
};
```

2. Now edit the "hello" view file `site/pages/hello/hello.html` and replace its contents with the code below:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Hello world</title>
</head>
<body data-portal-component-type="page">
    <h1>Hello world</h1>
    <h2>Countries</h2>
    <ul>
        <li data-th-each="country : ${countries}">
            <strong>
                <a data-th-href="${country.contentUrl}" data-th-text="${country.name}"></a>
            </strong>
        </li>
    </ul>
</body>
</html>
```

3. If you didn't start XP in *Development mode* then redeploy the app from the command line with `./gradlew deploy`.

Each country that you created is now listed on the home page and the names are also links to the individual content pages.

Hello Geo World

Going back to your site, you will now see a list of the countries we have added. To make this even more exciting, we will add a City content type with geo-location and a *City list* part with configuration capabilities.

City content

The next steps will create a content type for adding cities with location coordinates.

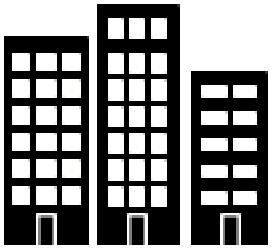
1. Create a folder called *city* inside the project's `site/content-types` folder.
2. Add the content type file below to your project. Because the contet type's folder is named "city" the file must be named "city.xml".

Listing 2.9: City content type - site/content-types/city/city.xml

```xml
<content-type>
  <display-name>City</display-name>
  <super-type>base:structured</super-type>
  <form>
    <input type="GeoPoint" name="location">
      <label>Location</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
    <input type="TextLine" name="population">
      <label>Population</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
  </form>
</content-type>
```

The file above defines a *content type* for cities with a required field for the location in latitude and longitude.

3. Copy the image below and save it in the same folder with the City content type. Name it "city.png".



City list part

We need a *part component* to display the city data. It will list the cities and show a Google map of each location.

1. Create a folder called *city-list* inside the project's `site/parts` folder.
2. Add the part descriptor file. It must be named city-list.xml.

Listing 2.10: City list part descriptor - site/parts/city-list/city-list.xml

```
<part>
  <display-name>City list</display-name>
  <config>
    <input type="ComboBox" name="mapType">
      <label>Map type</label>
      <occurrences minimum="0" maximum="1"/>
      <config>
        <option value="ROADMAP">ROADMAP</option>
        <option value="SATELLITE">SATELLITE</option>
        <option value="HYBRID">HYBRID</option>
        <option value="TERRAIN">TERRAIN</option>
      </config>
    </input>
    <input type="TextLine" name="zoom">
      <label>Zoom level 1-15</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
  </config>
</part>
```

The part descriptor above has a configuration similar to those found in content types.

3. Add the part controller file. It must be named city-list.js.

Listing 2.11: City list part controller - site/parts/city-list/city-list.js

```
var contentLib = require('/lib/xp/content'); // Import the content library functions
var portal = require('/lib/xp/portal'); // Import the portal functions
var thymeleaf = require('/lib/xp/thymeleaf'); // Import the Thymeleaf rendering function

// Handle the GET request
exports.get = function (req) {

    // Get the part configuration for the map
    var config = portal.getComponent().config;
    var zoom = parseInt(config.zoom) && config.zoom <= 15 && config.zoom >= 1 ? config.zoom : 10;
    var mapType = config.mapType || 'ROADMAP';

    // String that will be inserted to the head of the document
    var googleMaps = '<script src="http://maps.googleapis.com/maps/api/js"></script>';

    var countryPath = portal.getContent()._path;

    // Get all the country's cities
    var result = contentLib.query({
        start: 0,
        count: 100,
        contentTypes: [
            app.name + ':city'
        ],
        query: "_path LIKE '/content" + countryPath + "/*'",
        sort: "modifiedTime DESC"
    });

    var hits = result.hits;

    var cities = [];

    if (hits.length > 0) {
        googleMaps += '<script>function initialize() {';

        // Loop through the contents and extract the needed data
        for (var i = 0; i < hits.length; i++) {

            var city = {};
            city.name = hits[i].displayName;
            city.location = hits[i].data.location;
            city.population = hits[i].data.population ? 'Population: ' + hits[i].data.population : nu

            cities.push(city);

            if (city.location) {
                city.mapId = 'googleMap' + i;

                googleMaps += 'var center' + i + ' = new google.maps.LatLng(' + city.location + ');

                googleMaps += 'var mapProp = {center:center' + i + ', zoom:' + zoom +
                              ', mapTypeId:google.maps.MapTypeId.' + mapType + ', scrollwheel: false
                              'var map' + i + ' = new google.maps.Map(document.getElementById("google
                              'var marker = new google.maps.Marker({ position:center' + i + '}); mark
            }

        }

        googleMaps += '} google.maps.event.addDomListener(window, "load", initialize);</script>';
    }

    // Prepare the model object that will be passed to the view file
    var model = {
        cities: cities
    };
    // Specify the view file to use
    var view = resolve('city-list.html');

    // Return the response object
    return {
        body: thymeleaf.render(view, model),
        // Put the maps' javascript into the head of the document
        pageContributions: {
            headEnd: googleMaps
        }
    }
};
```

This controller uses *Page Contributions* to put the Google Maps JavaScript into the head of the document.

4. Add the part view file. It must be named city-list.html to match the “resolve” function in the controller.

Listing 2.12: City list part view - site/parts/city-list/city-list.html

```
<div class="cities" style="min-height:100px;">
    <h3>Cities</h3>
    <div class="city" data-th-each="city : ${cities}">
        <h3 data-th-text="${city.name}"></h3>
        <div data-th-if="${city.population}" data-th-text="${city.population}"></div>
        <div data-th-id="${city.mapId}" data-if="${city.mapId}" style="width:100%;height:300px;"></di
        <br/><br/>
    </div>
</div>
```

5. If you didn’t start XP in *Development mode* then build and deploy your project one final time with `./gradlew deploy`.

All of the project’s files are now complete. The rest of the steps will be performed in the Content Studio interface.

Create Cities

Now let’s make use of the new city content type and part component. First we need to add the “City list” part to the “Country” page template.

1. Edit the “Country” page template. Find it in the content pane below the templates 🏛
2. Open the Inspect Panel by clicking the cog button ⚙ in the toolbar.
3. Under the Insert tab, click and drag a *Part* 🧩 to the page region below the “country” part. (This may be a bit tricky because the “country” part is small.)
4. Select the “City list” part from the dropdown in the box.
5. Save and close the tab.

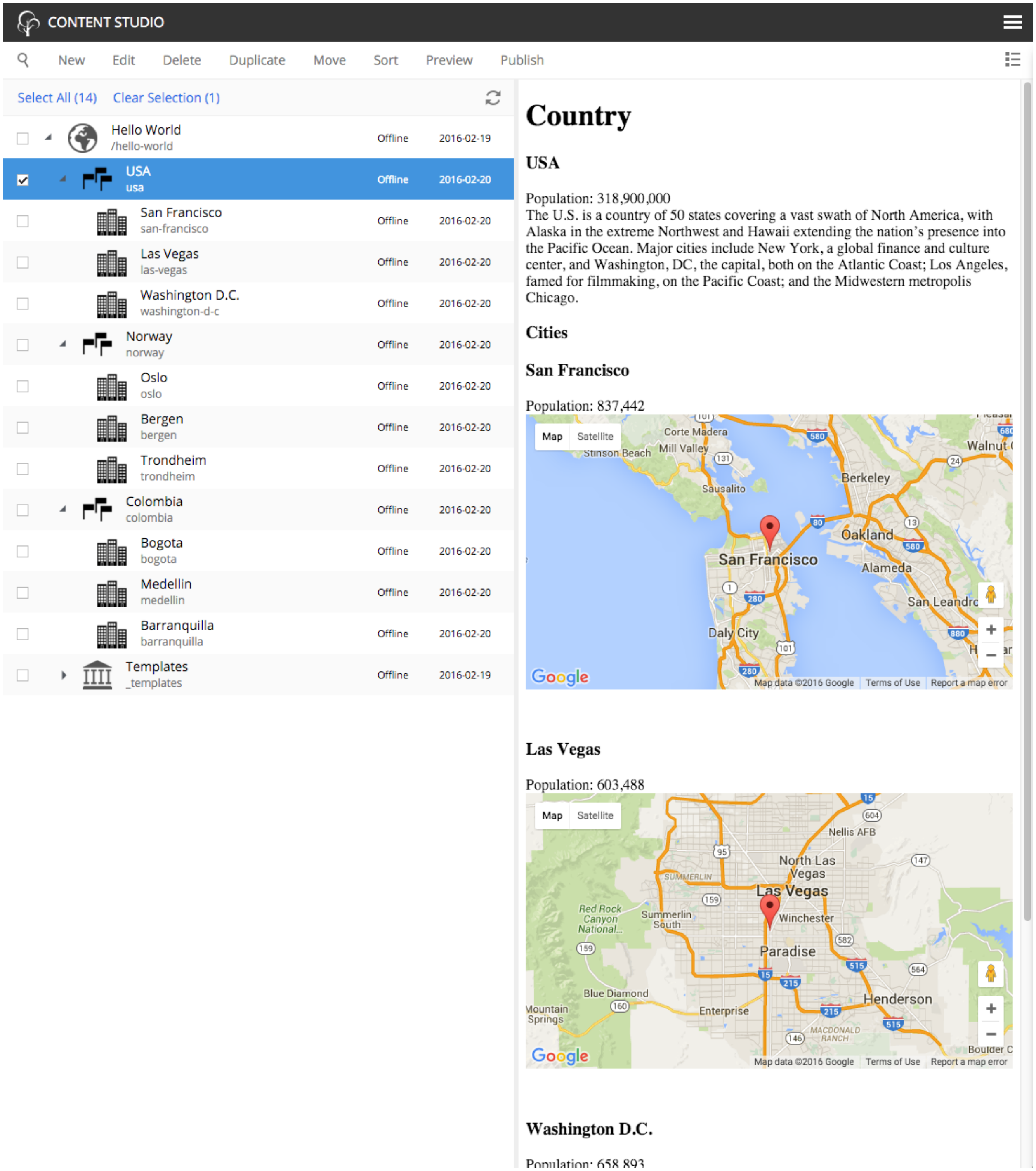
Now we need to create a few City contents under each country. (Sample data is available in the table below.)

1. From the content pane, select a country content that you created earlier. It is important that each city content is created under its country.
2. Right-click the country content and select “New”. The “Create content” dialogue will open.

3. Now select “City” from the list of content types.
4. Fill in the city name and location; the population is optional. The location format must be comma separated latitude and longitude with decimals.
5. Save draft.
6. Create several more city contents below each country content by repeating the previous steps. Sample data is provided in the table below.

Country	City	Lat,Long	Population
USA	San Francisco	37.7833,-122.4167	837,442
	Las Vegas	36.1215,-115.1739	603,488
	Washington D.C.	38.9047,-77.0164	658,893
Norway	Oslo	59.9100,10.7500	618,683
	Bergen	60.3894,5.3300	265,857
	Trondheim	63.4297,10.3933	178,021
Colombia	Bogota	4.5981,-74.0758	7,000,000
	Medellin	6.2308,-75.5906	2,440,000
	Barranquilla	10.9639,-74.7964	1,885,500

Each country page will now have a list of the cities you created with a Google map of the location. It should look something like this:



□	Hello World /hello-world	Offline	2016-02-19
☑	USA usa	Offline	2016-02-20
□	San Francisco san-francisco	Offline	2016-02-20
□	Las Vegas las-vegas	Offline	2016-02-20
□	Washington D.C. washington-d-c	Offline	2016-02-20
□	Norway norway	Offline	2016-02-20
□	Oslo oslo	Offline	2016-02-20
□	Bergen bergen	Offline	2016-02-20
□	Trondheim trondheim	Offline	2016-02-20
□	Colombia colombia	Offline	2016-02-20
□	Bogota bogota	Offline	2016-02-20
□	Medellin medellin	Offline	2016-02-20
□	Barranquilla barranquilla	Offline	2016-02-20
□	Templates _templates	Offline	2016-02-19

Country

USA

Population: 318,900,000
The U.S. is a country of 50 states covering a vast swath of North America, with Alaska in the extreme Northwest and Hawaii extending the nation's presence into the Pacific Ocean. Major cities include New York, a global finance and culture center, and Washington, DC, the capital, both on the Atlantic Coast; Los Angeles, famed for filmmaking, on the Pacific Coast; and the Midwestern metropolis Chicago.

Cities

San Francisco

Population: 837,442

Las Vegas

Population: 603,488

Washington D.C.

Population: 658,893

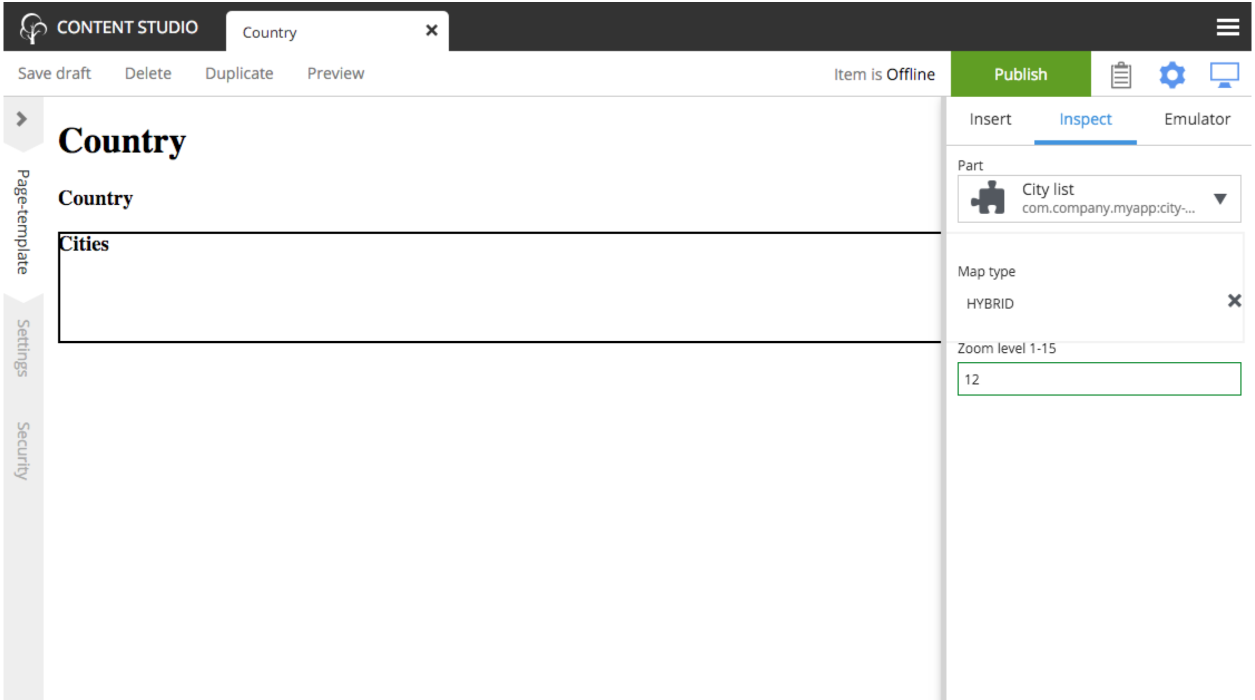
Configure City List

The *City list* part descriptor (site/parts/city-list/city-list.xml) has configuration inputs for the map type and zoom level. You can set the default values for these inputs by editing the *City list* part in the *Country* page template.

1. Open the *Country* page template for editing.

2. Open the Inspection Panel by clicking the cog button in the toolbar.
3. Click on the *City list* part in the Page Editor panel.
4. Under the "Inspect" tab of the Inspection Panel, set the Map type to "HYBRID" and Zoom level to 12.
5. Save draft and close the edit tab.

Now all of the countries will show the city maps with the new settings. You can override these defaults for any individual country by editing the Country content and changing its *City list* part configuration.



Part: City list com.company.myapp:city-...

Map type: HYBRID

Zoom level 1-15: 12

Go Online

Now that your "Hello World" is complete, it's time to publish.

1. Select the "Hello World" site in the content pane
2. Right-click and select "Publish" from the context menu, or click the "Publish" button in the toolbar
3. The Publishing Wizard will appear. Check the "Include children" checkbox
4. Verify that all your items are listed - click "Publish"!

When publishing content, all the selected items and changes are "cloned" from draft to the master branch (*Repository*).

You will always see the draft version of content in the preview window and the Page Editor of the Content Studio. If you have placed your site on root level, you can also see your live site at this url: `http://localhost:8080/portal/master/hello-world`.

Well done - you just created your first App for Enonic XP - The Enonic team congratulates you - we look forward to seeing all the brilliant things you will make and are always looking for feedback.

Some Pro Tips

Adding libraries to your project

Why re-invent the wheel for each app you make? Reusable code can be created in libraries and added to any XP project. Some essential tools can be found in the Util lib, Menu lib, RECAPTCHA and Landing page libraries.

Adding apps to your site

A website can be created from a single app. But a site's functionality can be extended by adding other pre-made apps. SEO Meta Fields, Disqus comments and Google Analytics are just a few of the many apps that can instantly add features to your site. See what's available at the Enonic Marketplace.

Handling Multiple projects

A **best practice** for working on multiple projects would involve keeping a separate XP_HOME folder for each project. The folder structure for such a setup would look something like this:

```
/Users/<name>/development
/Users/<name>/development/software/<xp-install-version>
/Users/<name>/development/xp-homes/<project-name>/home
/Users/<name>/development/projects/<project-name>/<project-source-files>
```

An actual implementation with projects called my-first-app and company-site would look like this:

```
/Users/mla/development/software/enonic-xp-6.3.1
/Users/mla/development/software/enonic-xp-6.4.0
/Users/mla/development/xp-homes/my-first-app/home
/Users/mla/development/xp-homes/company-site/home
/Users/mla/development/projects/my-first-app/...
/Users/mla/development/projects/company-site/...
```

This allows you to have one Enonic XP installation for each version and as many different XP_HOME folders as you need for your projects. When switching from one project to another, you only have to change the XP_HOME environment variable and then restart the installation of the Enonic XP version that the project was created for.

Check this Enonic Labs article for a more in-depth process. It also includes some bash scripting that will help with setting and changing $XP_HOME and starting and stopping XP.

Logging JSON objects

While developing an app, it can be helpful to see the structure of objects returned by library functions. The best way to do this is to set up a utilities JavaScript file in the project lib folder. Add the following function to the utilities file:

```
site/lib/utilities.js
```

```
exports.log = function (data) {
  log.info('Utilities log %s', JSON.stringify(data, null, 4));
};
```

Call the log function in any controller like the example below and then check the log after refreshing the page.

```
var util = require('utilities');

var content = portal.getContent();
util.log(content);
```

This logging function and many other useful functions are included in the Util library.

Continuous build with Dev Mode or Gradle

It can be quite time consuming to frequently use the terminal to redeploy an app during development. Starting Enonic XP in dev-mode will make most changes to your app code visible immediately on localhost. See the *Development mode* page for more information about its capabilities and limitations.

Another option is to use `./gradlew -t deploy` in the terminal from the project root. This will automatically build and redeploy your app every time changes to a file are detected. This method is slower than dev-mode, but it handles some situations that dev-mode doesn't. Gradle 2.7 or newer is required.

Both dev-mode and Gradle continuous-mode require the $XP_HOME environment variable to be set in the terminal window.

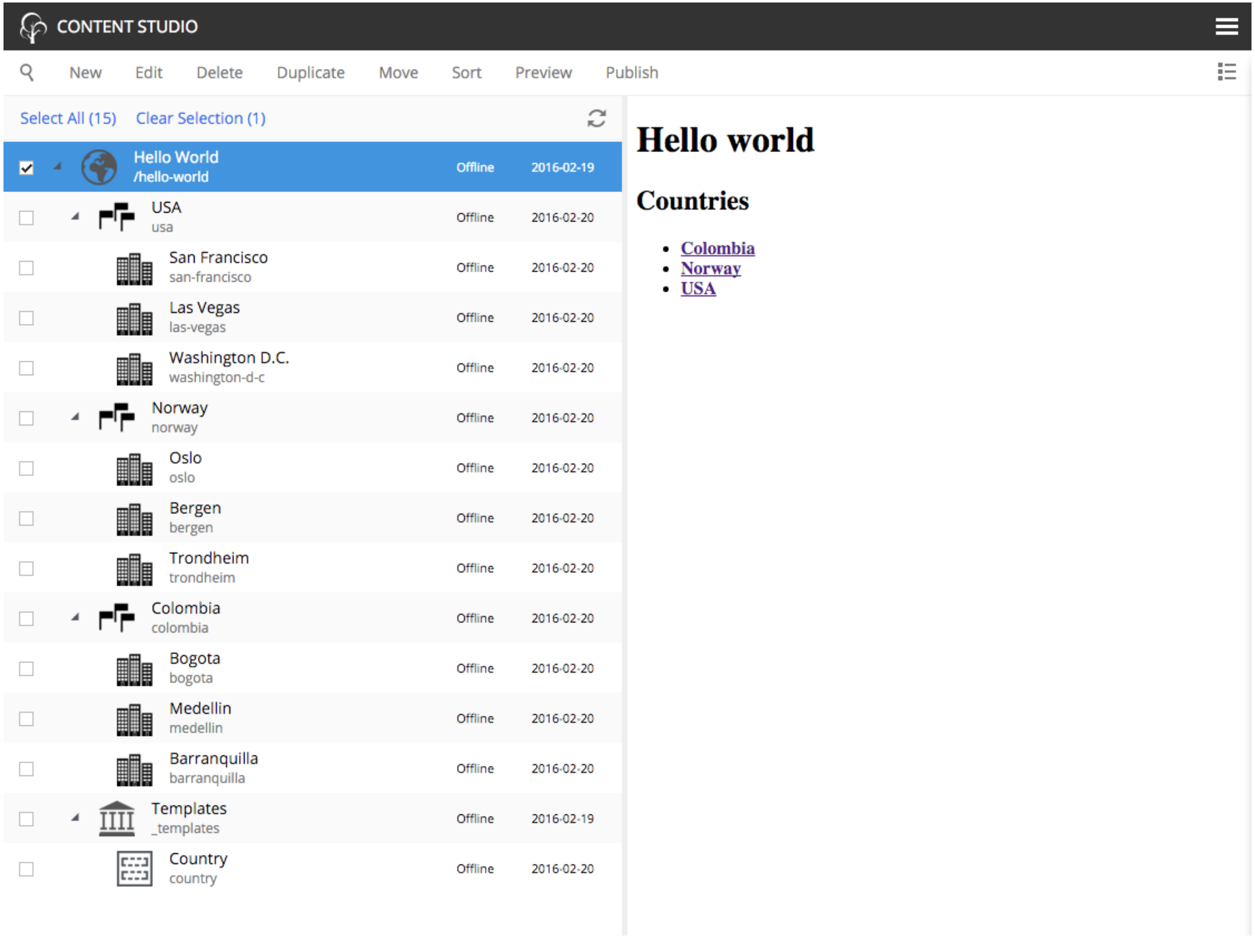
Next Steps

This tutorial only covered the basics of app development. Explore the documentation for more in-depth coverage.

Check our GitHub page for examples of more advanced apps. The Superhero blog app is also a good place to see more advanced code. The Google Analytics app demonstrates extending the Content Studio interface

Need help? Ask questions on our forum or answer questions from others.

The screenshot below shows the Content Studio interface with content on the left and a site preview on the right.



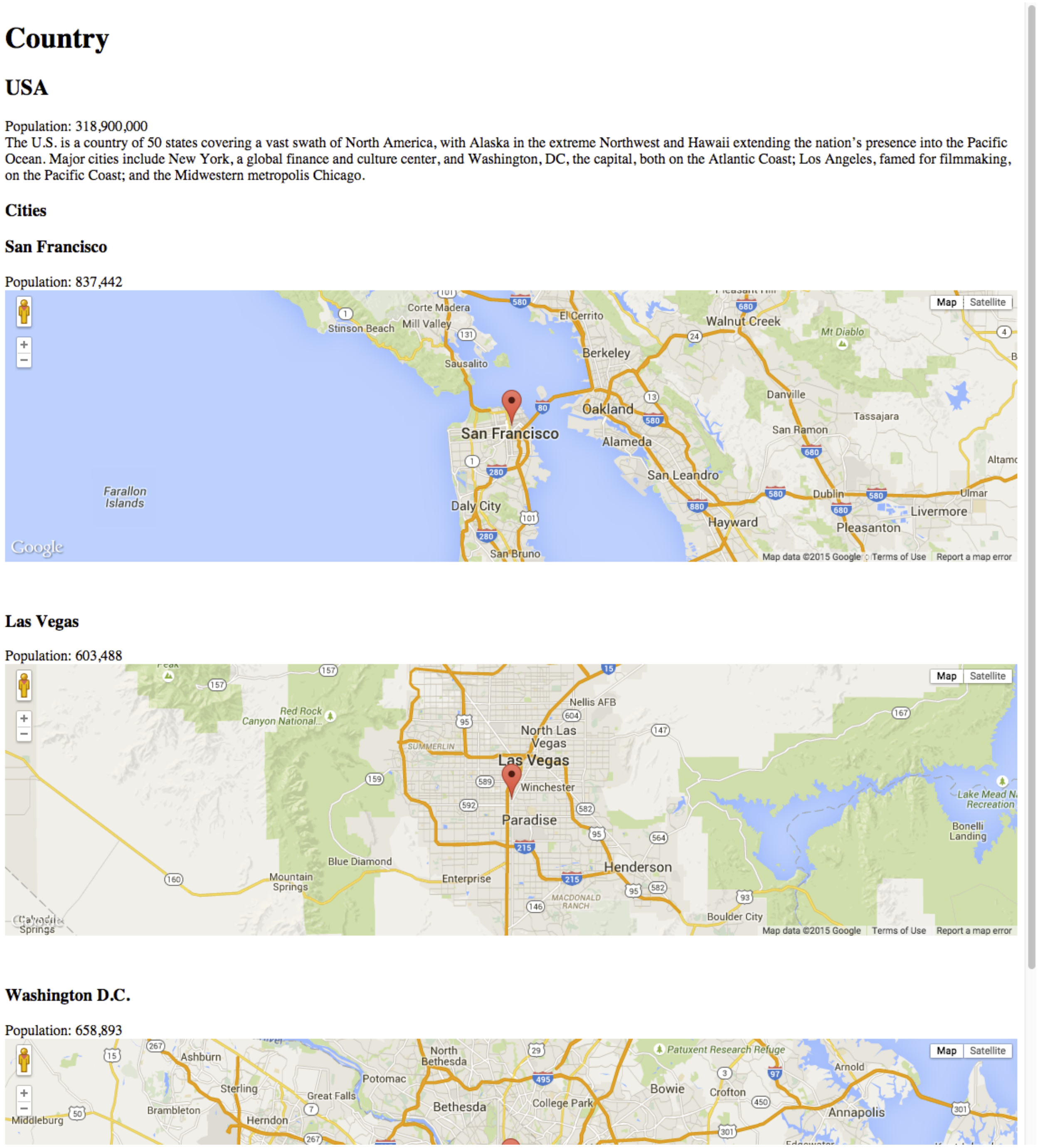
☑	Hello World	/hello-world	Offline	2016-02-19
□	USA	usa	Offline	2016-02-20
□	San Francisco	san-francisco	Offline	2016-02-20
□	Las Vegas	las-vegas	Offline	2016-02-20
□	Washington D.C.	washington-d-c	Offline	2016-02-20
□	Norway	norway	Offline	2016-02-20
□	Oslo	oslo	Offline	2016-02-20
□	Bergen	bergen	Offline	2016-02-20
□	Trondheim	trondheim	Offline	2016-02-20
□	Colombia	colombia	Offline	2016-02-20
□	Bogota	bogota	Offline	2016-02-20
□	Medellin	medellin	Offline	2016-02-20
□	Barranquilla	barranquilla	Offline	2016-02-20
□	Templates	_templates	Offline	2016-02-19
□	Country	country	Offline	2016-02-20

Hello world

Countries

- Colombia
- Norway
- USA

The image below is what a page of the site will look like when finished.



Country

USA

Population: 318,900,000
The U.S. is a country of 50 states covering a vast swath of North America, with Alaska in the extreme Northwest and Hawaii extending the nation’s presence into the Pacific Ocean. Major cities include New York, a global finance and culture center, and Washington, DC, the capital, both on the Atlantic Coast; Los Angeles, famed for filmmaking, on the Pacific Coast; and the Midwestern metropolis Chicago.

Cities

San Francisco

Population: 837,442

Las Vegas

Population: 603,488

Washington D.C.

Population: 658,893

Build a Custom Selector

This guide will lead you through the required steps to build an input of type Custom Selector.

- *Create a content type*
- *Create a service*
- *Response format*

- *Sample service*
- *Integration with Spotify API*

Create a content type

- Create a folder called “my-custom-selector” inside the “site/content-types” folder of your project.
- In that folder create a configuration schema for the “my-custom-selector” content type.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<content-type>
  <display-name>Custom Selector</display-name>
  <super-type>base:structured</super-type>
  <form>
    <input name="my-custom-selector" type="CustomSelector">
      <label>My Custom Selector</label>
      <occurrences minimum="0" maximum="0"/>
      <config>
        <service>my-custom-selector-service</service>
      </config>
    </input>
  </form>
</content-type>
```

Create a service (or refer to a service in another app)

- Create a folder called “my-custom-selector-service” (folder name must match the one specified in the config schema) inside the “resources/services” folder of your project.
- In that folder create a javascript service file called “my-custom-selector-service.js” (again, the name must match the config schema).
- Create GET handler method in this service file and make sure it returns JSON in the proper format.

Tip: You can also refer to service file in another application (for example, *com.myapplication.app:myservice*) instead of adding one to your application.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<content-type>
  <display-name>Custom Selector</display-name>
  <super-type>base:structured</super-type>
  <form>
    <input name="my-custom-selector" type="CustomSelector">
      <label>My Custom Selector</label>
      <occurrences minimum="0" maximum="0"/>
      <config>
        <service>com.myapplication.app:my-custom-selector-service</service>
      </config>
    </input>
  </form>
</content-type>
```

Response format

Format of JSON response from the service:

id Unique Id of the option

displayName Option title

description (optional) Detailed description

iconUrl (optional) Path to the thumbnail image file

icon (optional) Inline image content (for example, SVG)

Example of a simple service file

Below is a simple service file that returns two items in the result set, one with external thumbnail image, and another one with inline SVG markup:

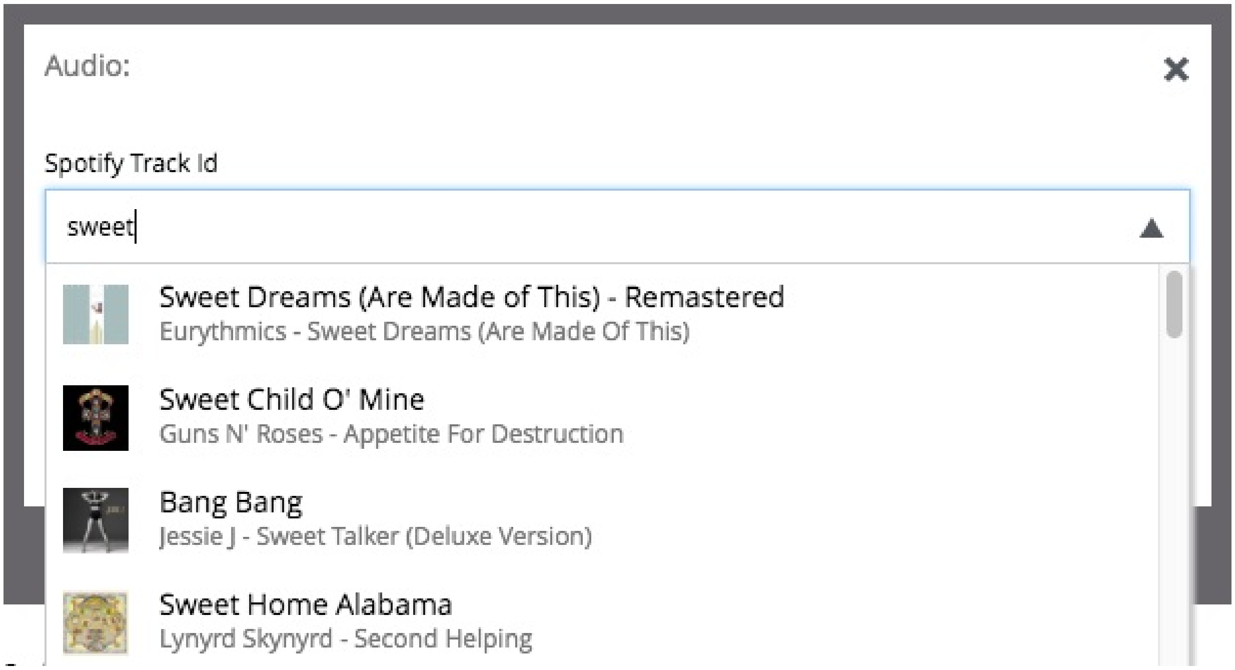
```
var portalLib = require('/lib/xp/portal');

exports.get = handleGet;

function handleGet(req) {

    var params = parseparams(req.params);

    var body = createresults(getItems(), params);

    return {
        contentType: 'application/json',
        body: body
    }
}

function getItems() {
    return [{
        id: 1,
        displayName: "Option number 1",
        description: "External SVG file is used as icon",
        iconUrl: portalLib.assetUrl({path: 'images/number_1.svg'}),
        icon: null
    }, {
        id: 2,
        displayName: "Option number 2",
        description: "Inline SVG markup is used as icon",
        iconUrl: null,
        icon: {
            data: '<svg version="1.1" xmlns="http://www.w3.org/2000/svg" width="32" height="32" viewB
            type: "image/svg+xml"
        }
    }];
}

function parseparams(params) {

    var query = params['query'],
        ids, start, count;
    try {
        ids = JSON.parse(params['ids']) || []
    } catch (e) {
        log.warning('Invalid parameter ids: %s, using []', params['ids']);
        ids = [];
    }

    try {
        start = Math.max(parseInt(params['start']) || 0, 0);
    } catch (e) {
        log.warning('Invalid parameter start: %s, using 0', params['start']);
        start = 0;
    }

    try {
        count = Math.max(parseInt(params['count']) || 15, 0);
    } catch (e) {
        log.warning('Invalid parameter count: %s, using 15', params['count']);
        count = 15;
    }

    return {
        query: query,
        ids: ids,
        start: start,
        end: start + count,
        count: count
    }
}

function createresults(items, params, total) {

    var body = {};

    log.info('Creating results with params: %s', params);

    var hitCount = 0, include;
    body.hits = items.sort(function (hit1, hit2) {
        if (!hit1 || !hit2) {
            return !!hit1 ? 1 : -1;
        }
        return hit1.displayName.localeCompare(hit2.displayName);
    }).filter(function (hit) {
        include = true;

        if (!!params.ids && params.ids.length > 0) {
            include = params.ids.some(function (id) {
                return id == hit.id;
            });
        } else if (!!params.query && params.query.trim().length > 0) {
            var qRegex = new RegExp(params.query, 'i');
            include = qRegex.test(hit.displayName) || qRegex.test(hit.description) || qRegex.test(hit
        }

        if (include) {
            hitCount++;
        }
        return include && hitCount > params.start && hitCount <= params.end;
    });
    body.count = Math.min(params.count, body.hits.length);
    body.total = params.query ? hitCount : (total || items.length);

    return body;
}
```

Integration with Spotify API

And here’s a bit more advanced version of the service file that fetches song names from the Spotify API:

```
var portalLib = require('/lib/xp/portal');
var httpClient = require('/lib/xp/http-client');
var cacheLib = require('/lib/xp/cache');

var trackIdCache = cacheLib.newCache({
    size: 100
});

var searchQueriesCache = cacheLib.newCache({
    size: 100,
    expire: 60 * 10
});

exports.get = handleGet;

function handleGet(req) {

    var params = req.params;
    var ids;
    try {
        ids = JSON.parse(params.ids) || []
    } catch (e) {
        ids = [];
    }

    var tracks;
    if (ids.length > 0) {
        tracks = getTracks(ids);
    } else {
        tracks = searchTracks(params.query, params.start || 0, params.count || 10);
    }

    return {
        contentType: 'application/json',
        body: tracks
    }
}

function getTracks(ids) {
    var tracks = [];

    for (var i = 0; i < ids.length; i++) {
        var id = ids[i];

        var track = trackIdCache.get(id, function () {
            var spotifyResponse = fetchSpotifyTrackById(id);
            return spotifyResponse ? parseTrackResults(spotifyResponse) : null;
        });

        if (track) {
            tracks.push(track);
        }
    }

    return {
        count: tracks.length,
        total: tracks.length,
        hits: tracks
    };
}

function searchTracks(text, start, count) {
    text = (text || '').trim();
    if (!text) {
        return {
            count: 0,
            total: 0,
            hits: []
        };
    }

    return searchQueriesCache.get(searchKey(text, start, count), function () {
        var spotifyResponse = searchSpotifyTracks(text, start, count);
        return parseSearchResults(spotifyResponse);
    });
}

function searchKey(text, start, count) {
    return start + '-' + count + '-' + text;
}

function fetchSpotifyTrackById(id) {
    log.info('Fetching Spotify tack by id: ' + id);
    try {
        var response = httpClient.request({
            url: 'https://api.spotify.com/v1/tracks/',
            method: 'GET',
            contentType: 'application/json',
            connectTimeout: 5000,
            readTimeout: 10000,
            params: {
                'ids': id
            }
        });
        if (response.status === 200) {
            return JSON.parse(response.body);
        }

    } catch (e) {
        log.error('Could not retrieve spotify track', e);
    }

    return null;
}
```

```
function searchSpotifyTracks(text, start, count) {
    if (!text) {
        return noTracks();
    }

    log.info('Querying Spotify: ' + start + ' + ' + count + ' "' + text + '"');
    try {
        if (text.length > 1) {
            text += '*';
        }
        var response = httpClient.request({
            url: 'https://api.spotify.com/v1/search',
            method: 'GET',
            contentType: 'application/json',
            connectTimeout: 5000,
            readTimeout: 10000,
            params: {
                'q': text,
                'type': 'track,artist',
                'limit': count,
                'offset': start
            }
        });

        if (response.status === 200) {
            return JSON.parse(response.body);
        }

    } catch (e) {
        log.error('Could not search for spotify tracks: ', e);
    }

    return noTracks();
}

function noTracks() {
    return {
        "tracks": {
            "items": [],
            "total": 0
        }
    };
}

function parseSearchResults(resp) {
    var options = [];
    var tracks = resp.tracks.items, i, option, track;
    for (i = 0; i < tracks.length; i++) {
        track = tracks[i];
        option = trackIdCache.get(track.id, function () {
            return parseTrack(track);
        });
        options.push(option);
    }

    return {
        count: resp.tracks.items.length,
        total: resp.tracks.total,
        hits: options
    };
}

function parseTrackResults(resp) {
    var tracks = resp.tracks;
    if (tracks && tracks.length === 0) {
        return null;
    }
    return parseTrack(tracks[0]);
}

function parseTrack(spotifyTrack) {
    var option = {};
    option.id = spotifyTrack.id;
    option.displayName = spotifyTrack.name;

    var artist = spotifyTrack.artists && spotifyTrack.artists.length > 0 ? spotifyTrack.artists[0].na
    var album = spotifyTrack.album && spotifyTrack.album.name ? spotifyTrack.album.name : '';
    option.description = artist + (album ? ' - ' + album : '');

    if (spotifyTrack.album && spotifyTrack.album.images && spotifyTrack.album.images.length > 0) {
        option.iconUrl = spotifyTrack.album.images[0].url;
    } else {
        option.iconUrl = defaultIcon();
    }

    return option;
}

function defaultIcon() {
    return portalLib.assetUrl({path: 'img/Spotify_logo_without_text.svg'});
}
```



Audio:

Spotify Track Id

sweet

Sweet Dreams (Are Made of This) - Remastered
Eurythmics - Sweet Dreams (Are Made Of This)

Sweet Child O' Mine
Guns N' Roses - Appetite For Destruction

Bang Bang
Jessie J - Sweet Talker (Deluxe Version)

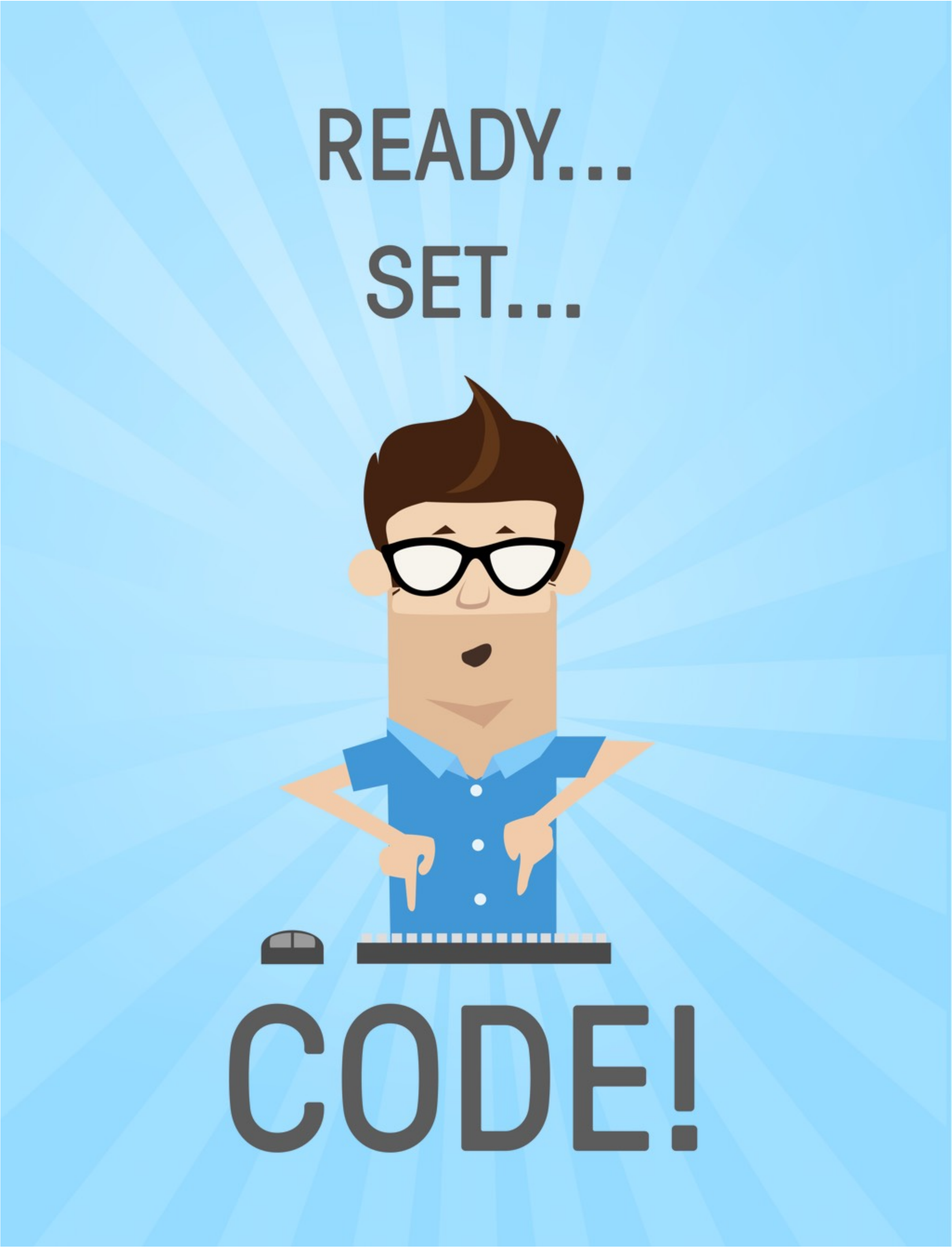
Sweet Home Alabama
Lynyrd Skynyrd - Second Helping

CHAPTER 3

Developer Guide

We <3 Developers!

We're thrilled to see you here - If you're a first timer to Enonic XP, we recommend starting on *My First App* and the *Tutorials* section - or if you're more familiar with XP, how about drilling into the *Node Domain* chapter. If you're actually looking for APIs - you'll find them over here *API and Reference Guide*.


READY...
SET...
CODE!

Applications

Like anything that calls itself an operating system - Enonic XP has Applications. The applications can be installed and run on a single server, or an entire cluster.

It is important to note that applications are not limited to being “Web Applications” - they can be running server jobs, provide internal API’s to other applications, offer HTTP services, include custom Admin Tools, extend other tools or applications, or be used to build sites.

Applications may even carry data, for instance, to initialize a repository or populate a new site.

Life Cycle

Applications have the following life-cycle:

- installed
- started
- stopped
- uninstalled

When installing applications in an XP cluster - the application is uploaded and stored in the system repository - and then started on all nodes. If an application contains an initialization script, this script will only be executed on the master node - running only once.

Composition

An application file is typically a JAR (.jar). This is short for Java Archive. Enonic XP is built on top of Java and the powerful OSGi framework, so developers with special requirements may utilize capabilities such as exposing and consuming services from other applications. However, since Enonic XP is designed to run *Serverside JavaScript*, most *Projects* will be completely free of Java.

To speed up development and enable a high degree of re-use, applications can be composed of *Libraries* in addition to your own code. Libraries can be built almost like you create applications. Libraries (and applications) are shared through Maven repositories. An example is https://repo.enonic.com. Anyone may configure and run their own repository - for internal as well as external use.

For your amusement, we can also tell you that Enonic XP itself is composed from more than 50 different applications - making the platform extremely modular.

Other Resources

To learn more about applications and how they are built - continue reading the *Developer Guide*, but pay special attention to the following chapters:

- *Projects*
- *Libraries*

Libraries

Enonic XP provides the concept of libraries in order to speed up development and re-use of functionality and code. Technically, libraries are very similar to *Applications*, but the main difference is that a library cannot be installed and started by itself.

So, a library may consist of all the same objects you find in applications - such as *Assets*, *Content Types* and *HTTP Controllers* - things you may need in an application.

Finding Libraries

A number of standard and 3rd party libraries are available with the core XP release, check out: *Javascript Libraries*. You will also find a wide range of libraries on the Enonic Market - https://enonic.com/market/libraries

Adding libraries

Libraries are added to your project by simply referring to them in your build script. Read more about this on our *Projects* documentation.

Best practice

If you wonder when/how you should create a library, here are some guidelines

- Strong cohesion: Keep the components in a library together only if they are strongly related. Split them up in multiple libs if they are not.
- Weak coupling: A library should avoid having dependencies. In practice, this will not always be possible, but apart from the XP APIs, other dependencies should raise an alarm, and only be included after careful analysis showing there is no other way.
- Use names that are self-explanatory and follow the java naming conventions - for example com.company.lib.mylib.

Projects

This section provides an overview of how to setup, build and deploy new projects for applications and libraries.

Project Initialization

The fastest way to get started with any XP project is to use a starter project. The starter-vanilla project on GitHub - https://github.com/enonic/starter-vanilla is often used.

To get going, we recommend using the *init-project* script that is a part of the Enonic XP installation.

In the terminal, move to the XP installation's toolbox folder. Copy/paste the command below and specify an empty folder for -d and an appropriate app name for -n. Then execute the command to get your very own project initialized.

```
toolbox.sh init-project -d <projectFolder> -n com.mycompany.myApp -r enonic/starter-vanilla
```

The init-project command simply clones the entire Git project (to the local folder that was specified with -d), then removes the Git references. It also updates your build script files by adding the specified app name (-n) to the project.

Once this is done, you must clean up and adapt the code to your own requirements.

You may, in principle, apply this command to any standard XP application or library project!

Project structure

To build applications with Enonic XP, you will typically setup a project. The fastest way to do this is using the init-project feature included in the Enonic XP toolbox utility.

The project structure is a similar to Maven projects for those who are familiar with that.

Below is a sample project folder structure - all items are folders, except for `site.xml` and `build.gradle`:

```
my-first-app/
  build.gradle
  src/
    main/
      java/
      resources/
        admin/
          widgets/
          tools/
        application.svg
        application.xml
        assets/
        lib/
        services/
        site/
          content-types/
          error/
          filters/
          i18n/
          layouts/
          mixins/
          pages/
          parts/
          site.xml
        views/
```

Every file and folder has a specific function and meaning.

build.gradle Gradle script for building the application or library. This file describes the actual build process.

src/main/java/ Optional folder where you place any java code that might be included in the project - following traditional Maven style development.

src/main/resources/ This is where all non-java code is placed, and thus where you will typically be working with your XP projects. All folders described below are relative to this folder

admin/tools This is where you place code for admin tools. Tools are administrative user interfaces (apps) running in their own separate browser tab. Create tools if you need a back-office utility to manage your applications or similar.

admin/widgets Widgets are essentially user interface components that can be embedded within selected tools. I.e. you can create a widget that extends the Content Studio detail panel.

application.svg Application icon in SVG format.

application.xml The XML file contains basic information for the application. Currently a description.

```xml
<application>
  <description>Application description goes here</description>
</application>
```

assets/ Public folder for external css, javascript and static images etc. etc.

lib/ This is the last place the global `require` JavaScript-function looks, so it is a good place to put default JavaScript files here.

services/ Services are a special type of http controller that will be mounted on a fixed url pattern that looks like this: _/service/<myapp>/<myservice>. You may use services like any other JavaScript controller in the system.

site/site.xml The `site.xml` file contains basic information for a site created with the application. Settings for the application can be defined in the `config` element and the values for these settings can be updated using the Content Studio tool.

```xml
<site>
  <config>
    <input type="TextLine" name="company">
      <label>Company</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
    <input type="TextArea" name="description">
      <label>Description</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
  </config>
</site>
```

site/content-types/ Content schemas are placed here. Used to create structured content (see *Content Types*).

site/error/ Create custom http error pages by placing an error controller in this directory (see *Error Handling*).

site/filters/ This is where generic http response filters are placed. Filters can be used for post processing any given request - also across applications added to a site. A common use case is adding script tags to pages - but possibilities are virtually endless.

site/i18n/ This folder will contain application localization files (i18n is short for Internationalization). Files placed in this folder must follow Java's standard property file format, one file for each language. Here is an example: https://docs.oracle.com/javase/tutorial/i18n/resbundle/propfile.html

site/mixins/ Mixin schema-types are placed here. A mixin can be used to add common fields to multiple content-types or other schemas (see *Mixins*).

site/pages/ Page controllers are placed here. They will be used to render pages and page templates (see *Page*).

site/parts/ Part controllers should be placed here. Parts are dynamically configurable components that can be placed on pages (see *Part*).

site/layouts/ Layout controllers should be placed here. Layouts are similar to parts, but in addition have one or more regions. Regions enable placement of other components inside the layout. (see *Layout*).

views/

Views are any kind of files that are used for rendering. The folder is optional, as view files can be placed anywhere you want, just keep in mind what path to use when resolving them (see *Views*).

Build script

By default, Enonic uses Gradle as the main build tool. This is a highly flexible Java-based utility that builds on the popular Maven project tools and code repository structures. Enonic provides a Gradle plugin that greatly simplifies the build process. If you used the starter-vanilla project to initialize your project, you will have all the basic tools you need to get going.

Running a build

If you have not installed Gradle, the fastest way to get going is to execute the gradle wrapper script.

Move into your project root folder and execute the following command:

OSX/Linux:

```
./gradlew build
```

Windows:

```
gradlew.bat build
```

The gradle wrapper will download all necessary files to run gradle and produce the project artifacts. These will typically be placed in the projects build/libs/ folder.

gradle.properties

Your project should contain a `gradle.properties` file. Set `xpVersion` to the version of Enonic XP you are working with, and look over the other settings to make sure they are correct.

build.gradle

The `build.gradle` file defines all the dependencies to other libraries.

There are three standard scopes (keywords) used in the dependency list

- Compile (default gradle scope, compiles library and adds it to class path - standard for pure Java libraries)
- Include (XP custom scope that merges the /src/main/resources folder in the library with your project - any code in your project overwrites the library files)
- Webjar (Extracts the content of the specified Webjar - http://www.webjars.org/ - placing it into the assets folder, using the version number as root folder)

gradle deploy

To have Gradle automatically deploy new applications to your XP installation, you have to specify an environment variable that tells Gradle where to place the artifact (application file).

OSX/Linux:

```
export XP_HOME=/path/to/xp-installation/home
```

Windows:

```
set XP_HOME=c:\path\to\xp-installation\home
```

With $XP_HOME set, run the following command to build and deploy the file

OSX/Linux:

```
./gradlew deploy
```

Windows:

```
gradlew.bat deploy
```

Once completed, your XP installation will detect, install and start the files.

Installing an application

There are several ways to install applications

- Uploading directly from the "Applications" admin tool - this will install and start the application in the entire cluster
- Use the *Toolbox CLI* command line utility - this will install and start the application in the entire cluster.
- And finally, the developer way - copying the application JAR file to the `$XP_HOME/deploy` folder - this will install and start the application on the local node (typically used by developers)

Once an application is placed in this folder, it will be picked up, installed and started by the local instance. If the application is removed it will be stopped and uninstalled.

OSX/Linux command line to copy the artifact to the deploy folder:

```
cp build/libs/[artifact].jar $XP_HOME/deploy/.
```

For your convenience - we have simplified this process by adding a `deploy` task to your build. Instead of manually copying to the deploy folder, you can simply execute `gradle deploy`:

```
./gradlew deploy
```

For the deploy command to work, you have to set the `XP_HOME` environment variable (in your shell) to your actual Enonic XP home directory.

Run the following command to set the XP_HOME variable

OSX/Linux:

```
export XP_HOME=/path/to/xp-installation/home
```

Windows:

```
set XP_HOME=c:\path\to\xp-installation\home
```

Continuous Deploy

To continuously build and deploy your application on changes, you can use Gradle continuous mode. This will watch for changes and run the specified task when something changes. To use this with the `deploy` task, you can run the following command:

```
./gradlew -t deploy
```

This will deploy and reload the application on the server when something changes in your project. The continuous deployment mode is most useful when coding Java, or other changes that require a full compile and re-deploy.

For the instant updates of JavaScript code without re-deploying, check out *Development mode*.

Sample library

In this example, we will create support for redirection of URLs. For this, we need a content-type, and a simple JavaScript file.

The content-type, let’s just call it url, defines the URLs that the code may redirect to. So in the `content-types` directory, add a new directory and name it `url`. Then, in this directory, create the content type:

Listing 3.1: src/main/resources/site/content-types/url/url.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<content-type>
  <display-name>URL</display-name>
  <content-display-name-script>$('url')</content-display-name-script>
  <super-type>base:structured</super-type>
  <form>
    <input type="TextLine" name="url">
      <label>URL</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
  </form>
</content-type>
```

To make it easier to notice when creating a new content in Enonic XP, add this icon, `url.png` in the same directory:

Now, we need the JavaScript. Since we are talking about a redirect here, the script must be placed on a page. So, in the `pages` directory, we add a folder: `url-redirects` . In this, we need the page descriptor:

Listing 3.2: src/main/resources/site/pages/url-redirect/url-redirect.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<page>
    <display-name>URL redirect</display-name>
    <config/>
</page>
```

Then, we add the code that does the actual redirect:

Listing 3.3: src/main/resources/site/pages/url-redirect/url-redirect.js

```javascript
var portal = require('/lib/xp/portal');

// Handle GET request
exports.get = handleGet;

function handleGet() {
    var result = portal.getContent();
    var url = result.data.url;
    var response = {};

    if (url) {
        response.redirect = url;
    }
    else {
        response.body = 'No URL configured.'
    }

    return response;
}
```

This library can now be included in any app where you might want redirect functionality, or in other libs that can build more advanced functions based on this simple example.

Development mode

You can start the server in dev-mode to speed up the development process. When using this mode, you only need to deploy your code once - or when certain situations arise (see below).

Start server using dev-mode:

```
$ $XP_INSTALL/bin/server.sh dev
```

First time you will need to deploy the code (app, lib, etc.) using the `deploy` task (see *Installing an application*):

```
$ ./gradlew deploy
```

After that, you do not need to redeploy your application except...

- when modifying Java code.
- when deleting a page, part or layout component.
- when deleting a content-type, mixin or relationship-type.
- when changing source directories.
- if you have source-transformation tools (typescript, less, sass).

> **Warning:** Do not use dev-mode in production environments. It takes a little more time to render pages and caches are sometimes disabled.

Schemas

As the low level storage of Enonic XP is entirely schema-less, we have created a high-level schema concept that is used to configure many of the forms you see in the Enonic XP Admin.

Schemas are made up of the following main concepts:

- A rich set of widgets called *Input Types*
- Rich *Forms* by combining Input Types and layouts
- Horizontal inheritance across forms through the use of *Mixins*

Input Types

Input types are specified by XML snippets and used in combinations to build forms. An input type has both a front-end and a back-end. Each input type will return a property with a specific value type.

The following XML configuration is common for all input types:

```
<input name="name" type="type-name">
  <label>Some label</label>
  <occurrences minimum="0" maximum="1"/>
  <help-text>Use me to provide some extra info for the input</help-text>
  <config/>
</input>
```

@name The `name` attribute is the technical name used in templates and result sets to refer to this value.

@type The type refers to one of the many input types which are explained below.

label The label text will become the label for the input field in the editable form of the admin console.

help-text This optional text will be shown next to the input field and can be used for explanation of the field's purpose.

occurrences Detailed definition of how many times this field may be repeated inside one content. Set `minimum` to zero for fields that are not required, and `maximum` to zero for fields that have no restriction on the number of values. This element is optional, if omitted the default will be `minimum="0"` and `maximum="1"`.

config Optional configuration that is used by some of the input-types. The config consists of elements with optional attributes. Each element/attribute name with dashes is automatically camel-cased (`relationship-type` -> `relationshipType`).

default Optional element that contains a default value for the particular input type. Currently it is only supported by some input types. See below for value formats for each input type. Invalid values for a given input type will be ignored.

AttachmentUploader

This field enables uploading of one or more files that will be stored as attachments to the current node/content. This is different from media content where each media is a separate node that can be linked to.

```
<input type="AttachmentUploader" name="myname">
  <label>My Label</label>
  <occurrences minimum="0" maximum="0"/>
</input>
```

CheckBox

A checkbox field has a value if it is checked, or no value if it is not checked. Therefore, the only values for occurrences that makes sense is a minimum of zero and a maximum of one.

```
<input name="name" type="CheckBox">
  <label>Required</label>
  <occurrences minimum="0" maximum="1"/>
  <default>checked</default>
  <config>
    <alignment>right</alignment>
  </config>
</input>
```

default This element specifies the default value. Set it to `checked` to check it, otherwise it will be left unchecked.

config

alignment This config setting can be used to configure checkbox position in relation to its label. Default alignment is to the left of the label. Supported values are: “left”, “right”, “top”, “bottom”.

ComboBox

A ComboBox needs a list of options.

```
<input name="name" type="ComboBox">
  <label>Required</label>
  <occurrences minimum="1" maximum="1"/>
  <config>
    <option value="one">Option One</option>
    <option value="two">Option Two</option>
  </config>
  <default>one</default>
</input>
```

option This element defines the option label. The `value` attribute defines the actual value to set when this option is selected. Multiple `option` settings are ordered.

default This element specifies the default option for the combo box. It should be equal to one of the `option` values.

ContentSelector

References to other content are specified by this input type.

```
<input name="name" type="ContentSelector">
  <label>Cited In</label>
  <occurrences minimum="0" maximum="0"/>
  <config>
    <relationshipType>system:reference</relationshipType>
    <allowContentType>citation</allowContentType>
    <allowContentType>my.other.app:quote</allowContentType>
    <allowPath>${site}/people/</allowPath>
    <allowPath>./*</allowPath>
    <allowPath>/quotes*</allowPath>
  </config>
</input>
```

relationship This setting defines the name of which relationship-type to use. Default is system:reference.

allowContentType This is used to limit the content types that may be selected for this input. Use one setting for each content-type.

allowPath This is used to limit the path of the content that may be selected for this input. The site on which the content exists, can be wildcarded with ${site} Use one setting for each path expression.

```
<!--
All children of <site>/people, e.g
  /mySite/people/myContent
  /mySite/people/myGroup/anotherContent
-->
<allowPath>${site}/people/*</allowPath>

<!--
All content in mySite starting with people, including children, e.g
```

```
  /mySite/peoples
  /mySite/people/myContent
  /mySite/peoples/myContent
  /mySite/people/myGroup/anotherContent
-->
<allowPath>/mySite/people*</allowPath>

<!-- All children of the current content -->
<allowPath>./*</allowPath>

<!-- All children of the current content's parent -->
<allowPath>../*</allowPath>
```

CustomSelector

Selector input type with custom data source. Application developers must create a service that returns results according to the required JSON format, and then specify the service name in the input config. For information on creating a service see the *Services* section.

Below there are two sample usages of CustomSelector input type:

```
<input name="my-custom-selector" type="CustomSelector">
  <label>My Custom Selector</label>
  <occurrences minimum="0" maximum="0"/>
  <config>
    <service>my-custom-selector-service</service>
  </config>
</input>

<input name="musicTrack" type="CustomSelector">
  <label>Intro song</label>
  <config>
    <service>spotify-music-selector</service>
    <option value="genre">classic</option>
    <option value="sortBy">length</option>
  </config>
</input>
```

service The name of a JavaScript service file, located under `/resources/services/[serviceName]/[serviceName].js`. You can also refer to a service file in another application, for example *com.myapplication.app:myservice*.

option Parameter to pass to the service. Option parameters allow customizing a CustomSelector to be used in different contexts. There can be multiple options or none. The options will be included in the HTTP request to the service as name-value query parameters.

Service Request

In addition to the option values, if set on the input type `<config>`, the service will receive the following query parameters in the HTTP request:

ids Array of item ids already selected in the CustomSelector. The service is expected to return the items with the specified ids.

start Index of the first item expected. Used for pagination of the results.

count Maximum number of items expected. Used for pagination of the results.

query String with the search text typed by the user in the CustomSelector input field.

Service Response

The service controller must have a GET handler that returns results in JSON format. The JSON object returned must include `total` and `count` properties as numbers, and `hits` containing an array of items. Each item in the hits property must have the following fields:

id Unique Id of the option

displayName Option title

description (optional) Detailed description

iconUrl (optional) Path to the thumbnail image file

icon (optional) Inline image content (for example, SVG)

Example of JSON response from a CustomSelector service:

```
{
  "total": 10,
  "count": 2,
  "hits": [
    {
      "id": "1",
      "displayName": "Option number 1",
      "description": "External SVG file is used as icon",
      "iconUrl": "/some/path/images/number_1.svg"
    },
    {
      "id": "2",
      "displayName": "Option number 2",
      "description": "Inline SVG markup is used as icon",
      "icon": {
        "data": "<svg xmlns=\"http://www.w3.org/2000/svg\"/>",
        "type": "image/svg+xml"
      }
    }
  ]
}
```

Please check our tutorial on how to *Build a Custom Selector*.

Date

A simple field for dates with a calendar pop-up box in the admin console. The default format is `yyyy-MM-dd`.

```
<input name="name" type="Date">
  <label>Some Date</label>
  <occurrences minimum="0" maximum="1"/>
  <default>2011-09-12</default>
</input>
```

default The format for the default date value can be:

- Date in ISO 8601 format: `yyyy-MM-dd` (e.g. “2016-12-31”)
- Relative date expression (e.g. “+1year -12days”)

A relative date expression is a sequence of one or more date offsets. An offset consists of: a plus or minus sign, followed by an integer, followed by a date unit string (e.g. “+3 days”)

The date unit can be expressed as a singular unit, plural unit, or initial letter:

“year”	“years”	“y”
“month”	“months”	“M”
“week”	“weeks”	“w”
“day”	“days”	“d”

An offset can also be the string `now`, which means current date.

DateTime

A simple field for dates with time. A pop-up box with a calendar and time selector allows easy editing. The format is `yyyy-MM-dd hh:mm` for example, `2015-02-09T09:00`. The date-time could be of type `local` (no datetime) or with a timezone. This is done using configuration:

```
  <input name="name" type="DateTime">
    <label>DateTime (with tz)</label>
    <occurrences minimum="0" maximum="1"/>
    <config>
      <timezone>true</timezone>
    </config>
  </input>
```

timezone `true` if timezone information should be used. Default is `false`.

default The format for the default date value can be:

- Combined date and time in ISO 8601 format, with timezone: `yyyy-MM-ddThh:mm±hh:mm` (e.g. “2016-12-31T23:59+01:00”)
- Combined date and time in ISO 8601 format, without timezone: `yyyy-MM-ddThh:mm` (e.g. “2016-12-31T23:59”)
- Relative datetime expression (e.g. “+1year -12hours”)

Note that the ISO8601 format consists of concatenating a complete date expression, the letter `T` as a delimiter, and a valid time expression.

The timezone offset is a plus or minus sign, followed by an hour offset, followed by a colon, followed by a minute offset. A timezone offset of zero can also be represented as ‘Z’, meaning UTC or Zulu time. It is equivalent to offset *+00:00*.

A relative date expression is a sequence of one or more datetime offsets. An offset consists of: a plus or minus sign, followed by an integer, followed by a date/time unit string (e.g. “+3 days”)

The date unit can be expressed as a singular unit, plural unit, or initial letter:

“year”	“years”	“y”
“month”	“months”	“M”
“week”	“weeks”	“w”
“day”	“days”	“d”
“hour”	“hours”	“h”
“minute”	“minutes”	“m”

An offset can also be the string `now`, which means current date and time.

Examples:

```
<input name="dateTimeDefaultTz" type="DateTime">
  <label>DateTime (with tz and default value)</label>
  <config>
```

```
    <timezone>true</timezone>
  </config>
  <default>2000-01-01T12:30+01:00</default>
</input>

<input name="dateTimeDefaultNoTz" type="DateTime">
  <label>DateTime (without tz and default value)</label>
  <default>2000-01-01T12:30</default>
</input>

<input name="dateTimeRelative" type="DateTime">
  <label>DateTime (relative default value)</label>
  <default>+1year -12hours</default>
</input>

<input name="dateTimeNow" type="DateTime">
  <label>DateTime (current time as default value)</label>
  <default>now</default>
</input>
```

Double

A double value input-type.

```
<input name="rate" type="Double">
  <label>Interest rate</label>
  <default>3.89</default>
</input>
```

default This element specifies a default value. The value can be any double-precision floating-point number, with the dot character as decimal separator.

GeoPoint

Stores a GPS coordinate as two comma-separated decimal numbers.

- The first number must be between -90 and 90, where a negative number indicates a location south of equator and a positive is north of the equator.
- The second number must be between -180 and 180, where a negative number indicates a location in the western hemisphere and a positive number is a location in the eastern hemisphere.

```
<input name="name" type="GeoPoint">
  <label>Location</label>
  <occurrences minimum="0" maximum="1"/>
</input>
```

HtmlArea

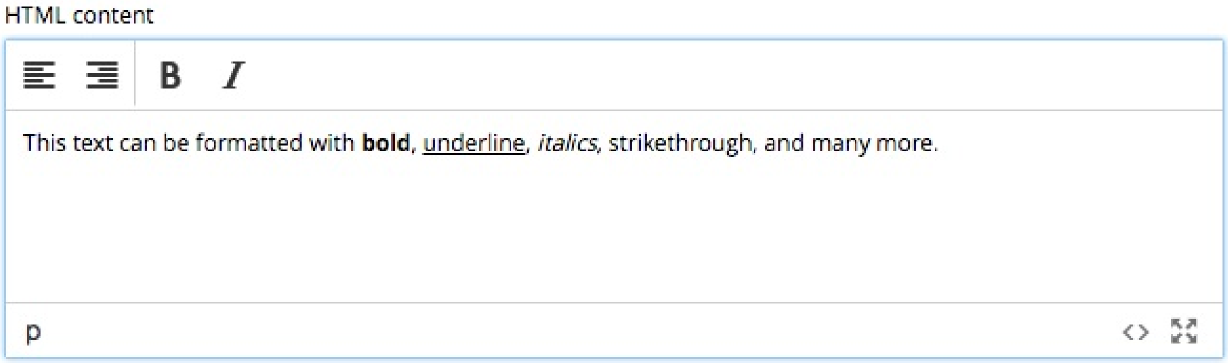
A field for inputting multi-line text, with formatting options.

```
<input name="description" type="HtmlArea">
  <label>Description</label>
  <default><h3>Enter description here</h3></default>
</input>
```

default This element specifies a default value. The value can contain any HTML elements, but tags must be correctly closed since the input type is defined inside an XML.

Tip: HTML Area is configured with default set of tools but the toolbar can be customized. Using the config setting you can exclude specific tools from being shown (use "*" to exclude all tools at once) and/or include those that you want to have in the toolbar. Separate tools with a space and use "|" character to group tool buttons together. Complete list of supported tools can be found in description of *Html Area input type*.

```xml
<input name="description" type="HtmlArea">
  <label>Description</label>
  <default><h3>Enter description here</h3></default>
  <config>
    <exclude>*</exclude>
    <include>alignleft alignright | bold italic</include>
  </config>
</input>
```



HTML content

This text can be formatted with **bold**, underline, *italics*, strikethrough, and many more.

p

Default configuration of the HTML Area toolbar is shown below:

```
styleselect | alignleft aligncenter alignright alignjustify | bullist numlist outdent indent | charma
```

Name	Description
styleselect	Text format menu
alignleft	Left align content
aligncenter	Center content
alignright	Right align content
alignjustify	Justify content
anchor	Insert an anchor
bullist	Add a bullet list
numlist	Insert a numbered list
outdent	Decrease indent
indent	Increase indent
charmap	Insert a special character
anchor	Insert an anchor
image	Insert/Edit an image
macro	Insert a macro
link	Insert/Edit a link
unlink	Remove link
table	Table format menu
pastetext	Toggle paste text mode

These are additional tools supported by HTML Area that can be used in the input config:

Name	Description
backcolor	Change text background color
blockquote	Add a quote block
bold	Make text bold
copy	Copy selected text into buffer
cut	Cut selected text into buffer
forecolor	Change text color
hr	Insert a horizontal line
italic	Make text italic
ltr	Left-to-right text direction
paste	Paste text from buffer into HTML Area
preview	Preview HTML Area contents
redo	Repeat last action
removeformat	Remove formatting of selected text
rtl	Right-to-left text direction
searchreplace	Find or replace text
strikethrough	Apply strikethrough effect to text
styleselect	Text format menu
subscript	Add subscript effect
superscript	Add superscript effect
underline	Underline text
undo	Undo last action
visualchars	Show hidden characters
visualblocks	Show hidden blocks

ImageSelector

Pick a reference to another existing image or upload a new image. Supported image types are:

- Jpeg
- Png
- Gif
- Svg

```
<input name="image" type="ImageSelector">
  <label>Non-required image</label>
  <occurrences minimum="0" maximum="1"/>
  <config>
          <allowPath>./*</allowPath>
  </config>
</input>
```

allowPath This is used to limit the path of the images that may be selected for this input. The site on which the content exists, can be wildcarded with ${site} Use one setting for path expression.

Long

A simple field for large integers.

```
<input name="count" type="Long">
  <label>Count</label>
  <default>42</default>
</input>
```

default This element specifies a default value. The value can be any valid integer.

RadioButton

An input type for selecting one of several options, defined in the `config` element.

```
<input name="name" type="RadioButton">
  <label>Radio Buttons</label>
  <occurrences minimum="0" maximum="0"/>
  <config>
    <option value="one">Option One</option>
    <option value="two">Option Two</option>
  </config>
  <default>one</default>
</input>
```

option This element defines the option label. `value` attribute defines the actual value to set when this option is selected. Multiple `option` settings are ordered.

default This element specifies the default option for the radio button. It should be equal to one of the `option` values.

Tag

An intuitive input format for specifying a set of simple strings.

```
<input name="name" type="Tag">
  <label>Location</label>
  <occurrences minimum="0" maximum="1"/>
</input>
```

TextArea

A field for inputting multi-line text.

```
<input name="description" type="TextArea">
  <label>Description</label>
  <default>Description goes here</default>
</input>
```

default This element specifies a default string value for the TextArea.

TextLine

A field for inputting a single line of text.

```
<input name="socialSecurityNumber" type="TextLine">
  <label>SSN</label>
  <occurrences minimum="0" maximum="1"/>
  <config>
    <regexp>\b\d{3}-\d{2}-\d{4}\b</regexp>
  </config>
  <default>000-00-0000</default>
</input>
```

regexp A regular expression that restricts the valid values for the input. Optional, if not set any text is a valid value.

default This element specifies a default string value for the TextLine.

Time

A simple field for time. A pop-up box allows simple selection of a certain time. The default format is `hh:mm`.

```
<input name="name" type="Time">
  <label>My Time</label>
  <occurrences minimum="0" maximum="1"/>
  <default>now</default>
</input>
```

default The format for the default time value can be:

- Time in 24h format: `hh:mm` (e.g. “23:59”)
- Relative time expression (e.g. “+1hour -12minutes”)

A relative time expression is a sequence of one or more time offsets. An offset consists of: a plus or minus sign, followed by an integer, followed by a time unit string (e.g. “+3 minutes”)

The time unit can be expressed as a singular unit, plural unit, or initial letter:

“hour”	“hours”	“h”
“minute”	“minutes”	“m”

An offset can also be the string `now`, which means current time.

Item Sets

Item sets represent a special capability of forms that allow you to nest other form items hierarchically.

Inputs in item sets are grouped into logical units, allowing them to repeat as a complex input type - since item sets support occurrences too. Item sets are both visually and semantically grouped as the name of the item set is used in the persisted property structure. An item set actually produces a property set.

Here is an example of an item set with two inputs. The resulting form will allow multiple entries of phone numbers with labels:

```
<item-set name="contact_info">
  <label>Contact Info</label>
  <items>
    <input name="label" type="TextLine">
      <label>Label</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
    <input name="phone_number" type="TextLine">
      <label>Phone Number</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
  </items>
  <occurrences minimum="0" maximum="0"/>
</item-set>
```

name The set needs a name for reference in result sets.

label The set label is printed as a header on the box that will surround the group in the input form.

occurrences Occurrence configuration can be done at any level.

Tip: It is also possible to nest item sets inside each other

Option Sets

An option set represents a group of options rendered as either radio-buttons or checkboxes. Each option may or may not have a form of inputs it consists of. An option can be considered to be a field-set with selectable header.

By default, an option form will only be shown upon selection of the option, but the entire option set may be configured to have all of its options expanded by default.

It’s also possible to pre-select specific options by default.

Here is an example of a multi-select option set with options expanded by default, empty first option and pre-selected second option:

```xml
<option-set name="checkOptionSet">
  <label>Multi-selection OptionSet</label>
  <expanded>true</expanded>
  <occurrences minimum="1" maximum="1"/>
  <help-text>You can select up to 2 options</help-text>
  <options minimum="1" maximum="2">
    <option name="option_1">
      <label>Option 1</label>
      <help-text>Help text for Option 1</help-text>
    </option>
    <option name="option_2">
      <label>Option 2</label>
      <default>true</default>
      <items>
        <input name="contentSelector" type="ContentSelector">
          <label>Content selector</label>
          <occurrences minimum="0" maximum="0"/>
          <config/>
        </input>
      </items>
    </option>
    <option name="option_3">
      <label>Option 3</label>
      <help-text>Help text for Option 3</help-text>
      <items>
        <input name="textarea" type="TextArea">
          <label>Text Area</label>
          <occurrences minimum="0" maximum="1"/>
        </input>
        <input name="long" type="Long">
          <label>Long</label>
          <indexed>true</indexed>
          <occurrences minimum="0" maximum="1"/>
        </input>
      </items>
    </option>
  </options>
</option-set>
```

@name The set needs a name for reference in result sets.

label The label is displayed as a header of the option set.

expanded Optional. Set to `true` to expand all of the options by default

occurrences Detailed definition of how many times this option set may be repeated inside one content.

help-text Optional. Help text for the entire option set.

options Container of options.

> **@minimum** Required. Minimum number of options that must be selected in this option set.
>
> **@maximum** Required. Maximum number of options that can be selected in this option set. Setting this attribute to a value greater than 1 will result in rendering of a multi-select option set with options rendered as checkboxes. Setting the attribute value to 1 will render options as radio-buttons (single-select option set). Once the maximum of selected options is reached, the rest of the options will be disabled.
>
> **option** Container of the option form.
>
> > **@name** Option name. Must be unique within the option set.
> >
> > **label** Label of the option's checkbox or radio button.
> >
> > **help-text** Optional. Help text for the option.
> >
> > **default** Optional. Set to `true` to pre-select the option.
> >
> > **items** Optional. Container of the option form's inputs.

Schema Layouts

To shape the presentation of a form, one can use layouts. Currently, only one layout exists.

Field set

A field set may be used to group items visually. The example below will create a form in the admin console with the inputs grouped under the label of the field set.

```
<field-set name="metadata">
  <label>Metadata</label>
  <items>
    <input name="tags" type="Tag">
      <label>Tags for tag cloud</label>
      <occurrences minimum="0" maximum="5"/>
    </input>
  </items>
</field-set>
```

@name The field set needs a name for reference.

label The label will appear as a heading above the inputs that are grouped inside.

items The fields inside the set must be listed inside an `items` element.

Mixins

Structures of data that are repeated in different content types or component descriptors may be defined as mixins. Such structures (like some address fields or a combobox with a standard set of values) would be defined once in a mixin and then the mixin would be called in other schemas that require these fields. The mixin definition file must be placed in the folder `site/mixins/[name]` and named `[name].xml`. For example, `site/mixins/us-address/us-address.xml`.

```xml
<mixin>
  <display-name>U.S. Address format</display-name>
  <items>
    <input type="TextLine" name="addressLine">
      <label>Street address</label>
      <occurrences minimum="0" maximum="2"/>
    </input>
    <input type="TextLine" name="city">
      <label>City</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
    <input type="TextLine" name="state">
      <label>State</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
    <input type="TextLine" name="zipCode">
      <label>Zip code</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
  </items>
</mixin>
```

Tip: A mixin may optionally have its own specific icon. The icon can be assigned to the mixin by adding a PNG file with the same name, in the mixin folder, e.g. `site/mixins/us-address/us-address.png`

Using a mixin

Below is an example of a simple content type that uses the `us-address` mixin (inline) and the `menu-item` mixin (x-data). Notice that the name of the mixin file is used and not the mixin's Display Name.

```xml
<content-type>
  <display-name>Using mixins</display-name>
  <super-type>base:structured</super-type>
  <form>
    <inline mixin="us-address"/>
  </form>
  <x-data mixin="menu-item"/>
</content-type>
```

inline When a mixin is added with the `inline` element, the inputs will be included with the content data. Inline mixins can be used in content types, component descriptors, and the site.xml file.

x-data Mixins can also be added with an `x-data` element in content types and the site.xml file. Be aware that any `x-data` inputs added to the site.xml file will be applied to all content types in the site.

Relationship Types

Note: Relationship types are optional and experimental, they currently do not provide relevant functionality to your projects

Custom content types may have relationships to each other or other content types. For instance, a person may have an image, or an employee may have a boss, or belong to a department. These relationships must be defined with a specific *relationship type*, then used in the custom content with an input type `ContentSelector`. The relationship type definition is an XML file. It must be placed in the folder, `site/relationship-types/[name]` and be named `[name].xml`. Here is an example of a relationship-type:

```xml
<relationship-type>
  <display-name>Citation</display-name>
  <from-semantic>citation in</from-semantic>
  <to-semantic>cited by</to-semantic>
  <allowed-from-types/>
  <allowed-to-types>
    <content-type>com.enonic.xp.modules.features:article</content-type>
  </allowed-to-types>
</relationship-type>
```

from-semantic Text to describe the “from” relationship.

to-semantic Text to describe the “to” relationship.

allowed-from-types Any content type may use this relationship-type.

allowed-to-types Wherever this relationship-type is used, only an article may be selected.

The content types have the format `module-name:content-type-name`. The module may be `system` for built-in types.

Tip: A relationship type may optionally have its own specific icon. The icon can be assigned to the relationship type by adding a PNG file with the same name, in the relationship type folder, i.e. `site/relationship-types/[name]/[name].png`

System relationship types

There are two default relationship types that may be used out of the box. These represent general relationship types that may be reused often.

`system:reference` No content type restriction, from-semantic = “relates to”, to-semantic = “related of”.

`system:parent` No content type restriction, from-semantic = “parent of”, to-semantic = “child of”.

Forms

The main purpose of the schema concept is to construct forms that can be edited through the admin interface or used programmatically, without coding a custom interface and complex controllers. A form is basically a composition of layouts and input types. When a form is populated and submitted, the result will be a basic property structure that can be stored directly into *Nodes*.

Some hands on examples where forms are used in the system are *Content Types* and *Sites*.

Basic Setup

Forms can be defined through Java or XML, where the latter is the most common.

Below is an example configuration in xml:

```
<form>
  <input name="choice1" type="ComboBox">
    <label>Choice1</label>
    <occurrences minimum="0" maximum="1"/>
    <config>
        ...
    </config>
  </input>
</form>
```

Adding Mixins

To simplify maintenance of forms, mixins can be created and injected into a form simply by referencing it. The form will render as if everything in the mixin was written directly in the form itself. Read more about *Mixins*.

```
<form>
  <field-set name="basic">
    <label>Status</label>
    <items>
      <inline mixin="us-address"/>
    </items>
  </field-set>
</form>
```

Serverside JavaScript

Enonic XP primarily uses server-side JavaScript for application development. Our goal is to enable any developer - PHP, .net, Java, Python, etc, etc to quickly be productive with Enonic XP.

Here are some highlights on how it works:

- Runs on the Java Virtual Machine using the Nashorn JavaScript engine, a high performance, portable and robust platform.
- Multithreaded request-response approach - simplifying software development and utilization of modern multi-core hardware
- Implements central parts of CommonJS module specification (http://wiki.commonjs.org/wiki/Modules/1.1) like RequireJS - but not all
- You can invoke Java directly from your scripts - quickly accessing powerful Java libraries

Beyond simply executing JavaScript on the server, the XP framework provides a range of capabilities, primarily associated with HTTP. Read more below to learn about the basic concepts.

HTTP Controllers

Serverside JavaScript is used in the http controllers of Enonic XP. Every *Page*, *Part*, *Layout*, *Services*, *Controller Mappings* etc. must have a controller.

JavaScript controllers are invoked from the portal by exporting functions matching the desired HTTP Method it implements. As such, any controller must explicitly declare one or more "exports" in order to handle requests: `GET`, `POST`, `DELETE` are examples of such methods.

The appropriate function will automatically be invoked for every request sent to the controller

Example usage

```
// Handles a GET request
exports.get = function(req) {}

// Handles a POST request
exports.post = function(req) {}
```

A handler function receives a parameter with a `request` object, and returns a `response` object.

```
exports.get = function(request) {

  if (request.mode === 'edit') {
    // do something...
  }

  var name = request.params.name;
  log.info('Name = %s', name);

  return {
    body: 'Hello ' + name,
    contentType: 'text/plain'
  };

};
```

Global JavaScript objects and functions

The following global functions and objects are available in the Enonic XP framework.

App

The globally available `app` object holds information about the contextual app it was delivered from.

`app.`**`name`**
`Name of the application.`

`app.`**`version`**
`Version of the application.`

`app.`**`config`**
`Application configuration.` **`This can be set using`**
`` ``$XP_HOME/config/<app.name>.cfg``. `` **`Every time the configuration is`**
`changed the app is restarted.`

Examples:

```
// Get application name
var name = app.name;  // com.enonic.app.superhero

// Get application version
var version = app.version;  // 1.2.0
```

Log

This globally available `log` object holds the logging methods. It's one method for each log level and takes the same number of parameters.

`log.`**`debug`**(*message*[, *args*])

Arguments

- **`message`** (*`string`*) – Message to log as a debug-level message.
- **`args`** (*`array`*) – Optional arguments used in message format.

`log.`**`info`**(*message*[, *args*])

Arguments

- **`message`** (*`string`*) – Message to log as an info-level message.
- **`args`** (*`array`*) – Optional arguments used in message format.

`log.`**`warning`**(*message*[, *args*])

Arguments

- **`message`** (*`string`*) – Message to log as a warning-level message.
- **`args`** (*`array`*) – Optional arguments used in message format.

`log.`**`error`**(*message*[, *args*])

Arguments

- **`message`** (*`string`*) – Message to log as an error-level message.
- **`args`** (*`array`*) – Optional arguments used in message format.

Examples:

```
// Log a simple message
log.debug('Hello World');

// Log a formatting message
log.info('Hello %s', 'World');

// Log a formatting message
log.warning('%s %s', 'Hello', 'World');

// Log using the built-in JSON converter
log.error('My JSON %s', object );
```

Resolve()

This globally available function resolves a fully qualified path to a local resource based on the current location. It does not check if a resource exists at the specified path. This function supports both relative (with dot-references) and absolute paths.

`resolve`(*path*)

Arguments

- **`path`** (*`string`*) – Path to resolve using current location.

Returns The fully qualified resource path of the location.

Examples:

```
// Absolute path
var path1 = resolve('/views/myview.html');

// Relative path - in this case, the resource must be in the same folder
var path2 = resolve('myview.html');

// Relative path (same as above)
var path3 = resolve('./myview.html');

// Relative path - resource is one level up
var path4 = resolve('../myview.html');
```

Require()

This globally available function will load a JavaScript file and return the exports as objects. The function implements parts of the CommonJS Modules Specification.

`require` (*path*)

Arguments

- **path** (*string*) – Path to the JavaScript to load.

Returns The loaded JavaScript object exports.

Examples:

```
// Absolute path
var lib1 = require('/lib/mylib.js');

// Relative path
var lib2 = require('mylib');

// Relative path (same as above)
var lib3 = require('./mylib.js');

// Relative path
var lib4 = require('../mylib');
```

If the path is relative then it will start looking for the file from the local directory. If the file is not found there, it will start scanning in parent directories that have a /lib folder until it reaches the resources/ folder. The file extension .js is not required.

Exports

The globally available `exports` keyword is used to expose functionality from a given JavaScript file (controllers, libraries etc). This is part of the require.js spec.

Simply use the `exports` keyword to expose functionality from any JavaScript file.

Double underscore __

The double underscore is available in any server-side JavaScript code and is used for wrapping Java objects in a JavaScript object. Read more about *Invoking Java*.

HTTP Request

The following object is passed along with every HTTP request. The object is similar to many traditional request objects, except for two special properties: mode and branch. These properties are specific to the XP Portal, automatically indicating the contextual branch and rendering mode.

The `request` object represents the HTTP request and current context for the controller.

```json
{
  "method": "GET",
  "scheme": "http",
  "host": "enonic.com",
  "port": "80",
  "path": "/my/page",
  "url": "http://enonic.com/my/page?debug=true",
  "remoteAddress": "10.0.0.1",
  "mode": "edit",
  "branch": "master",
  "params": {
    "debug": "true"
  },
  "headers": {
    "Language": "en",
    "Cookies": "mycookie=123; other=abc;"
  },
  "cookies": {
    "mycookie": "123",
    "other": "abc"
  }
}
```

method HTTP method of the request.

scheme Name of the scheme used to make this request ("http" / "https").

host Host name of the server to which the request was sent.

port Port of the server to which the request was sent.

path Path of the request.

url URL of the request.

remoteAddress IP address of the client that sent the request. If the X-Forwarded-For header is set, its value will override the client IP.

mode Portal rendering mode, one of: `edit`, `preview`, `live`.

branch Name of the repository branch, one of: `draft`, `master`.

body Optional text value

params Name/value pairs with the query/form parameters from the request.

headers Name/value pairs with the HTTP request headers.

cookies Name/value pairs with the HTTP request cookies.

HTTP Response

The `response` object is the value returned by an HTTP controller - as a response to an *HTTP Request*.

```
{
  "status": 200,
  "body": "Hello World",
  "contentType": "text/plain",
  "headers": {
      "key": "value"
  },
  "cookies": {},
  "redirect": "/another/page",
  "pageContributions": {},
  "postProcess": true,
  "applyFilters": true
}
```

status HTTP response status code (default is `200`).

body HTTP message body of the response that can either be a string or a JavaScript object.

contentType MIME type of the body (defaults to `text/plain; charset=utf-8`).

headers Name/value pairs with the HTTP headers to be added to the response.

cookies HTTP cookies to be added to the response. Will be described in a later section.

redirect URI to redirect to. If specified, the value will be set in the "Location" header and the status will be set to 303.

pageContributions A special filter available for sites and page components allowing page components to contribute html to the main page markup. See *Page Contributions*

postProcess Post-processing is a special filter for sites and pages, if enabled it will reprosess a page looking for page contributions and rendering components in a page. (See also *Page Contributions*) (default is `true`). Set to false if you want to speed up page rendering in cases where there are no regions or page components.

applyFilters Whether or not to execute the filters after rendering. Set to `false` to skip execution of filters. (See also *Response Filters*) (default is `true`).

HTTP Cookies

There are two ways that Http Cookie values can be set in responses.

- If the value is a string then the cookie is created using default settings.
- If the value is an object then it will try to apply the settings. Every field is optional except "value".

Here's an example of how the cookies should be set:

```
return {
    status: 200,
    body: "Hello World",
    headers: {
        "header1": "value1"
    },
    cookies: {
        "plain": "value",
        "complex": {
            value: "value",
            path: "/valid/path",
            domain: "enonic.com",
            comment: "Some cookie comments",
            maxAge: 2000,
      secure: false,
      httpOnly: false
    }
  }
};
```

Websockets

Warning: Websocket support is experimental.

Websocket support allows a service to act as a websocket channel that you can connect to from a web-browser.

A `get` method must be implemented to handle initialization of the websocket.

```
// Create a websocket if websocket request.
exports.get = function (req) {

  if (!req.webSocket) {
    return {
      status: 404
    };
  }

  return {
    webSocket: {
      data: {
        user: "test"
      },
      subProtocols: ["text"]
    }
  };
};
```

A websocket event handler named `webSocketEvent` is required. It will be called for every websocket event from a client. See example below.

```
// Listen to a websocket event
exports.webSocketEvent = function (event) {

  if (event.type == 'open') {
    // Do something on open
  }

  if (event.type == 'message') {
    // Do something on message recieved
  }

  if (event.type == 'close') {
    // Do something on close
  }

};
```

Below is an example of a simple chat. A library called `lib-websocket` has functions for sending messages back and adding/removing clients in groups. Adding to groups allows for multicast message sending.

```
// Lib that contains websocket functions.
var webSocketLib = require('/lib/xp/websocket');

// Listen to a websocket event
exports.webSocketEvent = function (event) {

  if (event.type == 'open') {
    // Send message back to client
    webSocketLib.send(event.session.id, 'Welcome to our chat');

    // Add client into a group
    webSocketLib.addToGroup('chat', event.session.id);
  }

  if (event.type == 'message') {
    // Propegate message to group
    webSocketLib.sendToGroup('chat', event.message);
  }

  if (event.type == 'close') {
    // Remove client from a group
    webSocketLib.removeFromGroup('chat', event.session.id);
  }

};
```

Invoking Java

In Enonic XP, there is a standard object named __ (double underscore), accessible from any serverside JavaScript code, which provides a way to wrap Java objects in a JavaScript object. The __ object has functions that allow JavaScript to communicate with Java classes. The `newBean` function will wrap the Java object named in the parameter, for instance:

```
var bean = __.newBean('com.enonic.xp.lib.io.IOHandlerBean');
```

This line is from the `lib-io` library, which is a good example of how this is used. In the Java `IOHandlerBean` class, there are several methods, like the `readLines` method:

```
public List<String> readLines( final Object value )
    throws Exception
{
    final CharSource source = toCharSource( value );
    return source.readLines();
}
```

This method is now accessible as a function on the JavaScript `bean` and may be invoked from JavaScript, like this:

```
exports.readLines = function (stream) {
    return __.toNativeObject(bean.readLines(stream));
};
```

This results in a global JavaScript function `readLines`. This example also shows the use of the `toNativeObject` method, which in this case, converts a Java String array to a JSON object. The reference documentation for the __ object can be found here: The __ object.

Assets

Applications and libraries commonly use files that will be delivered to the client (typically web browsers) without being modified. Examples are icons, css files, javascript files etc. Enonic XP provides a standard and optimized approach to serving assets across applications.

Developers simply place the files they want to use into their project's /src/main/resources/assets/ folder. These files can then be dynamically accessed through the asset service. The asset service is typically available through _/asset/<myapp>/path/to/myasset.ext.

The assetUrl() portal function lets you easily create links to assets.

assetUrl (*path [,application] [,type] [,params]*)

Arguments

- **path** (*string*) – Path to the asset.
- **application** (*string*) – Application where the asset exists. Default is current application.
- **type** (*string*) – URL type. Either server (server-relative URL) or absolute. Default is server.
- **params** (*object*) – Custom parameters to append to the url.

Returns The the relative or absolute URL to the asset.

Services

Services allow the creation of http endpoints without binding them to specific paths. Each service must have a JavaScript controller file and optionally an XML descriptor placed in the folder `services/[service-name]`

Descriptor

The service descriptor is an XML file that that is used to define which rights are required to access the service.

The descriptor file must have the same name as the service, i.e. `services/[service-name]/[service-name].xml`:

```
<service>
  <allow>
    <principal>role:system.admin</principal>
    <principal>role:myapp.myrole</principal>
  </allow>
</service>
```

Controller

A service controller handles requests to the service. The controller is a required file written in JavaScript and must have the same name as the service, i.e. `services/[service-name]/[service-name].js`.

A controller exports a method for each type of HTTP request that should be handled. The handle method has the request object as a parameter and returns the response object (see *HTTP Controllers*).

The following example is a simple service that returns a JSON object with the date and a counter.

```
var counter = 0;

exports.get = function(req) {

  counter++;

  return {
    body: {
      time: new Date(),
      counter: counter
    },
    contentType: 'application/json'
  };

};
```

Access

The service can then be accessed on a relatively mounted URL, as seen below, where `application` is the application name (without version):

```
*/_/service/[application]/[service-name]
```

The portal function `serviceUrl()` will create a dynamic URL for a service.

`serviceUrl` (*name [,application] [,type] [,params]*)

Arguments

- **`name`** (*`string`*) – Name of the service.
- **`application`** (*`string`*) – Application where the service exists. Default is current application.
- **`type`** (*`string`*) – URL type. Either server (server-relative URL) or absolute. Default is server.
- **`params`** (*`object`*) – Custom parameters to append to the url.

Returns The the relative or absolute URL to the service.

Views

Instead of composing the HTML, JSON or other output in your javascript controllers, it's often easier to use a full MVC (Model View Controller) approach.

Enonic XP supports pluggable view technologies and ships the following view libraries out-of-the-box.

- Thymeleaf - typically used with HTML (see `lib-thymeleaf` in *Javascript Libraries*)
- Mustache - typically for use with JSON (see `lib-mustache` in *Javascript Libraries*)
- Xslt - recommended for XML processing (see `lib-xslt` in *Javascript Libraries*)

Sites

Sites can be built from one (or more) applications. But only applications that contain a specific structure and a site descriptor can be used for this purpose.

This chapter will dive into the details on how to build sites and the various components of a site, such as pages and parts.

Site Descriptors

To indicate that an application provides "site capabilities" and allow it to be added to sites, a site descriptor must be placed into the application. Within your project, simply add a file called `/main/src/resources/site/site.xml`.

The site.xml file also makes use of the *Schemas* concept, so you may easily define custom forms for configuring the application when adding it to a site. These configurations are made in the <config> element.

Listing 3.4: /main/src/resources/site/site.xml

```
<site>
  <config>
    <input type="TextLine" name="company">
      <label>Company</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
    <input type="TextArea" name="description">
      <label>Description</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
  </config>
</site>
```

All controllers within the app can access the configured values with the portal.getSiteConfig() function.

```
var portal = require('/lib/xp/portal');

// Find the site configuration for this app in current site.
var siteConfig = portal.getSiteConfig();
```

Extensions

An app's site.xml file may optionally contain other elements, placed outside the `<config>` node.

Use `<x-data>` element for adding *Mixins*. Adding a mixin this way will add that data on all the content in Content Studio.

Additionally, the `<filters>` element can be used for adding *Response Filters*.

Controller Mappings can also be configured in a site descriptor with a `<mappings>` element.

Content Types

To enable simple configuration and setup of publishing forms, validation and data types - Enonic XP ships with a content api. Central to this api are Content Types. Structured, indexed and searchable content items are created from

Content types. Content Types build on the *Forms* concept, so they are very similar to other configurable forms in Enonic XP.

Content repository

Every content item produced is eventually stored as nodes in the low level storage, read more about the *Node Domain*.

A system standard repository called `cms-repo` is initialized when installing Enonic XP. This is where content is stored when working with content in the Content Studio application or the content-API.

The content-API actively uses the branch capabilities. The `cms-repo` has two branches:

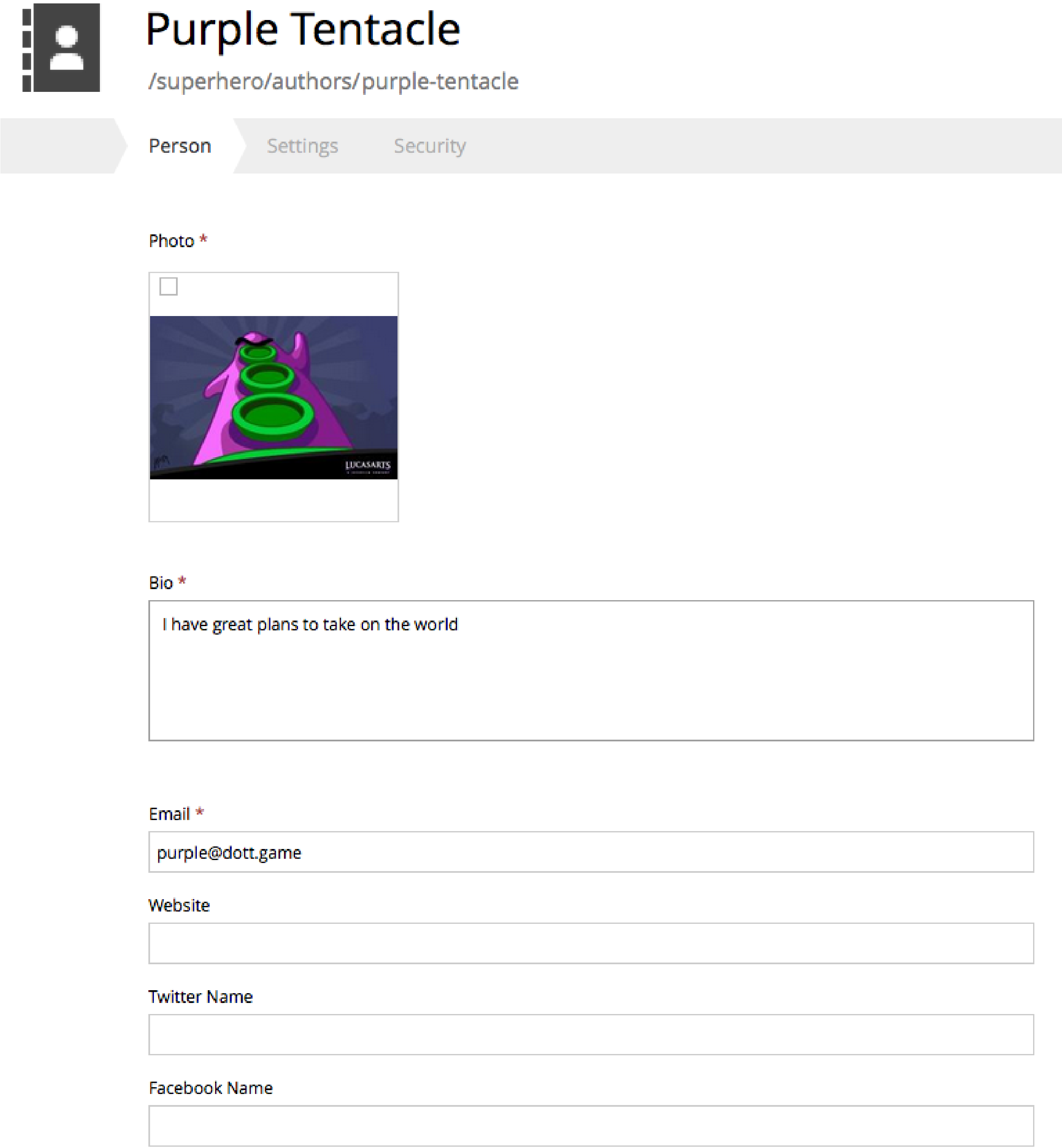
- `draft`
- `master`

The content seen while working in Content Studio is in the `draft` branch. Content in the portal is served from the `master` branch.

Publishing a content moves it from the `draft` branch to the `master` branch.

Sample Content type

A "Person" content type might look something like this:



Purple Tentacle

/superhero/authors/purple-tentacle

Photo *

□

Bio *

I have great plans to take on the world

Email *

purple@dott.game

Website

Twitter Name

Facebook Name

The underlying schema configuration would look like this

```
<?xml version="1.0" encoding="UTF-8"?>
<content-type>
  <display-name>Person</display-name>
  <super-type>base:structured</super-type>
  <form>
    <input type="imageSelector" name="photo">
      <label>Photo</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
    <input type="HtmlArea" name="bio">
      <label>Bio</label>
       <occurrences minimum="1" maximum="1"/>
    </input>
    <input type="TextLine" name="email">
      <label>Email</label>
      <occurrences minimum="1" maximum="1"/>
    </input>
    <input type="TextLine" name="website">
      <label>Website</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
    <input type="TextLine" name="twitter">
      <label>Twitter Name</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
    <input type="TextLine" name="facebook">
      <label>Facebook Name</label>
      <occurrences minimum="0" maximum="1"/>
    </input>
  </form>
</content-type>
```

And the persisted (and searchable node) would look like this: NB! The content type defined properties are stored within the 'data' propertySet.

```
{
  "_id": "c814b68c-7dd3-4851-a35e-6709b07409d4",
  "_name": "purple-tentacle",
  "_path": "/superhero/authors/purple-tentacle",
  "creator": "user:system:su",
  "modifier": "user:system:su",
  "createdTime": "2016-01-07T07:23:42.149Z",
  "modifiedTime": "2016-01-07T07:26:49.129Z",
  "type": "com.enonic.sampleapp:person",
  "displayName": "Purple Tentacle",
  "hasChildren": true,
  "language": "en",
  "valid": true,
  "data": {
    "photo": "10b6cf60-581a-43cd-aa76-30a0c3503a45",
    "bio": "<p>Have great plans to take on the world</p>",
    "email": "purple@dott.game"
  },
  "x": {},
  "page": {},
  "attachments": {}
}
```

Standard Content Properties

These are the standard content properties - value type and index options specified in parentheses.

_id (string) The content id (this is the same as node id)

_name (string, fulltext, ngram) The content name (same as node name)

_parent (reference) The parent content path (same as node parent)

attachment (propertySet) If content contains attachments, a list of attachments with respective properties will be listed here.

contentType (string) The content schema type.

creator (string) The user principal that created the content.

createdTime (dateTime) The timestamp when the content was created.

data (propertySet) Contains all user defined properties as defined by the contentType.

displayName (string, fulltext) Name used for display purposes.

language (string) The locale-property of the content.

modifiedTime (dateTime) Last time the content was modified.

owner (string) The user principal that owns the content.

page (propertySet) The page property contains page-specific properties, like template and regions. This will typically be reference to a page-template that supports the content-type.

publish (propertySet) Contains publish-information, e.g `publish.from`

site (propertySet) If the contentType is `portal:site`, this will contain site-specific information.

thumbnail A thumbnail representing the content.

type (string) the nodetype - Used to identify nodes of type content in the repository.

x (propertySet) A property-set containing properties from mixins, also known as xtra data.

Base Content Types

A set of basic content types are provided with the installation.

Content types have a set of properties you need to know about:

- Content types are named with their application name, i.e. base:folder, where "base" is the application - but also have a nice display name like "Folder"
- `abstract` (default: false) means you cannot create content with this content type
- `final` (default: false) means it is not possible to create content types that "extend" this
- `allow-child-content` (default: true) if false, it will prevent users from creating child items on content of this type. (i.e. prevents creating child items of images)

Folder (base:folder)

- abstract: false
- final: false
- allow-child-content: true

Folders are simply containers for child content, with no other properties than their name and Display Name. They are helpful in organizing your content.

Media (base:media)

- abstract: true
- final: false
- allow-child-content: false

This content type serves as the abstract supertype for all content types that are considered "files" in their natural habitat. These are listed on the *Media Content Types* page.

Shortcut (base:shortcut)

- abstract: false
- final: true
- allow-child-content: true

This is used for redirecting a visitor to another content item in the structure The content type name is `base:shortcut`.

Structured (base:structured)

- abstract: true
- final: false
- allow-child-content: true

This is possibly the most commonly used base type for creating other content types. The structured content type is the foundation for basically any other structured content you can come up with, such as the `Person` content in the previous example.

Unstructured (base:unstructured)

- abstract: false
- final: true
- allow-child-content: true

The unstructured content type is a special content type that permits the creation of any property or structure without actually defining it first. This is convenient for user generated content from forms on a site.

Caution: There is currently no UI for unstructured content so they will appear empty in the admin console. However, a custom page template that supports base:unstructured may easily be created to show name/value pairs.

Media Content Types

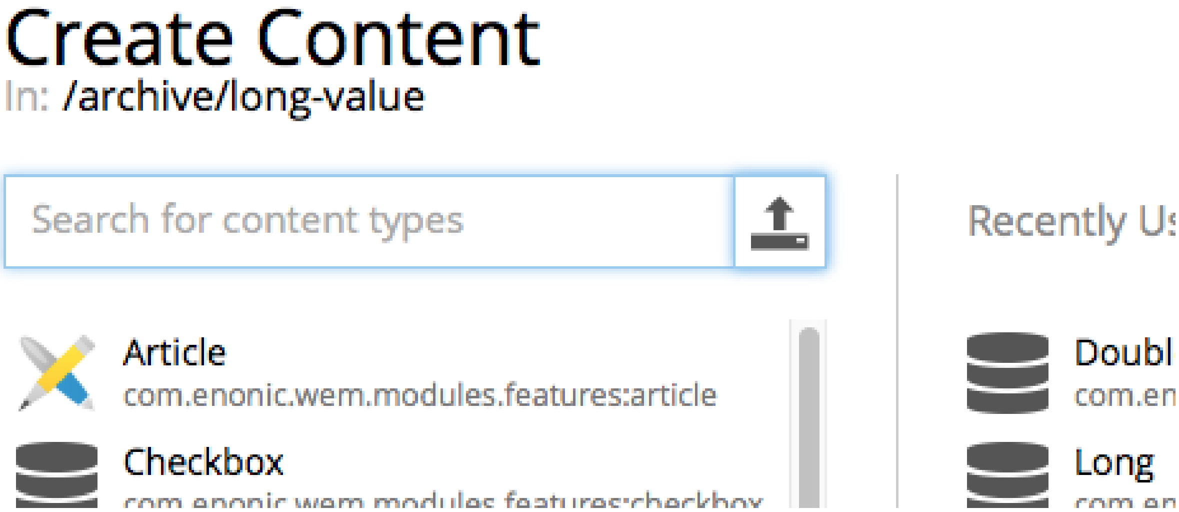
The system ships with a set of pre-defined media content types. When files are uploaded in the Content Studio interface or through the content API - they will be transformed to one of the following content-types.

Common settings for all the content types listed below.

- super-type: base:media

- abstract: false
- final: true
- allow-child-content: false

Tip: Enonic XP treats media content pretty much like any other content items - for instance the person, but they all have at least one attachment (namely the file).


Create Content
In: /archive/long-value
Search for content types
Recently Us
Article
com.enonic.wem.modules.features:article
Checkbox
com.enonic.wem.modules.features:checkbox
Doubl
com.en
Long
com.en

Here are the various media content types that also come installed with Enonic XP:

Text (media:text) Plain text files.

Data (media:data) Miscellaneous binary file formats.

Audio (media:audio) Audio files.

Video (media:video) Video files.

Image (media:image) Bitmap image files.

Vector (media:vector) Vector graphic files like .svg.

Archive (media:archive) File archives like .zip, tar and jar.

Document (media:document) Text documents with advanced formatting, like .doc, .odt and pdf.

Spreadsheet (media:spreadsheet) Spreadsheet files.

Presentation (media:presentation) Presentation files like Keynote and Powerpoint.

Code (media:code) Files with computer code like .c, .pl or .java.

Executable (media:executable) Executable application files.

Unknown (media:unknown) Everything else.

Portal content types

In order to build sites in a secure and fashionable manner, Enonic XP also ships with a few special purpose content types.

Site (portal:site)

- super-type: base:structured
- abstract: false
- final: true
- allow-child-content: true

The Site content type allows creating websites. By creating a content of type Site, it will become the root of a website.

This content type provides a special behavior for the content, allowing to select and configure applications for the website. The types (content types, relationship types and mixins) of the applications selected will be available to be used inside the website content tree.

Note: The content types of an application can only be used under a content of type Site which has the application selected.

Page Template (portal:page-template)

- super-type: base:structured
- abstract: false
- final: true
- allow-child-content: true

Page templates are the equivalent of "master slides" in keynote and powerpoint. They enable you to set up pages that will be used when presenting other content types. From the sample content type above, the page template "Person Show" was taking care of the presentation.

Template folder (portal:template-folder)

- super-type: base:folder
- abstract: false
- final: true
- allow-child-content: `portal:page-template` only

This is a special content-type. Every site automatically creates a child content of this type named `_templates`. The templates folder holds all the page templates of that site. It may not hold any other content type, and it may not be created manually in any other location.

Fragment (portal:fragment)

- super-type: base:structured
- abstract: false
- final: true
- allow-child-content: true

The Fragment content type represents a **reusable page component**. A content of this type contains a page component(*Part*, *Layout*, *Text*, *Image*) that can be re-used in other pages. But it only needs to be maintained in one place.

To create a content of type `portal:fragment` edit an existing page with *Page Editor*, select the context menu of an existing component in the page, and then clicking on “Create Fragment”. Once created, the fragment content can be referenced in other pages by inserting a *Fragment component* in the page.

A Fragment content can be edited with *Page Editor* and the changes applied to the component will immediately be available in the pages that include the fragment. When a page containing fragment a component is rendered, the components of the `portal:fragment` content pointed by the fragment component are rendered in the place of the fragment component.

There is a default page for rendering and edit fragments. The default page does not have any styles defined, but it is possible to render it with the application theme and styles by defining a controller mapping with `<match>type:'portal:fragment'</match>` (See *Controller Mappings*).

Custom Content Types

Custom Content Types can be created using Java or simple xml files - and deployed through applications.

When using xml, each content type must have a separate folder in the application resource structure. i.e. `site/content-types/<my-content-type-name>`.

Each folder must then hold a file with the name of the content type and `.xml` extension (e.g. `my-content-type-name.xml`).

This is the basic structure of a `content-type.xml` file:

```
<content-type>
  <display-name>Choices</display-name>
  <content-display-name-script>$('firstName', ' ', 'lastName')</content-display-name-script>
  <super-type>base:structured</super-type>
  <is-abstract>false</is-abstract>
  <is-final>true</is-final>
  <allow-child-content>true</allow-child-content>
  <form>
    <input name="choice1" type="ComboBox">
      <label>Choice1</label>
      <occurrences minimum="0" maximum="1"/>
      <config>
          ...
      </config>
    </input>
  </form>
</content-type>
```

display-name The display name of the content type is used throughout the admin console to recognize it. But the technical name is the name of the folder the file is placed in.

super-type Many properties are inherited from the super-type. All custom content types must either inherit `base:structured` directly or indirectly. The icon and the general form to edit the fields of the content are important properties that are inherited from `base:structured`.

is-abstract If a content type is abstract, no content of this type may be instantiated. It may still be used as a super type for other content types.

is-final Final content types may not be used as super types of other content types.

allow-child-content Default is true, which allows nodes to be added in the tree below a content of this type.

form Fields in the content type are defined as input elements which are placed inside the `form` element. All legal input types are described below.

input `name` and `type` are mandatory attributes of the input element. `label` and `occurrences` are mandatory child elements.

config Some input types have a complex configuration that is defined inside a `config` element. It is mandatory for the content types that need it.

content-display-name-script The name of a content may be generated by JavaScript from the values in the form, including values that are added through a mixin.

Tip: A content type may optionally have its own specific icon. The icon can be assigned to the content type by adding a PNG or SVG file with the same name, in the content type folder, e.g. `site/content-types/my-content-type-name/my-content-type-name.svg`

Response Filters

Response filters are scripts, similar to controllers, that allow customizing or adapting the response of page controllers. Notice that this actually applies to pages from any application added to the site.

The page and component controllers are processed during rendering and then the response filters will be executed afterward.

To add a response filter, create a folder `site/filters` in the application and place a `[filter-name].js` file within this folder. A filter must export a method named `responseFilter`. This method receives the request and response objects as parameters and must return a response object (see *HTTP Controllers*).

Here is an example of a `[filter].js` file:

Listing 3.5: site/filters/trackingScript.js

```
exports.responseFilter = function (req, res) {
    var trackingScript = '<script src="http://some.site/js/tracking.js"></script>';

    var bodyEnd = res.pageContributions.bodyEnd;
    if (!bodyEnd) {
        res.pageContributions.bodyEnd = [];
    }
    if (typeof bodyEnd == 'string') {
        res.pageContributions.bodyEnd = [ bodyEnd ];
    }
    res.pageContributions.bodyEnd.push(trackingScript);

    if (req.params.debug === 'true') {
        res.applyFilters = false; // skip other filters
    }

    return res;
};
```

In addition, the filter must be declared in the `site.xml` descriptor by adding a `<response-filter>` element within the `<filters>` element, with `@name` and `@order` attributes.

Listing 3.6: site/site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<site>
  <filters>
    <response-filter name="trackingScript" order="10"/>
  </filters>
  <x-data mixin="html-meta"/>
  <config>
    <input type="ContentSelector" name="profiles">
      <label>Profiles folder</label>
      <occurrences minimum="0" maximum="1"/>
      <config>
        <relationshipType>system:reference</relationshipType>
        <allowContentType>base:folder</allowContentType>
      </config>
    </input>
  </config>
</site>
```

Response filters may change any of the values of the response object, that includes: HTTP status code, response body, HTTP headers, cookies and page contributions.

It is also possible to return the response object received without any changes.

Execution order

An application can contain multiple filters declared in `site.xml`. Multiple applications can be selected for a Site. When a page is rendered, all the filters declared in all the applications selected for the site will be executed. The order in which the filters are executed depends on the filters `order` (as defined in `site.xml`) and the order of the applications configured on the Site.

The filters with a **lower order** will be executed **first**. In case there are several filters with the same `order` number, the position of the applications (as configured for a Site in Content Studio) determines the order of execution.

The filter-chain execution can be interrupted from either a controller or a filter by setting the `applyFilters` field in the response. When this value is set to `false` all of the remaining filters will be skipped. (See *HTTP Response*).

Controller Mappings

Controller mappings allow the creation of HTTP endpoints that are bound to a combination of a URL pattern and/or content property.

The mappings can be defined in the `site.xml` file (see *Site Descriptors*).

```
<site>

  <mappings>
    <mapping controller="/site/foobar/api.js" order="10">
      <!-- URL relative to site path, e.g.:  [site-path]/api/v42/action -->
      <pattern>/api/v\d+/.*</pattern>
    </mapping>

    <!-- handle fragment content-type -->
    <mapping controller="/site/pages/default/default.js">
      <match>type:'portal:fragment'</match>
    </mapping>
  </mappings>

</site>
```

Multiple mappings can be set for a site.

Each controller mapping has the following properties:

- **Controller**: application path to the JavaScript controller that will handle the request. This attribute is required.
- **Content match**: matching condition to evaluate on the content in the requested path. This element is optional.
- **URL pattern**: regular expression to match against the request URL. This element is optional.
- **Order**: determines which controller will be executed in case there is more than one that matches. The mapping with lowest order value will be executed. Default is `50`.

Controller mappings can be used for rendering **fragments**, i.e. contents of type `portal:fragment`, with custom styles and layout. That way they will be consistent with the other page components in the application (see *Portal content types*). This is useful for editing fragments from *Page Editor* in *Content Studio*.

Controller

The controller is specified with the `controller` attribute in a `<mapping>` element. A controller handles requests in the same way a *Page* or a *Part* controller does. The controller is a JavaScript file located in the application.

Unlike page and part controllers, a mapping controller is not required to be placed in a specific directory. In fact, an existing page or service controller can also be used as the controller for a mapping.

The controller must export a method for each type of HTTP request that should be handled. The handle method receives the request object as a parameter and returns the response object (see *HTTP Controllers*).

Example: `<mapping controller="/site/controller/foo.js">`

Content match

The `<match>` element specifies a condition related to the content corresponding with the requested URL path.

The condition takes the form of a **property path** followed by a **semicolon**, and then an **expected value**.

The property path can be one of the content properties (`_id`, `_name`, `_path`, `type`, `displayName`, `hasChildren`, `language`, `valid`) or a custom property in the `data` or `x` part.

Examples:

```
<match>type:'portal:fragment'</match>
<match>_path:'/features/.*'</match>
<match>data.employee.type:'developer'</match>
<match>data.product.category:42</match>
<match>x.com-enonic-myapp.menuItem.show:true</match>
```

The expected value can be either a regular expression to match the property value, or simply a string, number or boolean (`true | false`).

URL pattern

The `<pattern>` element specifies a regular expression to be matched against the request URL. The part of the URL that is taken into account for the matching is the path relative to the site where the application is configured. For example, if a site has a content path */mysite*, then the pattern `<pattern>/api/.*</pattern>` will match with requests with URL ending in */mysite/api/.**

If the pattern contains the question mark `?` character, the URL to match will also include query parameters. The query parameters will be normalized so they are always in alphabetical order.

For example the pattern `<pattern>/api\?category=foo&amp;key=\d+</pattern>` will match with both:

/api?category=foo&key=123 and also with */api?key=123&category=foo*

Note than in the previous example the question mark character `?` is escaped with a backslash because the question mark is a quantifier in regular expressions. And also the ampersand character `&` needs to be XML-escaped because the pattern string is in an XML. Another alternative to XML-escape is to wrap the string in a CDATA block, as in the example below.

The protocol, host and port are not involved in the matching.

The pattern element may also contain an `invert` attribute to indicate that the result of evaluating the regular expression should be negated: `<pattern invert="true">`

The pattern string must be a valid Java regular expression.

Examples:

```
<pattern>/api/.*\.json</pattern>
<pattern>/.*</pattern>
<pattern><![CDATA[/endpoint\?bar=\d+&foo=.*]]></pattern>
<pattern>/endpoint\?bar=\d+&amp;foo=.*</pattern>
<pattern invert="true">/section/.*</pattern>
```

Page

A page component is the most basic building block of a site. Each page component must have a JavaScript controller file and optionally an XML descriptor and an HTML view file. These files can define regions in the page where parts and layouts may be added, or they can define a simple page without any compositions. Page components can be added to content individually through the Content Studio interface or they can be used to create page templates that automatically render supported content types.

Any number of page templates can be created from a single page component. Thanks to the magic of page templates, even very large sites will typically have very few page components–perhaps one for all the HTML pages and one for RSS pages.

Pages should be placed in the folder `site/pages/[page-name]`

Descriptor

The page descriptor is an XML file that is used to define regions and custom input fields for page configuration. If a page does not require regions or configuration options then the descriptor may be omitted.

The file must be named `[page-name].xml`. For example, if a page component is named "default" then the file must reside at `site/pages/default/default.xml`.

```
<page>
  <display-name>My first page</display-name>
  <config>
    <!-- input fields... -->
  </config>
  <regions>
    <region name="top"/>
    <region name="bottom"/>
  </regions>
</page>
```

display-name A simple human readable display name.

config The `config` element is where input fields are defined for configurable data that may be used on the page.

regions This is where regions are defined. Various component parts can be dragged and dropped into regions on the page.

Controller

A page controller handles requests to the page. The controller is a required file written in JavaScript and must be named `[page-name].js`. A controller exports a method for each type of HTTP request that should be handled. The handle method has the request object as a parameter and returns the response object (see *HTTP Controllers*).

```
// Handles a GET request
exports.get = function(req) {}

// Handles a POST request
exports.post = function(req) {}
```

Here's a simple controller that acts on the `GET` request method.

```
exports.get = function(req) {

  return {
    body: '<html><head></head><body><h1>My first page</h1></body></html>',
    contentType: 'text/html'
  };

};
```

Render-view

If you feel like concatenating strings to create an entire web page is a little too much hassle, Enonic XP also supports views. A view is rendered using a rendering engine; we currently support XSLT, Mustache and Thymeleaf rendering engines. This example will use Thymeleaf.

To make a view, create a file `my-first-page.html` in the `view` folder.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <h1>My first page, with a view!</h1>
  </body>
</html>
```

In our `[page-name].js` file, we will need to parse the view to a string for output. Here is where the Thymeleaf engine comes in. Using the Thymeleaf rendering engine is easy; here is how we do it.

```
var thymeleaf = require('/lib/xp/thymeleaf');

exports.get = function(req) {

  // Resolve the view
  var view = resolve('/site/view/my-first-page.html');

  // Define the model
  var model = {
    name: "John Doe"
  };

  // Render a thymeleaf template
  var body = thymeleaf.render(view, model);

  // Return the result
  return {
    body: body,
    contentType: 'text/html'
  };

};
```

Unlike controllers and descriptors, view files can reside anywhere in your project and have any valid file name. This allows for code reuse as multiple page components can share the same view. If the view file is in the same folder as the page controller then it can be resolved with only the file name `resolve(’file-name.html’)`. Otherwise, the full path should be used, starting with a ‘/’ as in the example above.

Dynamic-content

We can send dynamic content to the view from the controller via the `model` parameter of the `render` function. We then need to use the rendering engine specific syntax to render it. The controller file above passed a variable called `name` and here is how to extract its value in the view using Thymeleaf syntax.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <h1>My first page, with a view!</h1>
    <h2>Hello <span data-th-text="${name}">World</span></h2>
  </body>
</html>
```

More on how to use Thymeleaf can be found in the official Thymeleaf documentation.

Regions

To be able to add components like images, component parts, or text to our page via the Page Editor drag and drop interface, we need to create at least one region. Regions can be declared in the page descriptor. Each region will be referenced by name.

```
<page>
  <display-name>My first page</display-name>
  <config />
  <regions>
    <region name="main"/>
  </regions>
</page>
```

You will also need to handle regions in the controller.

```
var portal = require('/lib/xp/portal');

// Get the current content. It holds the context of the current execution
// session, including information about regions in the page.
var content = portal.getContent();

// Include info about the region of the current content in the parameters
// list for the rendering.
var mainRegion = content.page.regions["main"];

// Extend the model from previous example
var model = {
    name: "Michael",
    mainRegion: mainRegion
};
```

To make the Page Editor understand that an element is a region, we need an attribute called `data-portal-component-type` with the value `region` in our HTML.

```
<!DOCTYPE html>
<html>
  <head>
  </head>
  <body>
    <h1>My first page, with a view!</h1>
    <h2>Hello <span data-th-text="${name}">World</span></h2>
    <div data-portal-region="main">
      <div data-th-each="component : ${mainRegion.components}" data-th-remove="tag">
        <div data-portal-component="${component.path}" data-th-remove="tag" />
      </div>
    </div>
  </body>
</html>
```

We can now use the Page Editor drag and drop interface to drag components into our page.

Part

A part is a building block that can be placed in a *region* on a page or layout. As with pages, each part is composed of a JavaScript controller, an XML descriptor and an HTML view.

The part descriptor and controller files must be placed in the folder `site/parts/[part-name]`

Descriptor

The part **descriptor** is where input fields are defined for custom configuration of the part. The descriptor is not required if the part does not need any custom configuration. Parts cannot contain regions.

When used, the descriptor file must have the same name as the *part* folder that contains it `site/parts/[part-name]/[part-name].xml`:

```
<part>
  <display-name>My favorite things</display-name>
  <config>
    <field-set name="things">
      <label>Things</label>
      <items>
        <input type="TextLine" name="thing">
          <label>Thing</label>
          <occurrences minimum="0" maximum="5"/>
        </input>
      </items>
    </field-set>
  </config>
</part>
```

Controller

To drive this part, we will also need a **controller**. The controller typically uses library functions to get content and/or configurations and prepare data which it passes to the view file for dynamic rendering.

```
site/parts/[part-name]/[part-name].js
```

```
var portal = require('/lib/xp/portal'); // Import the portal functions
var thymeleaf = require('/lib/xp/thymeleaf'); // Import the thymeleaf render function

// Handle GET requests
exports.get = function(req) {

  // Find the current component from request
  var component = portal.getComponent();

  // Find a config variable for the component
  var things = component.config["thing"] || [];

  // Define the model
  var model = {
    component: component,
    things: things
  };

  // Resolve the view
  var view = resolve('/site/view/my-favorite-things.html');

  // Render a thymeleaf template
  var body = thymeleaf.render(view, model);

  // Return the result
  return {
    body: body,
    contentType: 'text/html'
  };

};
```

View

A part view defines the markup for the part component. The `things` parameter is basically just JSON data passed from the controller and we can iterate over it easily in Thymeleaf and print its values.

```
<section>
  <h2>A list of my favorite things</h2>
  <ul class="item" data-th-each="thing : ${things}">
    <li data-th-text="${thing}">A thing will appear here.</li>
  </ul>
</section>
```

The part can now be added to the page via drag and drop. You will be able to configure the part in the *context window* in live-edit.

Important: The HTML generated for the part view must have a single root element.

Layout

Layouts are used in conjunction with `regions` to organize the structure of the various component parts that will be placed on the page via Page Editor drag and drop. Layouts can be dropped into the page `regions` and then `parts` can be dragged into the layout. This allows multiple layouts (two-column, three-column, etc.) on the same page and web editors can change things around without touching any code. Making a layout is similar to making pages and part components. Layouts cannot be nested.

Layout contains - like pages and parts - a descriptor, a controller and a view, and should be placed in the folder `site/layouts/[layout-name]`

Descriptor

The layout descriptor defines regions within the layout where parts can be placed with the Page Editor. The file must be named `[layout-name].xml`.

```
<layout>
  <display-name>70/30</display-name>
  <config/>
  <regions>
    <region name="left"/>
    <region name="right"/>
  </regions>
</layout>
```

Controller

The layout controller composes the view of the layout based on HTTP requests. The file must be named `[layout-name].js`.

```
var portal = require('/lib/xp/portal');
var thymeleaf = require('/lib/xp/thymeleaf');

exports.get = function(req) {

  // Find the current component.
  var component = portal.getComponent();

  // Resolve the view
  var view = resolve('./layout-70-30.html');

  // Define the model
  var model = {
    leftRegion: component.regions["left"],
    rightRegion: component.regions["right"]
  };

  // Render a thymeleaf template
  var body = thymeleaf.render(view, model);

  // Return the result
  return {
    body: body,
    contentType: 'text/html'
  };

};
```

View

A layout view defines the markup for the layout component. The sample view below is created in Thymeleaf, but it could be created in any view engine that is supported.

```
<div class="row">
  <div data-portal-region="left" class="col-sm-8">
    <div data-th-each="component : ${leftRegion.components}" data-th-remove="tag">
      <div data-portal-component="${component.path}" data-th-remove="tag" />
    </div>
  </div>

  <div data-portal-region="right" class="col-sm-4" >
    <div data-th-each="component : ${rightRegion.components}" data-th-remove="tag">
      <div data-portal-component="${component.path}" data-th-remove="tag" />
    </div>
  </div>
</div>
```

Important: The HTML generated for the layout view must have a single root element.

Styling

For a layout to have any meaning, some styling must be applied to the view. The desired CSS should be placed in the `/assets` folder of the application, and included in the page where the layout should be supported. For example, the view my-first-page.html supports Bootstrap layouts:

```
<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <link data-th-href="${portal.assetUrl({'_path=css/bootstrap.min.css'})}" href="../assets/css/boot
</head>
```

Fragment

Fragments are **reusable page components**. A fragment can be created from an instance of one of the other four page component types: *Part*, *Layout*, *Text* or *Image*.

When created, the fragment will be stored in a content of type `portal:fragment`, including the component's config. This content can then be edited independent of where it is included.

A *fragment* component can be inserted in any other page or layout from the same site with the *Page Editor*. A *fragment* component acts as a placeholder for the referenced fragment content. At the moment of rendering the page, the fragment is replaced by the specific component from which the fragment was created.

Unlike pages, parts, and layouts, fragment components do not require creating a descriptor in the application, since they are only placeholders for other components.

View

Content of type `portal:fragment` can be edited in the *Page Editor* as if it was a regular page. The components will be rendered inside a plain empty HTML page by default. But it is also possible to create a custom renderer with the application styles and layout.

To create a custom renderer for the `portal:fragment` content, configure a controller mapping in `site.xml`:

```
<site>

  <mappings>
    <mapping controller="/site/pages/default/default.js">
      <match>type:'portal:fragment'</match>
    </mapping>
  </mappings>

</site>
```

This mapping indicates that it will handle content of type `portal:fragment`.

The mapping points to a controller that will handle the rendering. The controller is just like any other page or part controller (See *HTTP Controllers* and *Controller Mappings* for details).

The custom view that will render `portal:fragment` content is like any other *Page* view, with one difference. A page has *regions*, but a fragment has only a component without any regions defined. To specify the component to be rendered we need to specify `data-portal-component="fragment"` instead of the component path.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="myStyles.css" type="text/css"/>
</head>
<body>
    <h1>Page header</h1>

    <!-- render Fragment component -->
    <div data-portal-component="fragment" data-th-remove="tag"/>

    <footer>Copyright © Enonic AS</footer>
</body>
</html>
```

Page Contributions

Page contributions are fragments of HTML that a component (**page**, **part**, **layout**) or **macro** can contribute to the page in which it is contained. The idea is to allow components to add JavaScript or CSS stylesheets globally in the page, although it is not restricted to scripts or styles.

Page contributions help with solving 2 problems:

- Allow components to insert scripts or styles in specific positions in the page where it is often required.

  For example, a component providing web analytics might require that a script is inserted at the end of the page `<body>`. Or a stylesheet needed for a component must be inserted in the `<head>` tag.

- Avoid duplicating script libraries or stylesheets required for a component. Even if the same component is included multiple times in a page, the library script contributed will only be added once.

Any page, part, layout or macro controller can contribute content to the page. The values from all component contributions will be included in the final rendered page. Duplicated values will be ignored. There are four positions where contributed content can be inserted in the page:

- `headBegin`: After the `<head>` opening tag.
- `headEnd`: Before the `</head>` closing tag.
- `bodyBegin`: After the `<body>` opening tag.
- `bodyEnd`: Before the `</body>` closing tag.

```
{
  "body": "<html>...</html>",
  "pageContributions": {
    "headEnd": "value",
    "bodyEnd": [
      "value1", "value2"
    ]
  }
}
```

Some remarks:

- All the `pageContributions` fields are optional. The `pageContributions` object is optional and each property inside is optional.
- The value for a contribution can be a string or an array of strings.
- The values are unique within an injection point (or tag position). If the same string is contributed from different parts, or from the same part that exists multiple times in the page, the value will only be inserted once. E.g. if two parts include a script for jQuery, it will be included once. But if one part is contributing to `headBegin` and another one contributes the same value to `bodyEnd`, then it will be inserted in both places.
- If the tag does not exist in the rendered page, the value is ignored. I.e. if there is no `<head>` tag, the contributions to `headBegin` and `headEnd` will just be ignored.
- The contributions are inserted in a post-processing step during rendering. That means that there will not be any processing of Thymeleaf tags or similar. Contributions are treated as plain text.

Warning: Avoid String - Array conflicts in response filters

When there is a single contribution for a placeholder, like headEnd, it will be a string. But adding to it in a response filter will require an array. It is best to always check if it is an array before adding a value.

```
var headEnd = res.pageContributions.headEnd;
if (!headEnd) {
    res.pageContributions.headEnd = [];
} else if(typeof headEnd == 'string') {
    res.pageContributions.headEnd = [ headEnd ];
}
res.pageContributions.headEnd.push(myCodeForHeadEnd);
```

Error Handling

Enonic XP enables you to displaying nice custom error pages for your site.

Create the following folder in your project `src/main/resources/site/error` and place an `error.js` within it. The file follows the same pattern as controllers and filters. If certain methods are implemented and exported, they will be executed in case of errors during rendering.

If an error occurs during processing - the system looks for an `error.js` script within the relevant application - sites specifically it will go through all applications added to the site (in order).

If an `error.js` script is found, it looks for an exported method named `handleXXX` where `XXX` is the HTTP status-code of the error. If not found, it will try to find the generic error method `handleError` instead.

Here is an example of an `error.js` file:

```
var thymeleaf = require('/lib/xp/thymeleaf');

var view404 = resolve('page-not-found.html');
var viewGeneric = resolve('error.html');

exports.handle404 = function (err) {
    var body = thymeleaf.render(view404, {});
    return {
        contentType: 'text/html',
        body: body
    }
};

exports.handleError = function (err) {
    var debugMode = err.request.params.debug === 'true';
    if (debugMode && err.request.mode === 'preview') {
        return;
    }

    var params = {
        errorCode: err.status
    };
    var body = thymeleaf.render(viewGeneric, params);

    return {
        contentType: 'text/html',
        body: body
    }
};
```

The input parameter for the `handleXXX` and `handleError` functions is an error JSON object containing the status code, error message, Java Exception object, and the original `request` object:

```
{
    "status": 404,
    "message": "Some error message",
    "exception": "<the actual exception object in Java>",
    "request": "<original request JSON>"
}
```

The expected returned value for the function is a `response` object (see *HTTP Controllers*).

The error processing logic will try every handle-function in application order until it can get a result (not `undefined` or `null`). This means that an error function can decide to not handle a specific error and let the next one deal with it. If no result is returned by any function, it will be eventually handled by the internal error page.

Also note that if an error occurs inside the custom-error code, then the internal error page will be rendered.

Macros

Warning: Macros are experimental.

Macros are instructions that allow adding extra functionality or include dynamic content in the Html Editor.

The Html Editor is used when editing *HtmlArea* input fields, or while editing a *Text component* in Page Editor.

There are two built-in macros included in XP. But its real power comes from macros provided by applications. When an application that contains macros is added to a site, they will be available for any HtmlArea or Text component inside the site.

Macro instruction

A macro instruction is similar to an HTML or XML tag but using square brackets instead of angle brackets. It has a name, a set of attributes, and optionally a body.

```
[macroname attrib1="value1" attrib2="value2"] body [/macroname]
[macroname attrib1="value1" attrib2="value2"/]
```

Macro instructions can be added anywhere in an HTML page, usually inside a content's HtmlArea field.

During the rendering of the page the macros are resolved and executed. Then the result from executing the macro replaces the instruction text. In addition, a macro can also add styles or scripts to the page, by setting *Page Contributions* in its response.

A user can add macro instructions by typing the square bracket tags, as the examples above. But more frequently it will click on the *Insert macro* button and select one of the macros available.

Descriptor

A macro descriptor is an xml file that allows assigning a user-friendly name, and a description to the macro. It also has a configuration to define the types and names of the macro parameters.

display-name A simple human readable display name.

description A description to show in the Insert macro dialog in Content Studio.

config The `config` element is a form where each input element corresponds to a macro parameter. The macro body is represented with an input named `"body"`.

Note: The config form does not support nested elements, so *Item Sets* are not allowed in the macro config form. Also the *HtmlArea* input type is not allowed in the config form, since it may contain macros itself.

Its path follows the pattern `site/macros/<macroName>/<macroName>.xml`

```
<macro>
  <display-name>Current user</display-name>
  <description>Shows currently logged user</description>

  <form>
    <input name="defaultText" type="TextLine">
      <label>Text to show if no user logged in</label>
    </input>
  </form>
</macro>
```

Although not strictly required, it is recommended to create a descriptor, as it provides the required details for adding macros through the UI in Content Studio.

Controller

The functionality of a macro is implemented in a JavaScript controller, inside an application.

Its path follows the pattern `site/macros/<macroName>/<macroName>.js`

A macro controller must export a single `macro` function that takes a `context` parameter and returns a response object (see *HTTP Response*).

The `context` parameter is a Javascript object with the following properties:

name a string containing the macro name.

body a string containing the body of the macro instruction.

params an object with key-value pairs containing the macro parameters. The values are the strings from the macro instruction attributes.

document a string with the HTML document that contains the current macro. The document contains the raw source HTML, before any macro instructions have been executed, and before image or content URLs have been resolved. The document is only an input parameter to the macro, it cannot be modified.

request the request object.

```
// Example usage: [currentUser defaultText="Anonymous"/]
var authLib = require('/lib/xp/auth');
var portalLib = require('/lib/xp/portal');

exports.macro = function (context) {
    var defaultText = context.params.defaultText;

    var user = authLib.getUser();
    var body = '<span>' + (user ? user.displayName : defaultText) + '</span>';

    var doc = context.document; // HTML document containing the current macro
    var lineCount = doc.split(/\r\n|\r|\n/).length;
    if (lineCount <= 1) {
        return {
            body: ''
        }
    }

    return {
        body: body,
        pageContributions: {
            headEnd: [
                '<link href="' + portalLib.assetUrl({path: 'css/current-user.css'}) + '"/>'
            ]
        }
    }
};
```

Note that only the `body` and `pageContributions` fields of the response are relevant for macro controllers.

Tip: A macro controller can also use libraries, like any other JavaScript controller.

Built-in macros

There are currently 2 built-in macros that are included in XP and available for any site:

disable The contents (body) of this macro will not be evaluated as macros. That allows rendering another macro instruction as text without executing it. It is useful for documenting macros, for example. This macro has no parameters.

embed It allows embedding an *<iframe>* element in an HTML area. This is a generic way for embedding content from an external source (e.g. YouTube videos). This macro has no parameters.

Examples:

```
[disable]Example of macro instruction: [myMacro param1="value1"/][/disable]

[embed]<iframe src="https://www.youtube.com/embed/cFfxuWUgcvI" allowfullscreen></iframe>[/embed]
```

Note: A macro may optionally have its own specific icon. The icon can be assigned to the macro by adding a PNG or SVG file with the same name, in the macro folder, e.g. `site/macros/myMacro/myMacro.svg`

Localization

Enonic XP provides a standard approach to code localizations, simply by adding resource bundles to your applications, and actively using the `localize` functions in “controllers” and “views”. Content localization currently requires building separate structures - we are working on process tools to simplify this in future releases.

To see how this is used in a controller, see `lib-i18n` in *Javascript Libraries*.

Resource Bundle

The resource-bundle consists of a collection of files containing the phrases to be used for localization. The resource-bundle should be placed in a folder named `i18n` under the application `site` folder.

Each locale to be localized should be represented by a single resource, e.g this could be a structure for an app supporting

- 'English' (default)
- 'English US'
- 'Norwegian'
- 'Norwegian Nynorsk'

```
site/i18n/phrases.properties
site/i18n/phrases_en_us.properties
site/i18n/phrases_no.properties
site/i18n/phrases_no_nn.properties
```

The filename of a resource determines what locale it represents:

```
phrases[_languagecode][_countrycode][_variant].properties
```

Caution: The filename should be in lowercase.

The `languagecode` is a valid ISO Language Code. These are the two-letter codes as defined by ISO-639. You can find a full list of these codes at a number of sites, such as: http://www.loc.gov/standards/iso639-2/php/English_list.php.

The `countrycode` is a valid ISO Country Code. These are the two-letter codes as defined by ISO-3166. You can find a full list of these codes at a number of sites, such as: http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/list-en1.html

A sample phrases.properties file would look like this:

```
user.greeting = Hello, {0}!
complex_message = Good to see you. How are you doing?
message_url = http://localhost:8080/{0}
message_multi_placeholder = My name is {0} and I live in {1}
message_placeholder = Hello, my name is {0}.
med_\u00e6_\u00f8_\u00e5 = This contains the norwegian characters æ, ø and å
```

Placeholders

Placeholders are marked with `{<number>}`. The given number corresponds with the function argument named `values` and the placement of the parameter. See below for an example.

Encoding and special characters

The encoding of localization resource bundle files must be ISO-8859-1, also known as Latin-1. All non-Latin-1 characters *in property-keys* must be entered using Unicode escape characters, e.g u00E6 for the Norwegian letter 'æ'. The values may also be encoded, but this is not required.

Resolving locale

A locale is composed of language, country and variant. Language is required, country and variant are optional.

The string-representation of a locale is:

```
LA[_CO][_VA]
```

where

- `LA` = two letter language-code
- `CO` = two letter country-code
- `VA` = two letter variant-code.

The variant argument is a vendor or browser-specific code. For example; WIN for Windows, MAC for Macintosh, and POSIX for POSIX. Where there are two variants, separate them with an underscore, and put the most important one first. For example, a Traditional Spanish collation might construct a locale with parameters for language, country and variant as: “es”, “ES”, “Traditional_WIN”.

When a localize function is called upon, a locale is resolved to decide which localization to use.

The following is considered, in this order:

- Given as argument to function
- Language specified on site

Finding best match

When localizing a keyword, a best match pattern will be applied to the resource bundle to select the localized phrase. If the locale for a request is resolved to “en-US”, these files will be considered in given order:

- `phrases_en_us.properties`
- `phrases_en.properties`
- `phrases.properties`

If the locale for a request would have been resolved to `en`, the `phrases_en_us.properties` file would not have been considered when localizing a keyword.

If the locale does not match a specific file, the default `phrases.properties` will be used.

If no matching localization key is found in any of the files in a bundle, a default `NOT_TRANSLATED` will be displayed.

Node Domain

At the core of Enonic XP lies a distributed data storage - all persistent items in Enonic XP are stored as nodes.

Overview

Years of experience has taught us that traditional approaches to data storage (read SQL) are unsuited for the common requirements of modern cloud-based applications and platforms. A key goal of Enonic XP was to deliver a complete stack - virtually eliminating complex dependencies to 3rd party applications, and minimize requirements to infrastructure.

With the growing popularity of various so-called NoSQL (Not Only SQL) solutions, we evaluated many different technologies and found great inspiration in the following:

Git

- (+) Cherry picking

  - (+) Branching
  - (+) Pull requests
  - (-) Performance search
  - (-) Granularity of access (all or nothing)

Java Content Repository

  - (+) Hierarchy
  - (+) Granularity
  - (+) Feature set
  - (+) Unstructured
  - (-) Performance
  - (-) Complexity (not document oriented)
  - (-) Attached data model
  - (-) Requires additional storage backend

Elasticsearch

  - (+) Document oriented
  - (+) Scalability
  - (+) Performance
  - (+) Search
  - (+) Aggregations
  - (-) Search engine, not a database
  - (-) No blob support
  - (-) No security
  - (-) Creates schemas

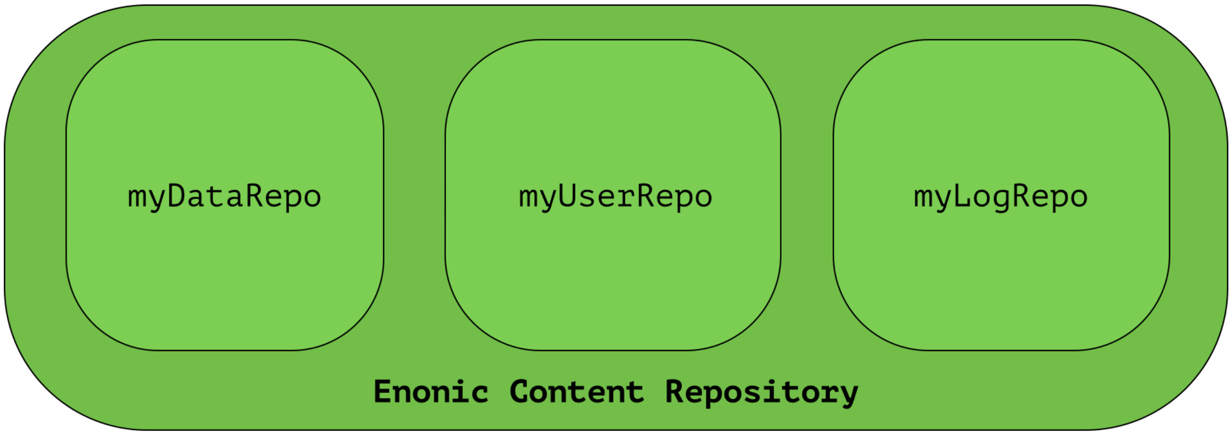
We were unable to find any single solution that was sufficiently simple and included our desired feature set - so we decided to build our own; the Enonic Content Repository.

The Enonic Content Repository is a place where you can store data, or more specific: *Nodes*.

It is built on top of Elasticsearch and exposes many of it's capabilities in search and aggregations and scalability - but in addition, provides the following capabilities:

- Hierarchical storage model
- Versioning support
- Complete Access Control and security model
- Blob support - using shared filesystem and append-only approach
- Repository and Branch concepts for content staging
- Schemaless - Add any property you like, at any time
- Rich set of value types (*HTMLPart*, *XML*, *Binary*, *Reference* etc..)
- SQL-like query syntax

The Enonic Content Repository itself contains one or more separate repositories based on the application need. For instance, an application could demand a setup having three repositories - one for application data, one for users and one for logging:



The reasons for having several separated repositories are many, and explained in detail in the *Repository* below.

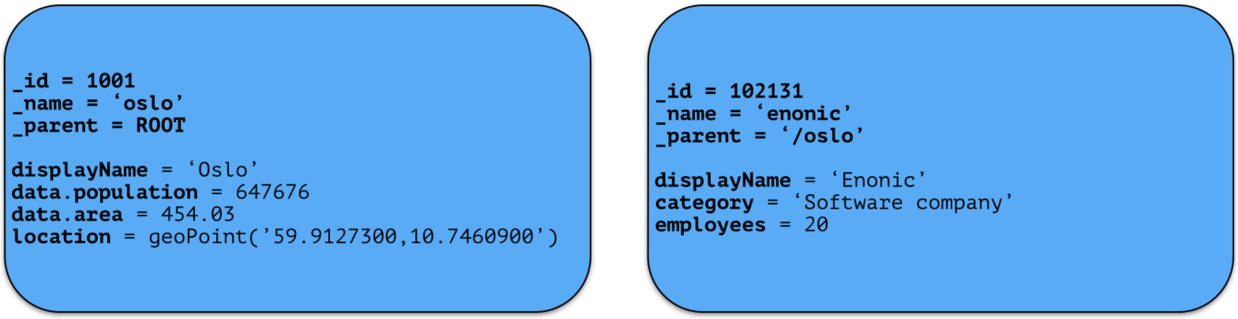
Nodes

A Node represents a single storable entity of data. It can be compared to a "row" in sql, or a "document" in document oriented storage models. Nodes are, as mentioned in the previous section, stored in a repository.

Every node has:

- a name
- a parent-reference
- an id
- a timestamp
- a (possibly empty) set of *Property* key/value.

Consider two nodes - one node representing the city "Oslo" and another representing the company, "Enonic":



```
_id = 1001
_name = ‘oslo’
_parent = ROOT

displayName = ‘Oslo’
data.population = 647676
data.area = 454.03
location = geoPoint(’59.9127300,10.7460900’)
```

```
_id = 102131
_name = ‘enonic’
_parent = ‘/oslo’

displayName = ‘Enonic’
category = ‘Software company’
employees = 20
```

The nodes have different properties. There is no schema to a node, so a node property value with the same property-name can have different value-types across nodes.

Property

Properties represent a placement of data in a node - following the simple `key = value` pattern. A property has a path. Elements in the path are separated by `.` (dot). Every property has also a type. See the complete list of *Value*

Types.

```
myProperty
data.myProperty
cars.brands.skoda
```

For a property to be able to hold other properties, it has to be of type `Set`. In the above samples, `data`, `cars` and `brands` are properties of type `Set`.

Some characters are illegal in a property key. Here’s a list of illegal characters:

- _ is illegal as the first character, because it is a reserved prefix for *System Properties*.
- `.` is illegal as any character, since it is the path separator.
- `[` and `]` are also illegal as any character. These are used as array index indicators.

Here’s an example of some properties:

```
first-name = "Thomas"
cities = ["Oslo", "San Francisco"]
city.location = geoPoint('37.785146,-122.39758')
person.age = 39
person.birth-date = localDate("1975-17-10")
```

Value Types

At the core of the node domain are value types. Every property to be stored in a node must have a value type. The value type enables the system to interpret and handle each piece of data specially - applying to both validation and indexing.

All value-types support arrays of values. All elements in an array must be of the same value-type.

Below is a complete list of all supported value-types.

String A character string.

> **Index value-type** `String`
>
> **Example** `’myString’`

BinaryReference Reference to a binary object.

> **Index value-type** `String`
>
> **Example** `’my-binary-ref’`

Boolean A value representing `true` or `false`.

> **Index value-type** `String`
>
> **Example** `true`

Double Double-precision 64-bit IEEE 754 floating point.

> **Index value-type** `Double`
>
> **Example** `11.5`

GeoPoint Represents a geographical point, given in latitude and longitude.

> **Index value-type** `GeoPoint`
>
> **Example** `’59.9090442,10.7423389’`

Instant A single point on the time-line.

Index value-type `Instant`

Example `2015-03-16T10:00:02Z`

LocalTime A time representation without timezone.

Index value-type `String`

Example `10:00:03`

LocalDateTime A date-time representation without timezone.

Index value-type `String`

Example `2015-03-16T10:00:02`

Long 64-bit two's complement integer.

Index value-type `Double`

Example `1234`

Reference Holds a reference to other nodes in the same repository.

Index value-type `String`

Example `'0b7f7720-6ab1-4a37-8edc-731b7e4f439e'`

Set A special value type that holds properties as it's value, allowing nested levels of properties, creating tree structures within a single node.

Index value-type N/A

XML Accepts a String containing valid XML.

Index value-type `String`

Example `'<property>myPropertyValue</property>'`

System Properties

To reduce complexity, we explicitly dropped the use of namespaces. Thus, in order to separate system properties from user defined properties, we reserved _ as a starting character for system properties.

Below are the system properties explained.

_id Holds the id of the node, typically generated automatically in the form of a UUID.

_name Holds the name of the node. The name must be unique within its scope (all nodes with same parent).

_parentPath Reference to parent node path.

_path The path is resolved from the node name and parent path.

_timestamp The last change to the node version.

_nodeType Used to create collections for nodes in a repository.

_versionKey The id of the node version.

_state Used for keeping state of a node in a branch.

_permissions_read The principals that have read access.

_permissions_create The principals that have create access.

_permissions_delete The principals that have delete access.

_permissions_modify The principals that have modify access.

_permissions_publish The principals that have publish access.

_permissions_readpermissions The principals that have access to read the node permissions.

_permissions_writepermissions The principals that have access to change the node permissions.

Repository

A repository is a place where nodes can be stored. Data stored in a repository will typically belong to a common domain. Fetches and searches are by default executed against a single repository, so it makes sense to keep data from different domains separated in different repositories. For instance, in the Enonic XP CMS, content and data concerning user management are separated into two repositories. The Content Studio application uses the `cms-repo` repository, and the User Manager application uses the `system-repo` repository.

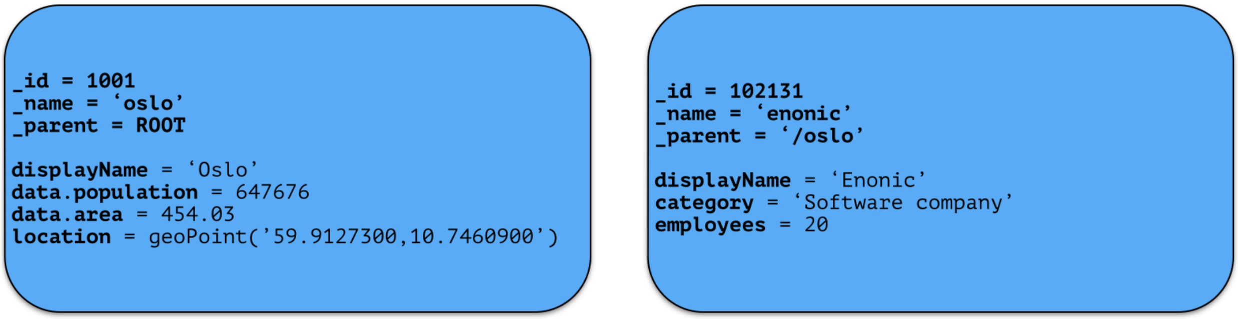
When nodes are stored in the repository, two things happens:

- The node properties are stored in a *Blobstore* as a *node-version*. A node-version is an entity representing the properties of the node, without name, parent and other meta-data.
- The node is inserted into a *branch*. The branch keeps track of a tree-structure referring to node-versions.

A repository will always contain a default branch, called 'master'. If there is more than one branch, API methods are used for resolving diff between and pushing changes from one branch to another.

Node versions and branches

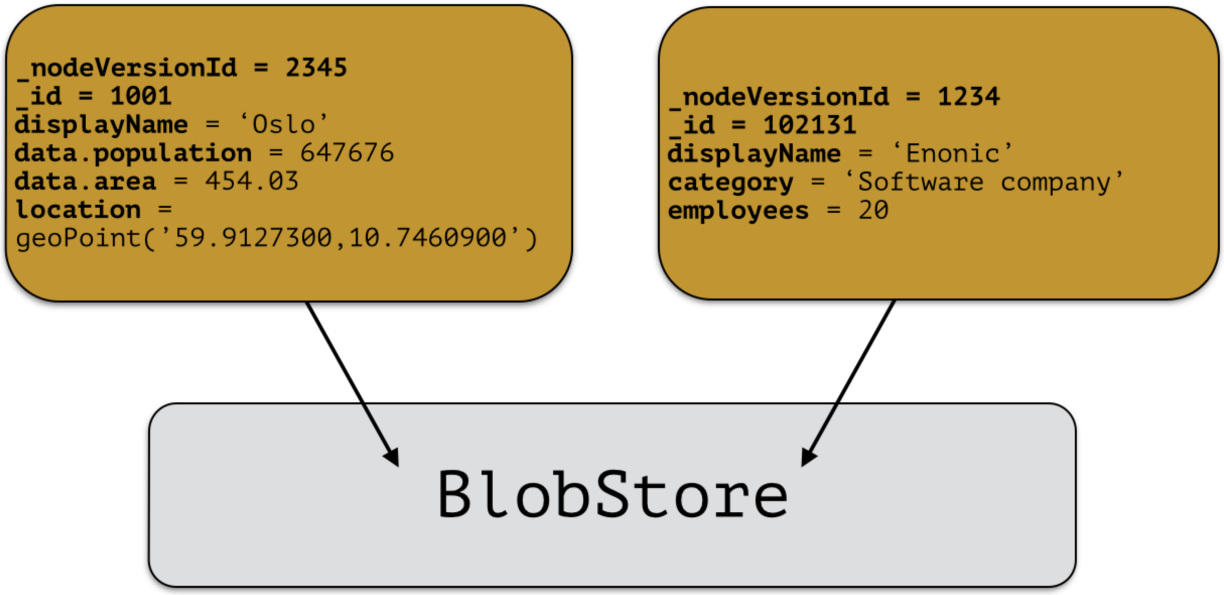
Consider the 'Oslo' and 'Enonic' nodes from earlier sections:



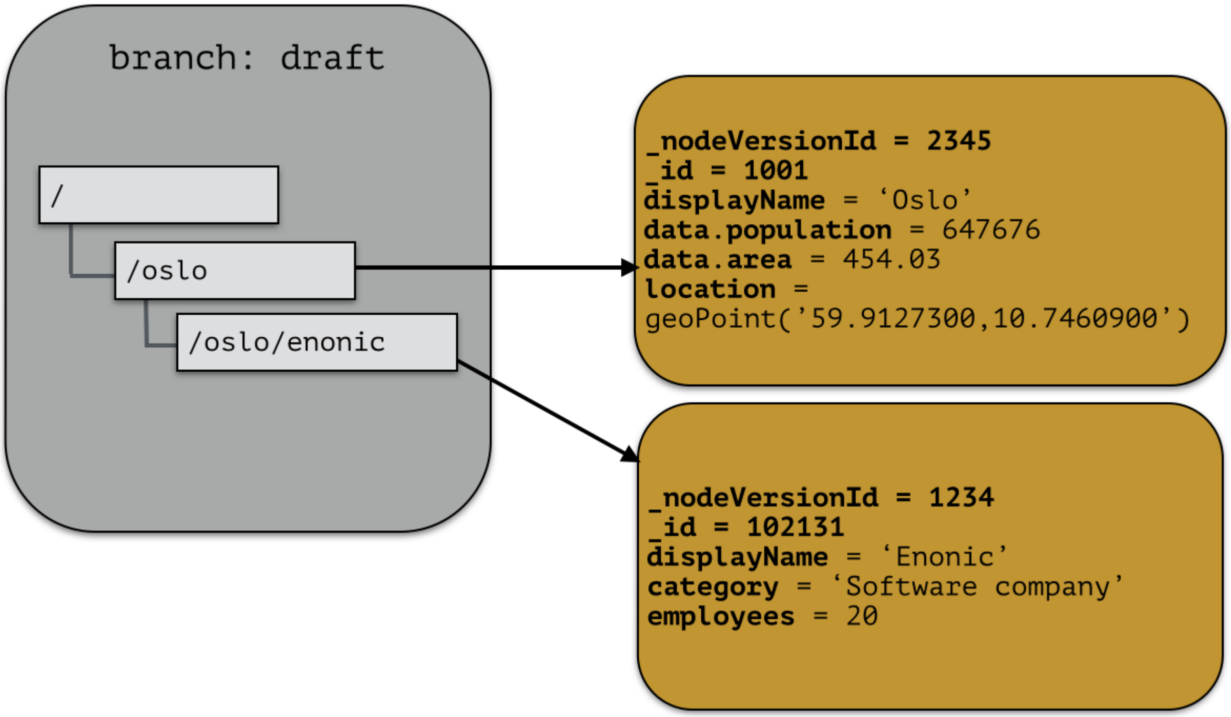
```
_id = 1001
_name = 'oslo'
_parent = ROOT

displayName = 'Oslo'
data.population = 647676
data.area = 454.03
location = geoPoint('59.9127300,10.7460900')
```

```
_id = 102131
_name = 'enonic'
_parent = '/oslo'

displayName = 'Enonic'
category = 'Software company'
employees = 20
```

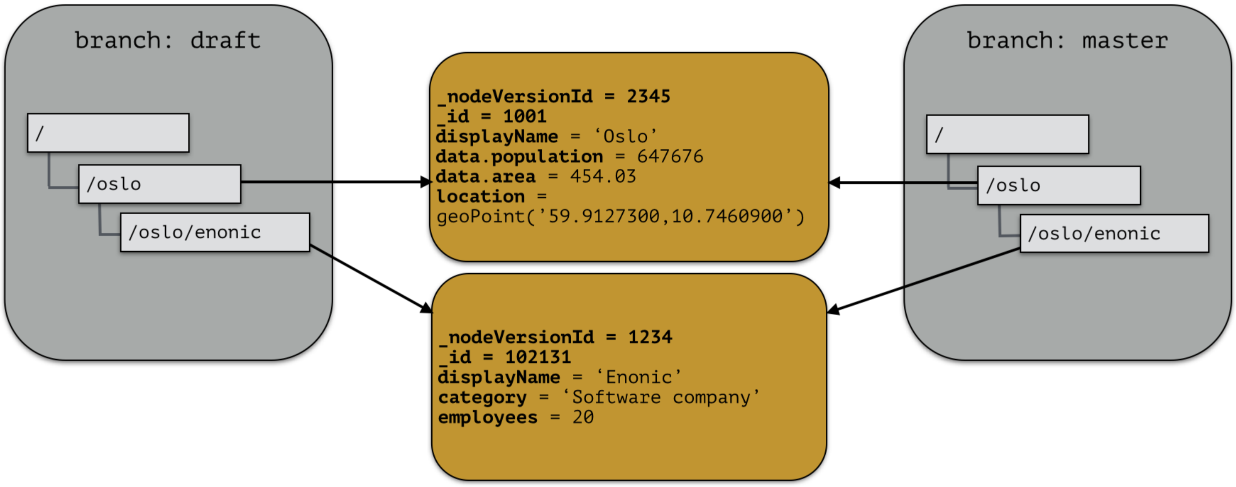
There will be two *node-versions* in the repository stored in the blobstore:



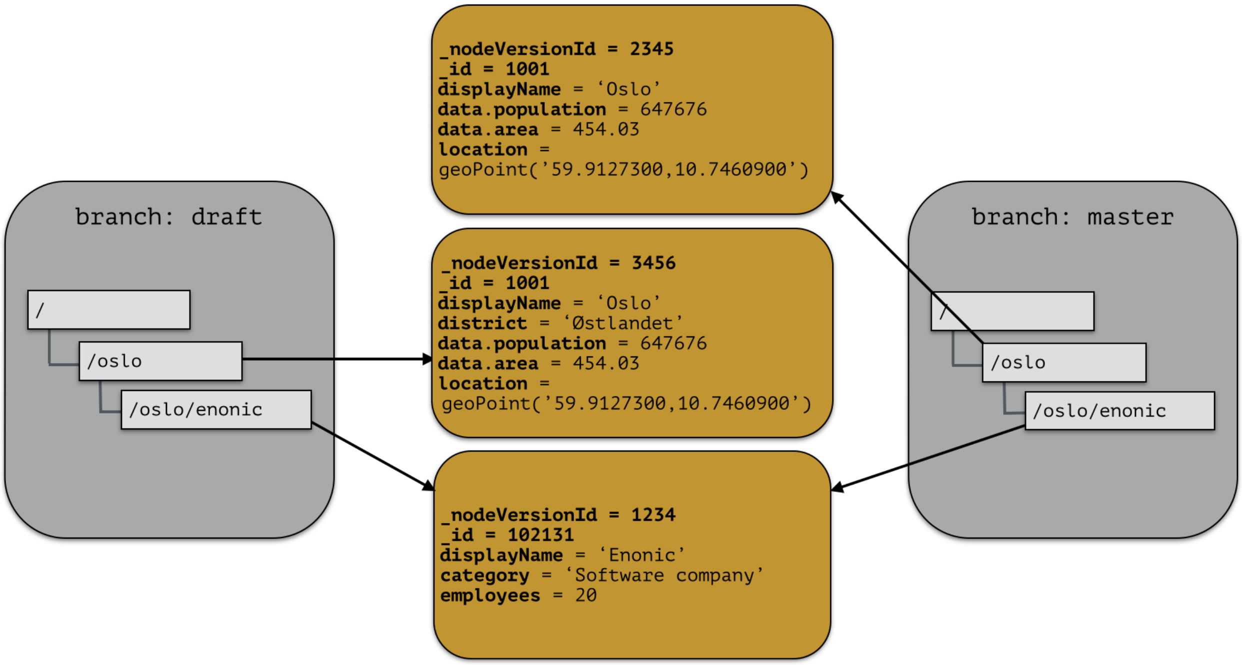
A node-version is a representation of a node's properties. A node-version has no knowledge of name, parent or other meta-data: just the properties of a node. At the same time, the targeted branch (named 'draft' in this example) gets two entries:



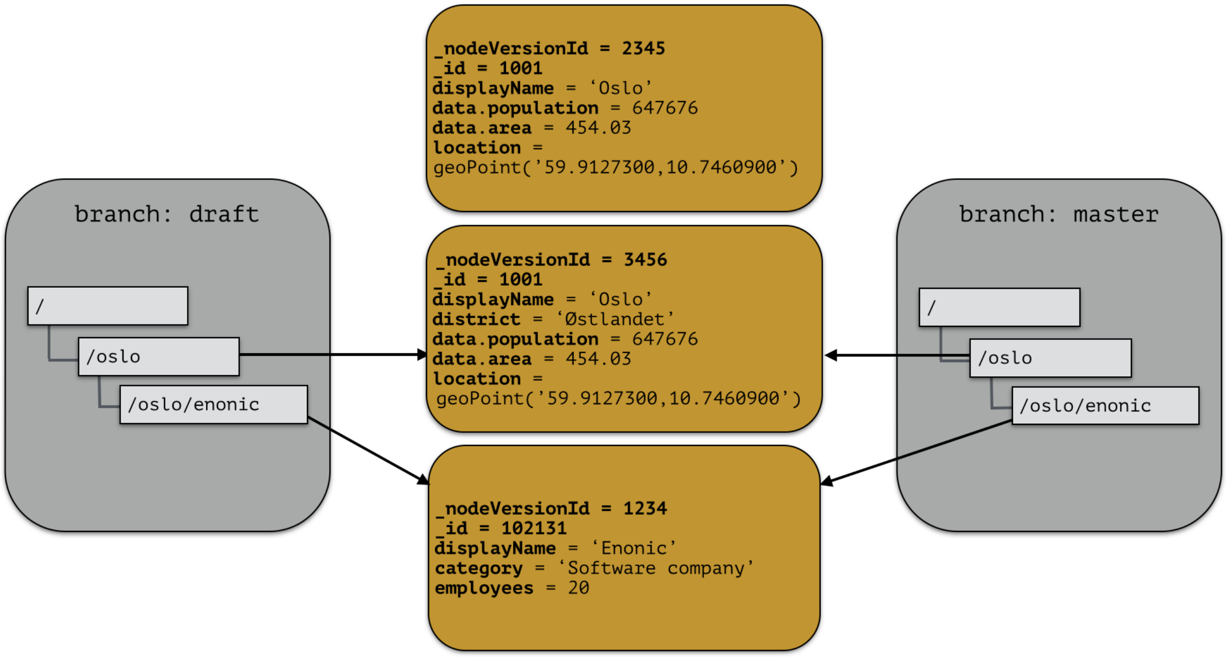
The node-versions are now a part of a tree-structure, based on the node's name and parent. If we *push* the content of branch 'draft' to the default branch 'master', we end up with something like this:



At the moment, there are two branches pointing to the same node-versions. This means that a single node version can exist in several branches with different structures. Now, consider that the 'oslo' - node is updated and stored to the 'draft'-branch, resulting in a new node-version with the same id and an updated pointer from the branch:



The two branches now point to different node-versions of the 'oslo' node. Again, doing a push-operation from 'draft' to 'master' will result in both nodes pointing to the same node-versions:



Repository characteristics

Note: Currently, there is no API for creating and managing repositories, so this information is for reference only at the moment.

A repository should be tuned to match the characteristic of the data you want to store, e.g:

- Expected number of documents
- Read or Write - optimized
- Real-time/near real-time/batch - data availability requirements
- Analyzing
- Archiving strategy

For instance; a log repository will have to be able to handle a large amount of data, but there will probably be no real-time requirements for data to be available. Also, archiving data will be needed to prevent the repository from growing infinitely.

Blobstore

The blobstore is a file system location defaulted to `$XP_HOME/repo/blob`. The blobstore itself is split into one directory for nodes and one for binaries.

Search

This section explains how to find data in the Enonic Content Repository. For a system that deals with storing and retrieving data, a rich search-API is paramount.

Overview

When searching in Enonic XP, you are searching for nodes, or content if working in the context of the CMS content API. This documentation is general and intended for the *Node Domain*, but except for some built-in property-values and the addition of some convenience parameters in the content domain, everything is valid for both domains.

In general, the search-APIs deal with a number of basic parameters:

- start
- count
- query
- filter
- aggregations

Start & count

When searching, the result will contain a number of matching nodes. This number if given by the provided `count` parameter in the query. The result will also contain a value indicating the total number of hits for the search: `total`. The `start` parameter indicates from what position in the result set we should start retrieving results.

Lets consider a search matching 1000 documents. Usually, one does not retrieve all these results at once, but rather a subset of the result - and fetch the next subset of the result if necessary. This type of data-retrieving is called paging.

Typically, one will decide the number of wanted results for each iteration, e.g 100:

- start = 0
- count = 100

Then, for the next iteration, we will start from the first result not retrieved in the first iteration:

- start = 100
- count = 100

The `total` return field can be used to create page-navigation for the search result, by dividing the `total` hits by the page-size (`count`) to get the needed number of pages.

Query

The query-part of a search is where the constraints are defined. All nodes in the repository will match when then query parameter is empty. The query is defined in the *Query Language* section.

The results matching the query constraint will be assigned a score. This is imperative for fulltext-type queries. The score of a matching document depends on how the constraint is defined, e.g which fulltext-like function is used. See the *Query Functions* section for details.

Filter & query-filter

A filter also applies constraints. The difference between a filter-constraint and a query-constraint, is that the hits matching the filter are not scored. Scoring hits is a costly operation, and makes no sense for typical filter constraints like "price > 10", so it's a good way of optimizing searches by appending non-fulltext operations to the filter-constraint instead of the query-constraint.

There are also two different kinds of filters. A *query-filter* is a part of the query-constraint, meaning that aggregations results are also affected by these constraints. A *filter* on the other hand, is not considered in the aggregations calculations, meaning that applying a filter will not impact the aggregation result.

Aggregations

An aggregation is a function, or something that is executed, on a collection of search results. The search-results are defined by the query and query-filter of the search request. See the *Aggregations* section for details.

Node Indexing

When nodes are persisted, they will automatically be indexed. If no special indexing instructions are passed with the node, the node will be indexed based on the property types.

Indexing instructions

By default, nodes will be indexed using the "type" instruction - using the propertyType to determine how it should be indexed. Special indexing options may be passed along with the node - forcing special handling of one or more properties. These options are described below:

type (default) Indexing is done based on type; e.g numeric values are stored as both string and numeric.

string Force indexing as a string-value only, no matter what type of data.

datetime Indexed as a datetime-value and string, no matter what type of data. If not able to parse value as date-time, no value will be indexed.

numeric Indexed as a numeric (double) and string, no matter what type of data. If not able to parse value as number, no value will be indexed.

minimal Indexed as a string-value only, no matter what type of data.

none Value not indexed.

ngram nGram-indexed fields are available for search by using the nGram-function. An nGram-analyzed field will index all substring values from 2 to 15 characters.

Consider this value of a property of type `text-line`:

```
"article"
```

This is split into the following tokens when analyzed:

```
'ar', 'art', 'arti', 'artic', 'articl', 'article'
```

For more information about how the nGram-function works, check out the nGram-function.

fulltext Fulltext-indexed fields are available for search by using the fulltext-function. A fulltext-analyzed field will be split into tokens.

Consider this value of a property of type `text-line`:

```
"This article contains information test-driven development"
```

This is split into the following tokens when analyzed:

```
'this', 'article', 'contains', 'information', 'about', 'test', 'driven', 'development'
```

For more information about how the fulltext-function works, check out the fulltext-function.

Indexed content properties

All user defined properties are indexed and available for queries.

In addition there are a number of standard content-properties available for search:

_alltext A collection of all fulltext-analyzed fields (textLine, textArea, htmlArea) in a content in one property

_id Holds the id of the content, typically generated automatically in the form of a UUID.

_manualordervalue The order value used when child-content is ordered manually

_name Holds the name of the content

_parentPath Reference to parent content path.

_path The content path

_permissions_read The principals that have read access.

_permissions_create The principals that have create access.

_permissions_delete The principals that have delete access.

_permissions_modify The principals that have modify access.

_permissions_publish The principals that have publish access.

_permissions_readpermissions The principals that have access to read the content permissions.

_permissions_writepermissions The principals that have access to change the content permissions.

_references Outgoing references to other content.

_score Calculated relevance for a hit

_state Used for keeping state of a content in a branch.

_timestamp The last change to the content version.

_versionKey The id of the node version.

attachment.size If any attachments, contains an array of attachment sizes

attachment.label If any attachments, contains an array of attachment labels

attachment.mimetype If any attachments, contains an array of attachment mime-types

attachment.name If any attachments, contains an array of attachment name

attachment.binary If any attachments, contains an array of attachment file-name

attachment.text If any attachments, contains the extracted text of e.g pdf-files

creator The user principal that created the content.

createdTime The timestamp when the content was created.

data A property-set containing all user defined properties defined in the content-type.

displayName Name used for display purposes.

language The locale-property of the content.

modifiedTime Last time the content was modified.

owner The user principal that owns the content.

page The page property contains page-specific properties, like template and regions.

page.region.component.textcomponent.text This property contains all values in the text-components added to pages

publish.from The time when the content was first published. This timestamp will be the set both in draft and master branch.

type The content-type name

x A property-set containing properties from mixins, also known as extra data. Indexed values depend on the specified mixins.

Query Functions

Here's a description of all functions that can be used in a query.

fulltext

The fulltext function is searching for words in a field, and calculates relevance scores for matches based on a set of rules (e.g number of occurences, field-length).

Tip: Only fields analyzed as text are considered when applying the fulltext-function. This includes, as default, all text-based fields in the content-domain.

Syntax

```
fulltext(<fields>, <search-string>, <operator>)
```

Fields

Fields is a string containing a comma-separated list of fields to include in the search. Wildcards are supported in field-names.

Some valid string values for "fields" are:

```
    'displayName' // Search in single field
    'displayName,data.description,data.title' // Search in multiple fields
    'data.*' // Wildcard usage
```

Note that "data." domain is used to access custom fields from custom content types. Default fields, like displayName, are directly available at the top level (without the "data." prepended).

You can boost - thus increasing or decreasing hit-score pr field basis - if providing more than one field to the query by appending a weight-factor: `^N`:

```
fulltext('displayName^5,data.description', 'my search string', 'AND')
```

Operator

The allowed operators are:

- `OR` Matches if any of the words in the search-string matches.
- `AND` Matches only if all words in search-string matches.

Search-string syntax

The search-string supports a set of operator:

- `+` signifies AND operation.
- `|` signifies OR operation.
- `-` negates a single token.
- `*` at the end of a term signifies a prefix query.
- `(` and `)` signify precedence.
- `~N` after a word signifies edit distance (fuzziness) with a number representing Levenshtein distance.
- `~N` after a phrase signifies slop amount.

Examples

Match if "myField" contains any of the given words.

```
fulltext("myField", "cheese fish cake onion", "OR")
```

Match if any field with path starting with "myData.myProperties" contains any of the given words.

```
fulltext("myData.myProperties.*", "cheese fish cake onion", "OR")
```

Match if "myField" contains any of the given words and "myCategory" = "soup".

```
myCategory = "'soup" AND fulltext("myField", "cheese fish cake onion", "OR")
```

Match if "myField" contains all the given words.

```
fulltext("myField", "cheese fish cake onion", "AND")
```

Match if "myField" contains "Levenshtein" with a fuzziness distance of 2.

```
fulltext("myField", "Levenshtein~2", "AND")
```

Match if "myField" contains "fish" and not "boat".

```
fulltext("myField", "fish -boat", "AND")
```

Match if any field under data-set data contains "fish" and not "boat".

```
fulltext("data.*", "fish -boat", "AND")
```

nGram

An n-gram is a sequence of n letters from a string. The nGram-function is used to search for words or phrases beginning with a given search string. Typically, find-as-you-type searches will use this function.

Tip: Only fields analyzed as text are considered when applying the ngram-function. This includes, as default, all text-based fields in the content-domain.

Syntax

```
ngram(<field>, <search-string>, <operator>)
```

Operator

The allowed operators are:

- `OR` Matches if any of the words in the search-string matches.
- `AND` Matches only if all words in search-string matches.

Examples

Matches if “myField” contains any word beginning with “lev”, e.g “Levenshteins Algorithm”.

```
    ngram("myField", "lev", "AND")
```

Matches if “myField” contains words beginning with “lev” and “alg”, e.g “Levenshteins Algorithm”.

```
    ngram("myField", "lev alg", "AND")
```

Matches if “myField” contains words beginning with “fish” or “boat”, e.g “fishpond” or “boatman”.

```
    ngram("myField", "fish boat", "OR")
```

range

The range functions test each value in the given field for a given range.

Syntax

```
range(<field>, <from>, <to>, [<includeFrom>], [<includeTo>])
```

The `from` and `to` values must be of the same type.

`includeFrom` and `includeTo` are optional with default value ‘false’, meaning that the actual values for the `from` and `to` are not included as matches.

Unbounded ranges can be queried by providing an empty string as argument.

Examples

Matches all that have a version-string in the range, including ‘6.3.0’

```
    range('version', '6.3.0', '6.4.0', 'true', 'false')
```

```
    range('publishFrom', instant('2015-08-01T09:00:00Z'), instant('2015-08-01T11:00:00Z') )
```

Matches all that have values between 2.0 and 3.0, including 2.0

```
    range('myValue', 2.0, 3.0, 'true', 'false' )
```

Matches all that have publishFrom-date newer that the given date.

```
    range('publishFrom', instant('2015-08-01T09:00:00Z'), '')
```

Matches all that have publishTo-date older that the given date.

```
    range('publishTo', '', instant('2015-08-01T09:00:00Z'))
```

pathMatch

The path-match matches a path in a same branch, scoring the paths closest to the given query path first. Also, a number of minimum matching elements that must match could be set.

Syntax

```
pathMatch(<field>, <path>, [<minimum_elements_must_match>])
```

If not given, the default mimum-must-match value will be 1.

Examples

Given these contents:

```
/content/mySite
/content/mySite/fish
/content/mySite/fish/onion
/content/mySite/cheese
/content/mySite/cheese/jam
/content/myOtherSite
```

```
pathMatch('_path', '/content/mySite/fish/onion/mayonnaise', 2)
```

This will return (orded by *_score*):

1. */content/mySite/fish/onion*
2. */content/mySite/fish*
3. */content/mySite/cheese/jam*
4. */content/mySite/cheese*
5. */content/mySite*

Order Functions

Here’s a description of all functions that can be used in order-by clause.

geoDistance

The geoDistance-function enables you to order the results according to distance to a given geo-point.

Tip: Documents with no geo-point property with the given path will be ordered last if matching the query.

Syntax

```
geoDistance(<field>, <location>)
```

Field Field-argument accepts a path to a property containing geoPoint data.

Location The location is a geoPoint from which the distance factor should be calculated, formatted as "latitude,longitude".

Examples

Order by distance from "shopLocation" to the fixed location.

```
    ORDER BY geoDistance("shopLocation", "59.9127300,10.7460900")
```

Aggregations

An aggregation is a function that is executed on a collection of search results. The search-results are defined by the query and query-filter of the search request.

For instance, consider a query returning all nodes that have a property "price" less than, say, $100. Now, we want to divide the result nodes into ranges, say 0-$25, $25-$50 and so on. We also would like to know the average price for each category. This could be done by doing multiple separate queries and calculating the average manually, but this would be very inefficient and cumbersome. Luckily, aggregations solve these types of problems easily.

In some API functions it is possible to send in an aggregations expression object. This object is either in Java or a JSON like the following:

```
"aggregations" : {
  "[name]" : {
    "[type]" : {
      ... body ...
    },
    "aggregations": {
      ... sub-aggregations ...
    }
  }
}
```

There are two different types of aggregations:

- Bucket aggregations: A bucket aggregation places documents matching the query in a collection - a bucket. Each bucket has a key.
- Metrics aggeregations: A metric aggeregation computes metrics over a set of documents.

Typically, you will divide data into buckets and then use metric aggregations to calculate e.g average values, sum, etc for each bucket, if necessary.

terms

The 'terms' aggergation places documents into bucket based on property values. Each unique value of a property will get its own bucket. Here's a list of properties:

field (string) The property path.

size (int) The number of bucket to return, ordered by the given orderType and orderDirection. Default to `10`.

order (string) How to order the results, type and direction. Default to `_term ASC`.

Types:

- `_term`: Alphabetic ordering of bucket keys.
- `_count`: Numeric ordering of number of document in buckets.

Here's an example of the terms aggregation:

```
"aggregations": {
  "categories": {
    "terms": {
      "field": "myCategory",
      "order": "_count desc",
      "size": 10
    }
  }
}
```

The above example gives a result with this structure:

```
"aggregations": {
  "categories": {
    "buckets": [{
      "docCount": 132,
      "key": "articles"
    },
    {
      "docCount": 101,
      "key": "documents"
    },
    {
      "docCount": 43,
      "key": "case-studies"
    }]
  }
}
```

range

The range aggregation query defines a set of ranges that represents a bucket. Here's a list of properties:

field (string) The property path.

ranges (range[]) The range-buckets to create.

range (from: number, to: number) Defines a range to create a bucket for. From-value is included in bucket, to is excluded.

Here's an example of the range aggregation:

```
"price_ranges": {
  "range": {
    "field": "price",
    "ranges": [
      { "to": 50 },
      { "from": 50, "to": 100 },
      { "from": 100 }
    ]
  }
}
```

The above example gives a result with this structure:

```
"price_ranges": {
  "buckets": [{
    "docCount": 2,
    "key": "a",
    "to": 50
  },
  {
    "docCount": 4,
    "from": 50,
    "key": "b",
    "to": 100
  },
  {
    "docCount": 4,
    "from": 100,
    "key": "c"
  }]
}
```

dateRange

The dateRange aggregation query defines a set of date-ranges that represents a bucket. Only documents with properties of type 'DateTime' will considered in the `dateRange` aggregation buckets. Here's a list of properties:

field (string) The property path.

format (string) The date-format of which the buckets will be formatted to on return. Default to `YYYY-MM-DDThh:mm:ssTZD`.

ranges (range[]) The range-buckets to create.

range (from: number, to: number) Defines a range to create a bucket for. From-value is included in bucket, to is excluded. The from and to follows a special date-math explained below.

Here's an example of the dateRange aggregation:

```
"my_date_range": {
  "dateRange": {
    "field": "date",
    "format": "MM-yyy",
    "ranges": [{
      "to": "now-10M"
    },
    {
      "from": "now-10M"
    }]
  }
}
```

The above example gives a result with this structure:

```
"price_ranges": {
  "buckets": [{
    "docCount": 2,
    "key": "a",
    "to": 50
  },
  {
    "docCount": 4,
    "from": 50,
    "key": "b",
    "to": 100
  },
  {
    "docCount": 4,
    "from": 100,
    "key": "c"
  }]
}
```

Date-math expression

The range fields accepts a date-math expression to calculate the time-spans.

Now minus a day:

```
now-1d
```

The given date minus 3 days plus one minute:

```
2014-12-10T10:00:00Z||-3h+1m
```

Range describing now plus one day and thirty minutes, rounded to minutes:

```
now+1d+30m/m
```

dateHistogram

The date-histogram aggregation query defines a set of bucket based on a given time-unit. For instance, if querying a set of log-events, a `dateHistorgram` aggregations query with interval `h` (hour) will divide each log event into a bucket for each hour in the time-span of the matching events. Here's a list of properties:

field (string) The property path.

interval (string) The time-unit interval for creating bucket. Supported time-unit notations:

- `y` = Year
- `M` = Month
- `w` = Week
- `d` = Day
- `h` = Hour
- `m` = Minute
- `s` = Second

format (string) Output format of date string.

minDocCount (int) Only include bucket in result if number of hits <= `minDocCount`.

Here's an example of the date_histogram aggregation:

```
"by_month": {
  "dateHistogram": {
    "field": "init_date",
    "interval": "1M",
    "minDocCount": 0,
    "format": "MM-yyy"
  }
}
```

The above example gives a result with this structure:

```
"by_month" : {
  "buckets" : [{
    "docCount" : 8,
    "key" : "2014-01"
  }, {
    "docCount" : 10,
    "key" : "2014-02"
  }, {
    "docCount" : 12,
    "key" : "2014-03"
  }]
}
```

stats

The stats-aggregations calculates the following statistics for the parent-aggregation buckets:

- avg
- min
- max
- count
- sum

Here's a list of properties:

field (string) The property path.

Here's an example of the stats aggregation:

```
{
  "start": 0,
  "count": 0,
  "aggregations": {
    "products": {
      "terms": {
        "field": "data.product.category",
        "order": "_count desc",
        "size": 10
      },
      "aggregations": {
        "priceStats": {
          "stats": {
            "field": "data.product.price"
          }
        }
      }
    }
  }
}
```

The above example gives a result with this structure:

```
"products": {
  "buckets": [{
    "key": "tv",
    "docCount": 123,
    "priceStats": {
      "count": 123,
      "min": 2599,
      "max": 87944,
      "avg": 7400,
      "sum": 578100
    }
  },
  {
    "key": "blu-ray player",
    "docCount": 42,
    "priceStats": {
      "count": 42,
      "min": 699,
      "max": 5999,
      "avg": 1548,
      "sum": 65016
    }
  },
  {
    "key": "reciever",
    "docCount": 12,
    "priceStats": {
      "count": 12,
      "min": 2999,
      "max": 26950,
      "avg": 5548,
      "sum": 66756
    }
  }]
}
```

geo-distance

The geo_distance aggregation needs a defined range to split the documents into buckets. Only documents with properties of type 'GeoPoint' will be considered in the `geo_distance` aggregation buckets.

Here's a list of properties:

field (string) The property path.

ranges (range[]) The range-buckets to create.

range (from: number, to: number) Defines a range to create a bucket for. From-value is included in bucket, to is excluded.

unit (string) The meassurement unit to use for the ranges. Legal values are either the full name or the abbreviation of the following: km (kilometers), m (meters), cm (centimeters), mm (millimeters), mi (miles), yd (yards), ft (feet) or nmi (nauticalmiles).

origin (lat: number, lon: number) The GeoPoint from which the distance is measured.

Here's an example of the range aggregation:

```
"aggregations": {
    "distance": {
        "geo_distance": {
            'field': "data.cityLocation",
            'unit': "km",
            'origin': {
                'lat': "90.0",
                'lon': "0.0"
            },
            'ranges': [ { 'from': 0, 'to': 1200 }, { 'from': 1200, 'to': 4000 }, { 'from': 4000,
        }
    }
}
```

The above example gives a result with this structure:

```
"aggregations":
  {"distance":
    {"buckets": [{
      "key": "*-1200.0",
      "doc_count": 3,
    },
    {
      "key": "1200.0-4000.0",
      "doc_count": 4,
    },
    {
      "key": "4000.0-12000.0",
      "doc_count": 5,
    },
    {
      "key": "12000.0-*",
      "doc_count": 1,
    },],},
  }
```

NOTE: At the time of writing, there is only one way of find out which result belongs to which bucket: By also sorting the result on geo_distance, and matching the order to the number of each bucket. In a future version, there will easier ways of doing this.

Ordering results

Any field can be used to order the result, either (default) DESC (descending) or ASC (ascending). The default field for ordering is *_score*

_score

The score-value is calculated based on a number of factors, e.g number of matching clauses in boolean expressions, how often the term appears in the documents when searching for text etc. See elasticsearch-documentation for more insight.

Querying date and time

Querying against date and time-fields may require some knowledge on how data is stored and indexed.

LocalDate

LocalDate represents a date without time-zone in the ISO-8601 calendar, e.g `2015-03-19`. LocalDate-properties are stored as a ISO LocalDate-formatted string in the index, thus all searches are done against string-values.

LocalDate string-format:

```
yyyy-MM-dd
```

Given a node with a property named 'myLocalDate' of type `localDate` and value `2015-03-19`, all of the following queries will match:

```
myLocalDate = '2015-03-19'
myLocalDate > '2015-03-18'
myLocalDate <= '2015-03-19'
```

LocalTime

LocalTime represents a time without time-zone in the ISO-8601 calendar, e.g `11:39:49`. LocalTime-properties are stored as a ISO LocalTime-formatted string in the index, thus all searches are done against string-values.

LocalTime string-format:

```
HH:mm[:ss[.SSS]]
```

LocalTime string value examples:

```
09:30
10:00
10:00:30
10:00:30.142
```

Since the queries are matching string-values, the input time in query must either adhere the same string-format restrictions, or be wrapped in a function `time` which accepts a time-formatted string as input.

Given a node with a property named 'myLocalTime' of type `localTime` and value = `09:36:00`, all the following queries will match:

```
myLocalTime > '09:00'
myLocalTime = '09:36'
myLocalTime = '09:36:00'
myLocalTime LIKE '09:*'
myLocalTime < '09:36:01'
myLocalTime < '09:36:00.1'
```

This must be wrapped in time-function since its not padded with a leading 0:

```
myLocalTime > time('9:00')
```

If optional fractions of seconds are given, the string format will also contain this even if 0, and expression will not match unless wrapped in time-function:

```
myLocalTime = time('09:36:00.0')
```

Even if the string-matching will do the job 99% of the time, the safest bet is to always go with the time-function when applicable.

LocalDateTime

LocalDateTime represents a date-time without time-zone in the ISO-8601 calendar, e.g `2015-03-19T11:39:49`. LocalDateTime-properties are stored as a ISO LocalDateTime-formatted string in the index, thus all searches are done against string-values.

LocalDateTime string-format:

```
yyyy-MM-ddTHH:mm[:ss[.SSS]]
```

Since the queries are matching string-values, the input dateTime in query must either adhere the same string-format restrictions, or be wrapped in a function `dateTime` which accepts a dateTime-formatted string as input.

Given a node with a property named 'myLocalDateTime' of type `localDateTime` and value `2015-03-19T10:30:00`, all of the following queries will match:

```
myLocalDateTime = '2015-03-19T10:30:00'
myLocalDateTime = dateTime('2015-03-19T10:30')
myLocalDateTime < dateTime('2015-03-19T10:30:00.001')
```

DateTime / Instant

DateTime represents a date-time with time-zone in the ISO-8601 calendar, e.g `2015-03-19T11:39:49+02:00`. Its possible to query properties of with value-type *DateTime* both as an ISO instant and as ISO dateTime, using the provided built-in functions `instant` and `dateTime`.

Instant string-format (instant always given in UTC-time):

```
yyyy-MM-ddTHH:mm[:ss[.SSS]Z
```

Instant string value examples:

```
2015-03-19T16:30:20Z
2015-03-19T16:30:20.123Z
```

DateTime string-format (*Z* for UTC, else offset in hours and minutes):

```
yyyy-MM-ddTHH:mm[:ss[.SSS](Z|+hh:mm|-hh:mm)
```

DateTime string value examples:

```
2015-03-19T16:30:20Z
2015-03-19T16:30:20+01:00
2015-03-19T16:30:20-01:30
2015-03-19T16:30:20.123-01:30
```

Given a node with a property named 'myDateTime' of type `dateTime` and value `2015-03-19T10:25:00+02:00`, all of the following queries will match:

```
myDateTime = instant('2015-03-19T08:25:00Z')
myDateTime = dateTime('2015-03-19T08:25:00Z')
myDateTime = dateTime('2015-03-19T10:25:00+02:00')
myDateTime = dateTime('2015-03-19T11:25:00+03:00')
```

Querying paths

All nodes have three system-properties concerning the node placement in a branch, all of type `String`:

- `_name`: The node name without path.
- `_parentPath`: The parent node path.
- `_path`: The full path of the node.

See the query-function *pathMatch* for advanced path-matching

Note: When working with content, all paths are under a special root containing content; */content*.

While this mostly is explicit when working in the content-domain, this has to be dealt with when using paths in query-expressions and functions since you are actually querying nodes.

Examples

Finds node with path `/content/mySite/myCategory/myContent`.

```
_path = '/content/mySite/myCategory/myContent'
```

Finds all nodes with name `myContent` in a folder named `myCategory` e.g `/content/test/thisIsMyCategory/myContent` and `/content/myCategory/myContent`.

```
_name = 'myContent' AND _parentPath LIKE '*myCategory'
```

Finds all nodes under the path `/content/mySite/myCategory` including children of children.

```
_path LIKE '/content/mySite/myCategory/*'
```

Finds only first level children under the path `/content/mySite/myCategory`.

```
_parentPath = '/content/mySite/myCategory'
```

Querying references

References to other nodes are stored in a property named `_references`. This could be used to find incoming references to a node.

Find all nodes referring to the node with id = 'abc':

```
_references = 'abc'
```

Admin

Go go backoffice!

Complex applications often require some kind of "back office" tools for management. Enonic XP provides a standardized approach to deliver and extend Admin tools.

This document describes how you can build your own *Admin Tools* and extend some of the default tools shipped with XP through *Widgets*.

Admin Tools

> **Warning:** Admin Tool support is experimental.

Admin Tools are independent “back office” user interfaces designed to manage Enonic XP or installed applications. Each tool will run in it’s own browser tab - here are some of the reasons for this:

- Faster user interfaces and better deep-linking support
- Developers can use their favorite front-end frameworks
- Simplified debugging

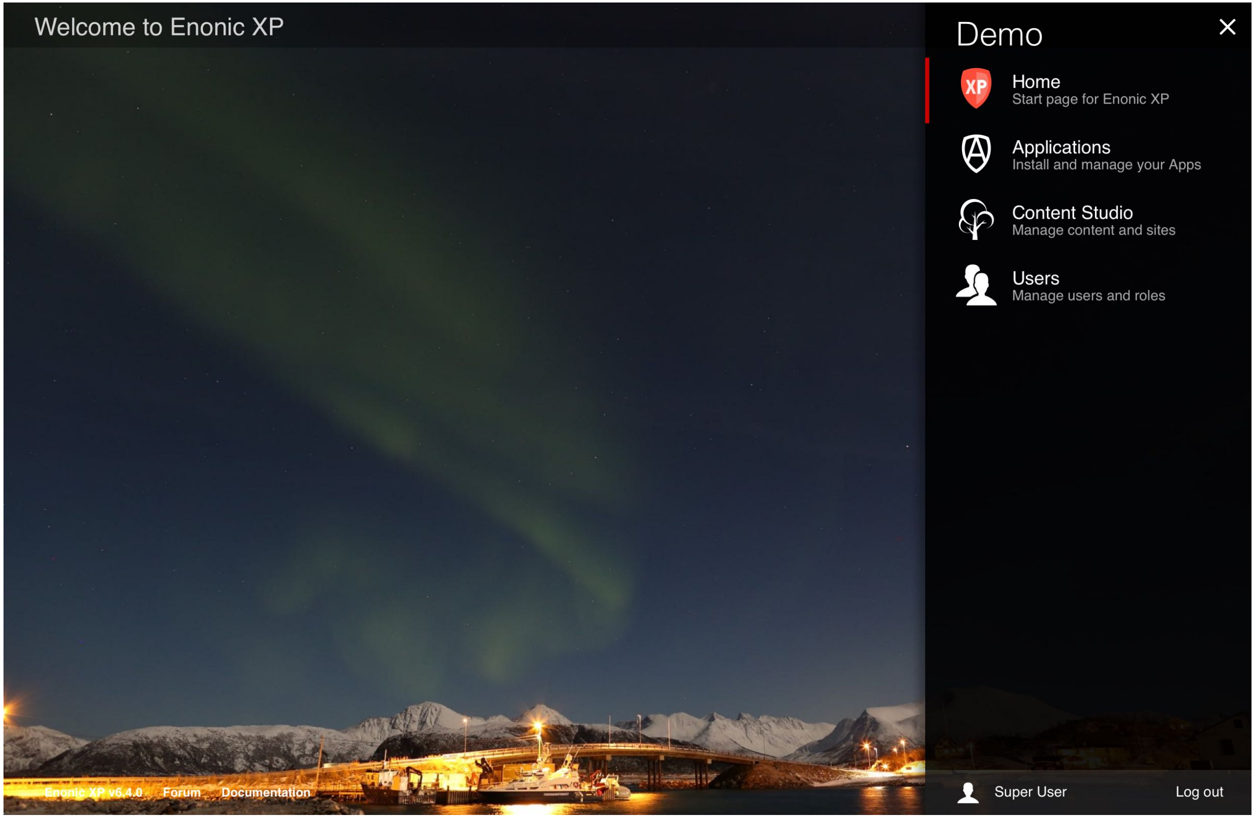
Standard Tools

Enonic XP ships with the following tools by default:

- Home (The default tool)
- Applications (Install, stop, start and uninstall applications)
- Content Studio (Create and manage content and sites)
- Users (Create, setup and manage users, groups and roles)

Launcher

Navigation between the various Admin Tools is done via the “Launcher Panel”, Accessible from the top right corner. This icon and the Launcher panel should be available across all Admin Tools.



To create a new Admin Tool, you must create a new folder in your project structure, i.e. `admin/tools/[tool-name]`. Then you must place a descriptor, an icon and a controller there.

Descriptor

The tool `descriptor` defines the basic info to be displayed in the launcher and which roles are required to access the tool.

The descriptor file must have the same name as the tool, i.e. `admin/tools/[tool-name]/[tool-name].xml`:

```xml
<tool>
  <display-name>My Tool</display-name>
  <description>Control my stuff</description>
  <allow>
    <principal>role:system.admin</principal>
    <principal>role:myapp.myrole</principal>
  </allow>
</tool>
```

Icon

You should add an SVG icon to the tool. This will be displayed in the launcher panel together with the info from the descriptor. The icon file must have the same name as the tool, i.e. `admin/tools/[tool-name]/[tool-name].svg`:

Controller

To drive the tool, we will need a **controller** (See *HTTP Controllers*). The controller typically produces the initial tool html. Depending on the tool implementation it may also handle sub-requests from the tool.

The controller must have the same name as the tool, i.e. `admin/tools/[tool-name]/[tool-name].js`:

```javascript
var mustache = require('/lib/xp/mustache');
var portalLib = require('/lib/xp/portal');
var timestamp = Date.now();

function handleGet() {
    var view = resolve('./my-view.html');

    var params = {
        adminUrl: portalLib.url({path: "/admin"}),
        assetsUri: portalLib.url({path: "/admin/assets/" + timestamp}),
        appId: 'my-custom-tool',
        appName: 'My custom tool'
    };
    return {
        contentType: 'text/html',
        body: mustache.render(view, params)
    };
}

exports.get = handleGet;
```

Adding the Launcher Panel menu

Adding the Launcher Panel menu to a custom admin tool requires two steps.

1. In the <body> section of the view add a Javascript snippet where you define an object variable called “CONFIG” with properties adminUrl, assetsUri and appId which will get their values from the controller:

```
<body>
    <!-- Configuration -->
    <script type="text/javascript">
    var CONFIG = {
        adminUrl: '{{adminUrl}}',
        assetsUri: '{{assetsUri}}',
        appId: '{{appId}}'
    };
    </script>
</body>
```

2. Under the Javascript snippet add a reference to the Launcher's Javascript file as shown below:

```
<body>
    <!-- Append the launcher -->
    <script type="text/javascript" src="{{assetsUri}}/apps/launcher/js/_all.js" async></script>
</body>
```

The entire view:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=Edge"/>
    <meta name="viewport" content="width=device-width, user-scalable=no">
    <meta name="theme-color" content="#ffffff">

    <title>{{appName}}</title>

    <!-- Styles -->
    <link rel="stylesheet" type="text/css" href="{{assetsUri}}/common/styles/_all.css">

    <!-- Common lib -->
    <script type="text/javascript" src="{{assetsUri}}/common/lib/_all.js"></script>

</head>
<body>

<!-- Configuration -->
<script type="text/javascript">
    var CONFIG = {
        adminUrl: '{{adminUrl}}',
        assetsUri: '{{assetsUri}}',
        appId: '{{appId}}'
    };
</script>

<!-- Append the Launcher Panel -->
<script type="text/javascript" src="{{assetsUri}}/apps/launcher/js/_all.js" async></script>

</body>
</html>
```

Widgets

Warning: Widget support is experimental.

Widgets are user interface components that can be used to extend various Admin Tools. Currently, the only available extension point for widgets are detail panels in the Content Studio tool.

To create a widget, you must create a new folder in your project structure, i.e. `admin/widgets/[widget-name]`. Then you must place a descriptor and a controller there.

Descriptor

The widget `descriptor` defines the display name and the interfaces it matches.

An interface is simply a unique identifier that is used to create a link between a tool and the widget. For example, for your widget to be displayed in the "Content Studio" detail panel, add the interface "contentstudio.detailpanel"

The descriptor file must match the widget name, i.e. `admin/widgets/[widget-name]/[widget-name].xml`:

```
<widget>
  <display-name>My first widget</display-name>
  <interfaces>
    <interface>contentstudio.detailpanel</interface>
  </interfaces>
</widget>
```

Controller

To drive the widget, we will need a `controller` (See *HTTP Controllers*). The controller typically produces the initial widget html. Depending on the widget implementation it may also handle sub-requests from the widget.

The controller file must match the widget name, i.e. `admin/widgets/[widget-name]/[widget-name].js`:

```
exports.get = function (req) {
    return {
        body: '<html><head></head><body><h1>My first widget</h1></body></html>',
        contentType: 'text/html'
    };
};
```

ID Providers

Warning: ID Providers support is experimental.

An ID Provider is a pluggable authentication module, contained inside an application. Install an existing ID Provider or write your own, configure it and associate it to a user store.

To create an ID provider, you must create a new folder in your project structure, i.e. `idprovider`. Then you must place a descriptor and a controller there.

Descriptor

The ID Provider descriptor is a required XML file used to define the mode and the configuration required by the provider. The descriptor file must have the following path: `idprovider/idprovider.xml`:

```
<id-provider>
  <mode>LOCAL</mode>
  <config>
    <input name="title" type="TextLine">
      <label>Title</label>
      <occurrences minimum="0" maximum="1"/>
      <default>User Login</default>
    </input>
  </config>
</id-provider>
```

mode Specifies how the provider uses the user store.

LOCAL: Both the users and groups are stored locally in the user store.

EXTERNAL: Both the users and groups are stored in a remote system. The user store is only a snapshot view of this remote system and therefore the users and groups are not editable in the admin tool *Users*.

MIXED: The users are stored in a remote system and the groups in the user store. The users in the user store are only a snapshot view of this remote system and therefore the users are not editable in the admin tool *Users*.

config Specifies the input fields of the configuration required by the ID provider.

Controller

The controller is a required file written in JavaScript and must have the following path: `idprovider/idprovider.js`:

```
var authLib = require('/lib/xp/auth');
var portalLib = require('/lib/xp/portal');

exports.handle401 = function (req) {
    var body = generateLoginPage();
    return {
        status: 401,
        contentType: 'text/html',
        body: body
    };
};

exports.login = function (req) {

    // If this function was called with a parameter "redirect", a validation of the origin is perfor
    // The result of the validation is passed to the ID Provider as a request property "validTicket"
    var redirectUrl = req.validTicket ? req.params.redirect : undefined;

    var body = generateLoginPage(redirectUrl);
    return {
        contentType: 'text/html',
        body: body
    };
};

exports.logout = function (req) {

    // Calling "authLib.logout()" will log out the current user from Enonic XP.
    authLib.logout();
    // If this function was called with a parameter "redirect", a validation of the origin is perforn
    // The result of the validation is passed to the ID Provider as a request property "validTicket"
    var redirectUrl = req.validTicket ? req.params.redirect : undefined;

    if (redirectUrl) {
        return {
            redirect: redirectUrl
        };
    } else {
        var body = generateLoginPage();
        return {
            contentType: 'text/html',
            body: body
        };
    }
};

exports.autoLogin = function (req) {
    log.info('Auto login. Invoked only when user is not authenticated');
};

function generateLoginPage(redirectUrl) {
    var authConfig = authLib.getIdProviderConfig();
    var title = authConfig.title || "User Login";
    var redirectionLink = redirectUrl ? '<a href="' + redirectUrl + '">Return</a>' : '';
    return '<html><head></head><body><h1>' + title + '</h1>' + redirectionLink + '</body></html>';
};
```

handle401 Optional function rendered in the case of a 401 error. This function typically produces a login or error page.

get/post/... Functions rendered. An ID provider controller exports a method for each type of HTTP request that should be handled. The portal function `idProviderUrl()` will create a dynamic URL to this function.

login Function rendered. The portal function `loginUrl()` will create a dynamic URL to this function.

logout Function rendered. The portal function `logoutUrl()` will create a dynamic URL to this function.

autoLogin Optional function executed if the current user is unauthenticated. This functions allows you to, for example, handle web tokens or other request headers.

Set up an ID Provider

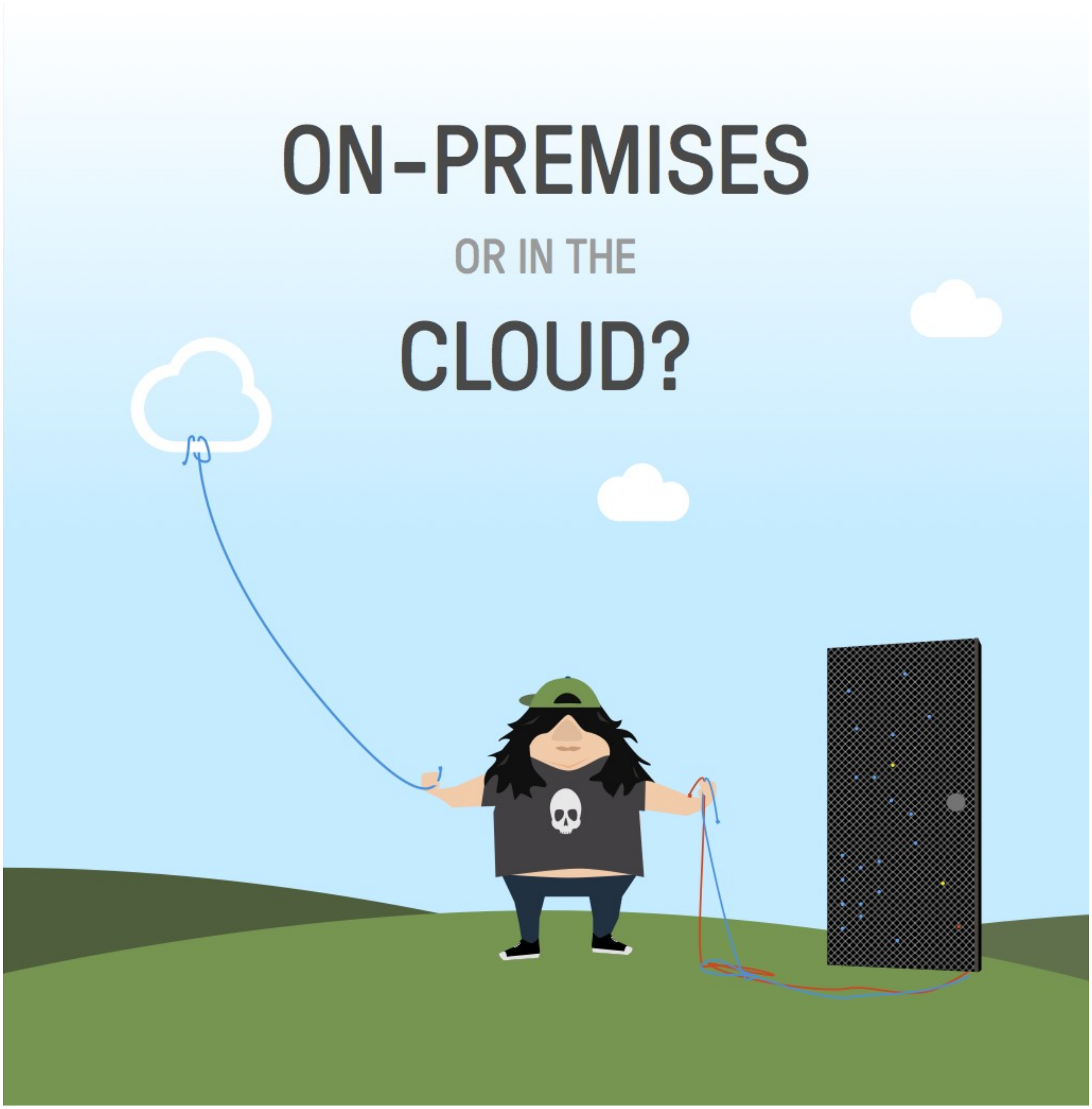
To set up an ID Provider, you must follow the 3 steps below:

1. An application containing an ID Provider must be installed and started (Use the admin tool *Applications*).
2. The application must be associated to a user store and configured (Use the admin tool *Users* and edit the user store to guard).
3. The user store must be associated to a virtual host in the virtual host configuration file (see *Virtual Host Configuration*).

CHAPTER 4

Operations Guide

This guide gives you all the gory details on how to tune Enonic XP - if you're looking for installation guides try *Getting Started*



Package Structure

The unzipped Enonic XP distribution will have the following structure (folders with a * will not appear until the installation is started the first time)

```
enonic-xp-[version]
 |- bin/
 |- home/
   |- config/
   |- data/ *
   |- deploy/
   |- logs/
```

```
  |- repo/ *
  |- snapshots/
  |- work/ *
|- lib/
|- system/
|- toolbox/
|- work/ *
```

The root installation folder is referred to as XP_INSTALL. Here's an explanation of all the other folders:

bin/ Contains the scripts for starting and stopping Enonic XP and setting environment variables.

home/ Home directory, also called XP_HOME. All files for a specific instance of XP reside here. This folder can be copied to other locations for working with multiple projects.

> **config/** Configuration files are placed here, including Virtual Host and system.properties.
>
> **data/** Additional data like exports and dumps. This folder will not appear until certain operations are run.
>
> **deploy/** Hot deploy directory. Applications are automatically installed upon placing their JAR files in this directory.
>
> **logs/** Default location for logs.
>
> **repo/** Repository data (blobs and indexes). This folder will not appear until the installation is started for the first time.
>
> **snapshots/** This is where snapshots are stored when using the *snapshot*-operation. This folder will not appear until a snapshot is done.
>
> **work/** Cache and generated bundles are stored here. This folder will not appear until the installation is started the first time.

lib/ Contains the bootstrap code used to launch Enonic XP.

system/ System OSGi bundles are placed here.

toolbox/ Command-line interface tool to manage the server. See *Toolbox CLI*.

work/ OSGI cache is stored here. This folder will not appear until the installation is started for the first time.

Configuration

Enonic XP, system modules, and 3rd party modules can easily be configured by editing the files in the $XP_HOME/config/ directory.

When changing files ending with .cfg, their respective modules will automatically restart with the new configuration. Files ending with .properties require a full restart of Enonic XP to be applied. In a clustered environment each node must be restarted.

System Configuration

The default system.properties are listed below.

Listing 4.1: `$XP_HOME/config/system.properties`

```
#
# Installation settings
#
xp.name = demo

#
# Configuration FileMonitor properties
#
felix.fileinstall.poll = 1000
felix.fileinstall.noInitialDelay = true
```

Virtual Host Configuration

Virtual hosts have their own configuration file and settings are automatically updated upon changes. A sample virtual host configuration is listed below.

Listing 4.2: `$XP_HOME/config/com.enonic.xp.web.vhost.cfg`

```
enabled = true

mapping.test.host = localhost
mapping.test.source = /
mapping.test.target = /
mapping.test.userStore = system

mapping.intranet.host = enonic.com
mapping.intranet.source = /
mapping.intranet.target = /portal/master/enonic.com
mapping.intranet.userStore = enonic

mapping.admin.host = enonic.com
mapping.admin.source = /admin
mapping.admin.target = /admin
mapping.admin.userStore = system
```

In this example file, three mappings are configured.

host Host-name to match.

source Requested path to match.

target Path to which the request is sent.

userStore Key of the user store associated to this virtual host (see *ID Providers*).

In the second example, mapping “intranet”, a site is mapped to the root of the URL, which would be normal in production environments.

In the third example, the admin site is mapped to `enonic.com/admin`.

Mail Configuration

The mail server used for sending email messages can be configured. A sample mail configuration is listed below.

Listing 4.3: `$XP_HOME/config/com.enonic.xp.mail.cfg`

```
smtpHost=mail.server.com
smtpPort=25
smtpAuth=true
smtpUser=user
smtpPassword=secret
smtpTLS=false
```

smtpHost Host name of the SMTP server. Default `localhost`.

smtpPort TCP port of the SMTP server. Default `25`.

smtpAuth Enable authentication with the SMTP server. Default `false`.

smtpUser User to be used during authentication with the SMTP server, if 'smtpAuth' is set to true.

smtpPassword Password to be used during authentication with the SMTP server, if 'smtpAuth' is set to true.

smtpTLS Turn on Transport Layer Security (TLS) security for SMTP servers that require it. Default `false`.

Storage Configuration

Blobstore Configuration

Main blobstore configuration is configured in `com.enonic.xp.blobstore.cfg`.

Listing 4.4: `$XP_HOME/config/com.enonic.xp.blobstore.cfg`

```
provider = file
cache = true
cache.sizeThreshold = 1mb
cache.memoryCapacity = 100mb
```

provider The blobstore provider to be used for storing. Default value is `file`. Other providers will be available in future versions / enterprise-edition. Each provider will have a separate configuration file named `com.enonic.xp.blobstore.<providername>.cfg`

cache Enable or disable memory caching of blobs fetched from the blobstore. Default true

cache.sizeThreshold The maximum size for objects to be cached, defaults to 1MB. You will usually avoid filling up the cache with large blobs, but rather cache smaller objects that are used often. The size notation accepts a number plus byte-size idenfier (`b`/`kb`/`mb`/`gb`/`tb`/`pb`)

cache.memoryCapacity The maximum memory footprint of the blob cache. Defaults to 100MB. The size notation accepts a number plus byte-size idenfier (`b`/`kb`/`mb`/`gb`/`tb`/`pb`)

File blobstore configuration

Listing 4.5: `$XP_HOME/config/com.enonic.xp.blobstore.file.cfg`

```
baseDir = ${xp.home}/repo/blob
readThrough.provider =
readThrough.enabled = false
readThrough.sizeThreshold = 100mb
```

baseDir Base-directory for storing blobs. Defaults to `${xp.home}/repo/blob`.

readThrough.provider = none Readthrough provider name, if enabled. A readthrough provider stores and fetches blobs through an intermediate blobstore. Typically used for providers using a remote blobstore where caching as local files will improve performance.

readThrough.enabled = false Enable or disable readthough provider usage. Default to false.

readThrough.sizeThreshold = 100mb The maximum size for objects to be stored in the readthrough provider, defaults to 100MB. The size notation accepts a number plus byte-size idenfier (`b`/`kb`/`mb`/`gb`/`tb`/`pb`)

Elasticsearch configuration

The cluster functionallity is facilitated by Elasticsearch, so all relevant Elasticsearch settings are available.

When changing `com.enonic.xp.elasticsearch.cfg`, the node will automatically restart with the new configuration.

Listing 4.6: `$XP_HOME/config/com.enonic.xp.elasticsearch.cfg`

```
node.name = local-node
node.master = true
node.local = true

path = ${xp.home}/repo/index
path.data = ${path}/data
path.work = ${path}/work
path.conf = ${path}/conf
path.logs = ${path}/logs
path.plugins = ${path}/plugins

cluster.name = mycluster
cluster.routing.allocation.disk.threshold_enabled = false

http.enabled = false
network.host = 127.0.0.1
transport.tcp.port = 9300-9400

gateway.expected_nodes = 1
gateway.recover_after_time = 5m
gateway.recover_after_nodes = 1
discovery.zen.minimum_master_nodes = 1
discovery.zen.ping.unicast.hosts = 127.0.0.1

index.recovery.initial_shards = quorum
```

node.name Node name. Default `local-node`.

node.master Allow this node to be eligible as a master node. Default `true`.

node.data Allow data to be distributed to this node. Default `true`.

path Path to directory where elasticsearch stores files. Default `${xp.home}/repo/index`. Should be on a local file-system, not sharded.

path.data Path to directory where to store index data allocated for this node. Default `$path/data`.

path.work Path to temporary files. Default `${xp.home}/repo/index/work`.

path.conf Path to directory containing configuration. Default `$path/conf`.

path.logs Path to log files. Default `${xp.home}/repo/index/logs`.

path.plugins Path to where plugins are installed. Default `$path/plugins`.

cluster.name Cluster name. Default `mycluster`.

cluster.routing.allocation.disk.threshold_enabled Prevent shard allocation on nodes depending on disk usage. Default `false`.

http.enabled Enable the HTTP module. Default `false`.

network.host Set the bind address and the address other nodes will use to communicate with this node. Default `127.0.0.1`. Can be an explicit *IP-address*, a *host-name* or an *alias*. See the section below for an overview of aliases.

transport.tcp.port Custom port for the node to node communication. Defaults to the range `9300-9400`.

gateway.expected_nodes Number of nodes expected to be in the cluster to start the recovery immediately. Default `1`.

gateway.recover_after_time Time to wait until recovery happens once the nodes are met. Default `5m`.

gateway.recover_after_nodes Number of nodes expected to be in the cluster to start the recovery after gateway.recover_after_time. Default `1`.

discovery.zen.minimum_master_nodes Ensure a node sees N other master eligible nodes to be considered operational within the cluster. Default `1`.

discovery.zen.ping.multicast.enabled Enable multicast ping discovery. Default `false`.

discovery.zen.ping.unicast.hosts List of master nodes in the cluster to perform discovery when new nodes are started. Default `127.0.0.1, [::1]`.

index.recovery.initial_shards Number of shards expected to be found on full cluster restart per index. Default `quorum`.

Network host aliases

- `_local_` : Will be resolved to the local ip address.
- `_non_loopback_` : The first non loopback address.
- `_non_loopback:ipv4_` : The first non loopback IPv4 address.
- `_non_loopback:ipv6_` : The first non loopback IPv6 address.
- `_[networkInterface]_` : Resolves to the ip address of the provided network interface. For example `_en0_`
- `_[networkInterface]:ipv4_` : Resolves to the ipv4 address of the provided network interface. For example `_en0:ipv4_`
- `_[networkInterface]:ipv6_` : Resolves to the ipv6 address of the provided network interface. For example `_en0:ipv6_`

Jetty HTTP Configuration

Jetty HTTP settings can be configured using `com.enonic.xp.web.jetty.cfg` file.

Listing 4.7: `$XP_HOME/config/com.enonic.xp.web.jetty.cfg`

```
# Set this if host name (or ip) needs to be fixed.
host =

# Socket timeout for connections.
timeout = 60000
```

```
# True to send server header.
sendServerHeader = false

# True to enable HTTP connections.
http.enabled = true

# Http port number to use.
http.port = 8080

# Session timeout (when inactive) in minutes.
session.timeout = 60

# Cookie name to use for sessions.
session.cookieName = JSESSIONID

# Enable GZIP compression for responses.
gzip.enabled = true

# Minimum number of bytes in response to consider compressing the response.
gzip.minSize = 16

# True to enable request logging.
log.enabled = false

# Request log file.
log.file = ${xp.home}/logs/jetty-yyyy_mm_dd.request.log

# True to append to the file, or create new one when started.
log.append = true

# True to use extended format.
log.extended = true

# Timezone to display timestamp in.
log.timeZone = GMT

# Number of days to retain the logs.
log.retainDays = 31
```

OSGi Shell Configuration

To enable or configure OSGi shell, use `com.enonic.xp.server.shell.cfg` file.

Listing 4.8: `$XP_HOME/config/com.enonic.xp.server.shell.cfg`

```
#
# Remote shell configuration
#

enabled = false
telnet.ip = 127.0.0.1
telnet.port = 5555
telnet.maxConnect = 2
telnet.socketTimeout = 0
```

DoS Filter Configuration

The DoS (denial of service) filter can be configured using `com.enonic.xp.web.dos.cfg` file.

Listing 4.9: `$XP_HOME/config/com.enonic.xp.web.dos.cfg`

```
# Enable dos filter (true/false)
enabled = false

# Maximum number of requests from a connection per second. Requests in excess of this are first delay
maxRequestsPerSec = 25

# Delay imposed on all requests over the rate limit. -1 = reject request, 0 delay.
delayMs = 100

# Length of time, in ms, to blocking wait for the throttle semaphore.
maxWaitMs = 50

# Number of requests over the rate limit able to be considered at once.
throttledRequests = 5

# Length of time, in ms, to async wait for semaphore.
throttleMs = 30000

# Length of time, in ms, to allow the request to run.
maxRequestMs = 30000

# Length of time, in ms, to keep track of request rates for a connection, before deciding that the us
maxIdleTrackerMs = 30000

# If true, insert the DoSFilter headers into the response.
insertHeaders = true

# If true, usage rate is tracked by session if a session exists.
trackSessions = true

# If true and session tracking is not used, then rate is tracked by IP+port (effectively connection)
remotePort = false

# A comma-separated list of IP addresses that will not be rate limited.
ipWhitelist =
```

Market Configuration

The market-place for installing applications can be configured using the `com.enonic.xp.market.cfg` file

Listing 4.10: `$XP_HOME/config/com.enonic.xp.market.cfg`

```
marketUrl = https://market.enonic.com/applications
```

UDC Configuration

UDC (Usage Data Collector) is collecting anonymous usage data 10 minutes after startup and every 24 hours. It is only used for finding out what platforms to focus on and improve platform stability. To switch this off, you can configure it sing the `com.enonic.xp.server.udc.cfg` file

Listing 4.11: `$XP_HOME/config/com.enonic.xp.server.udc.cfg`

```
#
# Set to false to disable usage data collector (UDC)
#
enabled = true
```

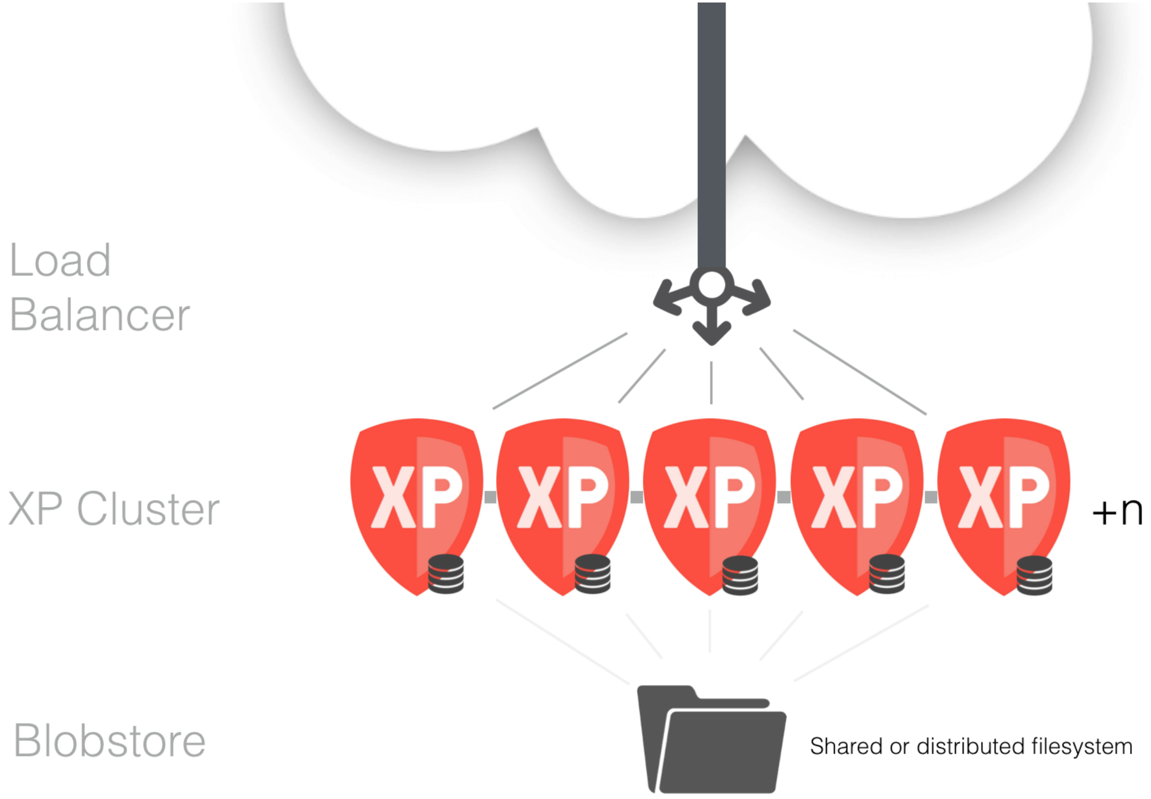
Clustering

Introduction

System Requirements

Enonic XP clusters have minimal requirements to infrastructure, it needs:

- Distributed (or shared) filesystem
- Load balancer - to make sure traffic is routed to different nodes



These components are standard ingredients in modern clouds and they are readily available as software as well. An XP cluster can also be launched on a regular computer for testing or development purposes.

Basic cluster setup on local machine

We have tried to make deployment of XP as simple and fail-safe as possible. By default it is configured to run on a local computer and it will not start looking for nodes in the network until you configure it to do so.

To test a cluster on your local machine, you need to do the following:

1. **Get two XP installations:** Download an $XP_DISTRO and copy it to a second $XP_DISTRO folder.

Typically, you will already have an XP-installation by now, so just copy the $XP_DISTRO folder to make another node.

2. **Share data:** Prepare a common place for storing data and configure both XP instances:

In `$XP_DISTRO/home/config/com.enonic.xp.blobstore.file.cfg` set the following property to point to a common directory:

```
    baseDir = /some/common/path
```

3. **Give each node its own HTTP-port:** Since you will run two nodes on the same machine, you also need to set two different HTTP-ports to be able to run two instances at once:

In `$XP_DISTRO/home/config/com.enonic.xp.web.jetty.cfg` set the following property to different values for the two nodes, typically `8080` and `8090`

```
    http.port = somePort
```

4. **Enable clustering:** In `$XP_DISTRO/home/config/com.enonic.xp.elasticsearch.cfg` set the property `node.local` to `false`

```
    node.local = false
```

5. **Start your cluster:** Start both nodes by their respective `bin/server.sh` or `bin/server.bat`. They will connect and you should have a live cluster on your machine. You can check the current cluster info at:

```
    http://localhost:8080/status/cluster
```

Note: By default, if no XP_HOME environment variable is set, the XP_HOME used is the one located in the XP_DISTRO/home folder which will work nicely for the above example. If you have set XP_HOME in the shell where you try to start the server, this will override the default settings. So for the above test, unset the XP_HOME variable if needed:

```
unset XP_HOME
```

Cluster configuration

There are a well of options at your disposal to configure and tune the cluster behavior. See *Elasticsearch configuration* for a subset of the available settings. All settings referred to in this chapter are set in `$XP_HOME/config/com.enonic.xp.elasticsearch.cfg` unless otherwise specified.

There are some key elements to consider when setting up a cluster:

1. Set up a shared storage for the nodes -> *Shared storage Configuration*
2. Make sure that nodes are connected -> *Network configuration*
3. Distribute the data between the nodes -> *Replica setup*
4. Ensure cluster data integrity -> *Cluster partition settings*
5. Ensure cluster stability -> *Cluster stability settings*
6. Make sure nodes recover correctly -> *Node recovery settings*

7. Monitoring the cluster -> *Cluster monitoring*
8. Deploying applications -> *Deploying Apps in cluster*
9. Securing data -> *Backing up a cluster*

Shared storage Configuration

For now, the nodes in the cluster need a shared storage to store data as files. Setting this up is highly individual for different operating systems and infrastructures, but as a basic guideline:

1. Get access to a shared or distributed file system and mount it on the nodes that will be part of the cluster
2. Configure `$XP_HOME/config/com.enonic.xp.blobstore.file.cfg` to point to the mounted storage:

```
baseDir = /path/to/shared/disk/folder
```

Network configuration

The nodes in a cluster need to be able to discover and communicate with other nodes in the network. The nodes communicate through TCP.

Each node binds to an IP-address and port, and communicates to other nodes specified in a list of other nodes bind addresses. Verify that your network allows TCP traffic on a specific port or port-range for the nodes to communicate and then configure the nodes to use these addresses.

Settings

node.local

When this setting is `true`, the node will never try to join a cluster. In all cluster setups, nodes must set this to `false`

network.host

The `network.host` setting specifies the TCP-address used for node communication. The default value for this is 127.0.0.1, which means that this node will never be able to talk to other nodes.

The `network.host` setting can be an explicit *IP-address*, a *host-name* or an *alias*. See the *Elasticsearch configuration* section for an overview.

transport.tcp.port

The `transport.tcp.port` value defines the port that the node will use for communication. This defaults to a range of ports; `9300-9400`, meaning that it will use the first available port in this range.

discovery.zen.ping.unicast.hosts

The `discovery.zen.ping.unicast.hosts` value contains a comma-separated list of nodes that are allowed to join the cluster. Each value is either in the form of `host:port` or `host:port1-port2` (port-range).

Sample config

```
transport.tcp.port = 9300-9400
network.host = _eth0:ipv4_
discovery.zen.ping.unicast.hosts = 10.0.6.47[9300-9400],10.0.6.49[9300-9400],10.0.6.73[9300-9400],10
```

Tip: Why aren't my nodes connecting

The most common issue is that the node binds to a different network address than specified in the unicast list. When a node starts, the log will show the current bind-address of the node in a message similar to this:

```
09:01:43.282 INFO  org.elasticsearch.http - [loadtest-appserver1] bound_address {inet[/10.0.6.49:9300
```

Make sure that the bind-addresses match those specified in the unicast-list. If it still doesn't work, it's time to blame the firewall or consult the *Troubleshooting*

Replica setup

Number of replicas

For a cluster to perform, each node must be able to do its share of work. Enonic XP searches for data in a number of Elasticsearch indices. An index can have a number of replicas (copies) spread around to the nodes in the cluster, so each node can query its local index for data.

The indices in Enonic XP have one replica configured by default. When a cluster has more than two nodes, this number must be increased to ensure that each node has a replica of the indices.

The number of replicas can be set at runtime with the Toolbox CLI *set-replicas*, and the recommended settings for replicas is `number of nodes - 1`

So for a 3 node cluster, the number of replicas should be set to 2.

Cluster partition settings

One of the main motivations of a cluster is to ensure that even if one or several nodes fail, the service you are providing should still be available. In an ideal world, a 100 node cluster should be fully operational even if 99 nodes are down. But in the real world, we also need to consider the cluster data integrity. This introduces a common dilemma in clustered environments; how to avoid the dreaded split-brain situation.

In a split-brain scenario, nodes get divided into smaller clusters that don't communicate with each other, and each cluster believing that the nodes in the other cluster are dead. This can easily happen in a cluster with 4 nodes on two different locations:

<<<Figure>>>

If the nodes on location-1 are disconnected from the master node on location-2, they will regroup and select a new master on location-1 and still provide service. The nodes on location-2 will assume that the nodes in location-1 are dead, so they will also continue serving requests. but they have no way of synchronizing data between the locations. This will break the integrity of the cluster and make data invalid.

To avoid this situation, there are a couple of basic properties of a cluster that should be ensured:

1. Beyond a two node cluster, there should be an odd number of nodes. So 1,2,3,5,7 etc are all acceptable cluster configurations.

2. When nodes are forming separate smaller clusters, only the cluster-partition with the majority of nodes should be fully operational and accept writes.
3. The minority cluster partitions can be allowed to serve read-only requests if that is acceptable for the provided service.

Settings

discovery.zen.minimum_master_nodes

This is the most important setting to set correctly to ensure cluster data integrity. A node will not accept requests before the number of 'minimum_master_nodes' are met. For instance, in a 3 node cluster with 3 master nodes and 'minimum_master_nodes' setting of '2', imagine that one of the nodes loose connection to the two other nodes. This node will only see one possible master node (itself) and will not accept requests. The remaining two other nodes will still work, and when the lost node reconnects again, it will get the fresh data from the other nodes and rejoin the cluster.

Important: As a rule of thumb, this setting should be set to N/2+1, where N is the total number of nodes. So for a 5 node cluster, discovery.zen.minimum_master_nodes = 5/2+1 = 3 (rounding down to the nearest integer)

So what about a 2 node cluster? It will be impossible to avoid a possible split-brain scenario with this setup. It's highly recommended to add one node as a tie-breaker. This node may act as a dedicated master node (with *node.data = false*, see *Cluster stability settings*) which enables it to run on less expensive hardware since it will not handle any external requests.

Tip: Why nodes leave the cluster

There are 2 main reasons why cluster nodes leave the cluster

1. Network failure
2. Node not responding

Network failures

Network failures are the main reason for cluster stability-issues. The problems could have any number of reasons, from a router breaking down to complex scenarios where e.g a firewall cuts the connection in one direction between two nodes

Node not responding

If a node does not get a response on a ping to the master node within a set timeout, it will consider it as dead and invoke an election process. Likewise, the master node expects that a slave node will respond within a certain amount of time. This is usually caused by a node doing a stop-the-world garbage collection, and not being able to respond to the request at all for a period of time.

Cluster stability settings

In a low load environment, there is probably no need to do a lot of tuning since it will perform acceptable with the default setup. If you expect heavy load, there are a couple of things to consider when setting up the cluster topology.

Dedicated master nodes

A cluster consists of a number of nodes sharing data and state between them. A cluster needs to have exactly one node acting as a master-node at any time. The master-node is responsible for managing the cluster-state. In a busy cluster, the master-nodes will have to do a lot of work to ensure that all other nodes get the needed information.

Since the cluster stability depends on a healthy master node, it may be a good idea to set aside a number of nodes as *dedicated master nodes*. These dedicated master nodes should not be handling external requests, but rather concentrate on keeping the cluster nodes in sync and stable.

A node can be configured to be allowed to act as a master-node by the setting `node.master`.

A dedicated master node should have the following settings:

```
node.master = true
node.data = false
```

Data nodes

Data nodes are the workhorses of the cluster. They will handle the bulk load of the requests, depending on the master node to keep them in sync. These nodes need the most memory and CPU power.

A dedicated data node should have the following settings:

```
node.master = false
node.data = true
```

Node recovery settings

Node recovery happens when a node starts or reconnects to the cluster after a e.g a network shortage.

Consider a cluster of 2 nodes. When a node starts for the first time, it will try to connect to a cluster. If no master found, it will elect itself as master, then proceed to initialize the index-data locally. If it does find an existing master node, it will require the master to provide it with data. This is all good, but there may occur situations where a new node in an existing cluster starts initializing data before the nodes with existing data can inform the new node that there is already data in the cluster.

Settings

gateway.recover_after_nodes

Defaults to 1. Do not start the recovery of local indices before this number of nodes (master or data) has joined the cluster.

gateway.recover_after_master_nodes

Defaults to 0. Do not start the recovery of local indices before this number of master-nodes is present in the cluster.

Cluster monitoring

See *Cluster monitoring*

Deploying Apps in cluster

To deploy applications in a cluster you need to deploy the application to every node, as loading and installation of apps is done on a per-node basis. This also means you can choose what applications to deploy on each node.

Warning: Remember that XP only supports running one version of an application at any time. So don't leave the old versions of your applications in the deploy directory.

Backing up a cluster

Backing up a cluster is done in the same way as backing up a single node installation, the only difference is that the `snapshots.dir`-option should point to a shared file system location, see *Storage Configuration*.

1. First, on any cluster node, take a :*toolbox-snapshot* of the indices. This will store a cluster-wide snapshot of all data at a point of time. This can be configured to run as an automatic job; Only the diff from the last snapshot will be stored, so the operation is quick.
2. Second, take a file copy of your blobstore.

We recommend uisng incremental backup for the blobstore (rsync or similar) as this will only copy the recently changed files. The combined data from the snapshots and blobstore copy is all you need in order to restore Enonic XP.

Sample configurations

2-node cluster

```
node.name = local-node-1
node.master = true
node.local = false

cluster.name = mycluster

network.host = _en0_
transport.tcp.port = 9300-9400

gateway.expected_nodes = 1
gateway.recover_after_nodes = 1
discovery.zen.minimum_master_nodes = 1
discovery.zen.ping.unicast.hosts = <node1Address>,<node2Address>
```

3-node cluster

```
node.name = local-node-1
node.master = true
node.local = false

cluster.name = mycluster

network.host = _en0_
transport.tcp.port = 9300-9400

gateway.expected_nodes = 2
gateway.recover_after_nodes = 2
discovery.zen.minimum_master_nodes = 2
discovery.zen.ping.unicast.hosts = <node1Address>,<node2Address>,<node3Address>
```

5-node cluster

```
node.name = local-node-1
node.master = true
node.local = false

cluster.name = mycluster

network.host = _en0_
transport.tcp.port = 9300-9400

gateway.expected_nodes = 3
gateway.recover_after_nodes = 3
discovery.zen.minimum_master_nodes = 3
discovery.zen.ping.unicast.hosts = <node1Address>,<node2Address>,<node3Address>,<node3Address>,<node5
```

7-node cluster with dedicated roles

```
# NODE1 - NODE-3 dedicated masters
# --------------------------------

node.name = node-1/node-2/node-3
node.master = true
node.data = false
node.local = false

cluster.name = mycluster

network.host = _en0_
transport.tcp.port = 9300-9400

gateway.recover_after_master_nodes = 2
discovery.zen.minimum_master_nodes = 2
discovery.zen.ping.unicast.hosts = <node1Address>,<node2Address>,<node3Address>,<node3Address>,<node5


# NODE4 - NODE-7 dedicated data nodes
# -----------------------------------

node.name = node-4/node-5/node-6/node-7
node.master = false
node.data = true

cluster.name = mycluster

network.host = _en0_
transport.tcp.port = 9300-9400

gateway.recover_after_master_nodes = 2
discovery.zen.minimum_master_nodes = 2
discovery.zen.ping.unicast.hosts = <node1Address>,<node2Address>,<node3Address>,<node3Address>,<node5
```

Monitoring

We provide some basic metric tools for monitoring which are easily accessed in a simple JSON format. To access the monitoring JSON feed you can point to the following url:

```
http://localhost:8080/status
```

This will give you a list of status-reporters. Each reporter has a name and can be accessed using the following pattern:

```
http://localhost:8080/status/<name>
```

Here's a list of all the status pages and what is shows:

`cluster` Information of the current cluster. Local-node and members.

`dump.deadlocks` This will try to detect thread deadlocks and show them if any.

`dump.threads` Dumps all current thread-states.

`index` Shows ElasticSearch index status.

`jvm.gc` Information about JVM GC status.

`jvm.info` General JVM information (version, vendor, uptime).

`jvm.memory` JVM memory information (heap and non-heap).

`jvm.os` Information about OS (name, version, architecture).

`jvm.properties` Shows all JVM properties.

`jvm.threads` JVM thread stats (count, peak, total).

`metrics` Shows metrics. The information can be filtered using `?filter=...`.

`osgi.bundle` Information about all OSGi bundles.

`osgi.component` Information about registered SCR OSGi components.

`osgi.service` Shows all OSGi services registered.

`server` Information about the server (version, build).

Cluster monitoring

There are two tools at your disposal for monitoring the health of the cluster and indices:

Cluster health

```
http://<host>:<port>/status/cluster
```

Which should give you a response like this:

```
{
    "localNode": {
        "hostName": "Runars-MacBook-Pro.local",
        "id": "xUfgZ2FdRu6q-CLJA4CIkA",
        "isMaster": true,
        "numberOfNodesSeen": 2,
        "version": "1.4.4"
    },
    "members": [
        {
            "address": "inet[/127.0.0.1:9300]",
            "hostName": "Runars-MacBook-Pro.local",
            "id": "xUfgZ2FdRu6q-CLJA4CIkA",
            "isMaster": true,
            "version": "1.4.4"
        },
        {
            "address": "inet[/127.0.0.1:9301]",
            "hostName": "Runars-MacBook-Pro.local",
            "id": "yvHpvM7KQn6y6yTZ-QIRNw",
            "isMaster": false,
            "version": "1.4.4"
        }
    ],
    "name": "mycluster",
    "state": "YELLOW"
}
```

This view gives a brief overview of the nodes in the cluster. For convenience, the current local node to which the request was made has a separate entry in addition to being in the list of members.

The "state" property is the most important:

- **Green**: Cluster is operational and all configured replicas are distributed to a node
- **Yellow**: Cluster is operational, but there are replicas that are not distributed to any node
- **Red**: Cluster is not operational

To see the details about how the replicas are distributed, let's continue to the `Index stats` report:

Index stats

```
http://<host>:<port>/status/index
```

Which should give you a response like this:

```
{
    "summary": {
        "total": 8,
        "started": 4,
        "unassigned": 4
        "initializing": 0,
        "relocating": 0,
     },
    "shards": {
        "initializing": [],
        "relocating": [],
        "started": [
            {
                "id": "search-cms-repo(0)",
                "nodeAddress": "10.0.6.146",
                "nodeId": "xUfgZ2FdRu6q-CLJA4CIkA",
                "type": "PRIMARY"
            },
            {
                "id": "search-system-repo(0)",
                "nodeAddress": "10.0.6.146",
                "nodeId": "xUfgZ2FdRu6q-CLJA4CIkA",
                "type": "PRIMARY"
            },
            {
                "id": "storage-system-repo(0)",
                "nodeAddress": "10.0.6.146",
                "nodeId": "xUfgZ2FdRu6q-CLJA4CIkA",
                "type": "PRIMARY"
            },
            {
                "id": "storage-cms-repo(0)",
                "nodeAddress": "10.0.6.146",
                "nodeId": "xUfgZ2FdRu6q-CLJA4CIkA",
                "type": "PRIMARY"
            }
        ],
        "unassigned": [
            {
                "id": "search-cms-repo(0)",
                "nodeAddress": "UNKNOWN",
                "type": "REPLICA"
            },
            {
                "id": "search-system-repo(0)",
                "nodeAddress": "UNKNOWN",
                "type": "REPLICA"
            },
            {
                "id": "storage-system-repo(0)",
                "nodeAddress": "UNKNOWN",
                "type": "REPLICA"
            },
            {
                "id": "storage-cms-repo(0)",
                "nodeAddress": "UNKNOWN",
                "type": "REPLICA"
            }
        ]
    }
}
```

This gives an overview of how the indices are distributed and what state the index parts (**shards**) are currently in. A shard could be either `PRIMARY` or `REPLICA` (copy of a primary shard). These are the possible states:

- **total**: Total number of index parts (e.g two repositories with two indices with one replica for each index)
- **started**: Shards that are currently assigned to a node
- **unassigned**: Shards waiting to be distributed to a node. Typically a setup with a number of replicas where one or more nodes are not running
- **relocating**: Shards that are currently moving from one node to another
- **initializing** Shards that are currently being recovered from disk at startup.

The `shards` section gives a more detailed overview on the shard distribution.

Install as service

When installing Enonic XP on a standard production server, you will want to set it up to run as a service.

First, make sure the correct version of Java is installed on your system. See *Install Java* for guidance.

Linux

Install with script

> **Attention:** The script is written for and tested on Ubuntu / CentOS. For other distrubutions, the manual installation may be neccessary.
>
> ***Prerequisites***
>
> - User with *sudo* rights
> - Java JRE 1.8+ installed
> - `/lib/lsb/init-functions` installed

1. Download the Enonic XP distribution
2. Unzip the distributiopn
3. Run script: `sudo ./enonic-xp-6.8.1/service/install_service.sh`
4. *Optional:* Set **JAVA_HOME** and **JAVA_OPTS** variables in `/etc/xp.conf`
5. Start service: `sudo service xp start`
6. Check log: `sudo tail -f /home/xp/enonic/xp/logs/server.log`

Manual installation

See linux-detailed-service-install

Windows

Info on running XP as a service in windows will come later.

Backup and Restore

Backing up your data is vital for any installation.

All the data in an Enonic XP installation is stored in `$XP_HOME/repo`. This directory has two folders: `blob` and `index`.

The `blob` folder contains all files needed by the system to manage your data, while the `index` folder contains the Elasticsearch index folders. These are dependent on each other in the sense that one is not much use without the other.

That leaves us with ensuring that two elements are safely stored for retrieval in an emergency:

- `$XP_HOME/repo/blobs`
- `$XP_HOME/repo/index`

Backup vs Export

The export/import enables you to export your data to a serialized format. The serialized data could then be imported into another instance. This is very useful, but is not optimal for a backup/restore scheme since it requires some work to get things up and running again, especially when working with big installations with a lot of data. The backup/restore-process described below on the other hand, should enable a quick and safe way to get your system back to operation when in a hurry.

See *Export and Import* for more information on export/import.

Backing up blobs

The blobs are just files on a filesystem. This should be backed up by your preferred way of doing file-backups.

The folder to backup is:

- `$XP_HOME/repo/blobs`

Backing up indexes

Backing up the indices is a bit more complex than just copying the index-folder since it involves floating data with state, especially in a clustered environment. To help you out, we have a snapshot-API. A snapshot is exactly that; a snapshot of the indices state at a point of time. There are 4 rest-resources at your disposal.

`http://<your-installation>/admin/rest/repo/snapshot` Stores a snapshot of the current indices state.

`http://<your-installation>/admin/rest/repo/list` Returns a list of available snapshots for the installation.

`http://<your-installation>/admin/rest/repo/restore` Restore a snapshot of the indices state.

`http://<your-installation>/admin/rest/repo/delete` Deletes a snapshot or a group of snapshots.

Snapshot

The snapshot rest-service accepts a JSON in this format:

```
{
  "repositoryId": "<repository-id>"
}
```

A snapshot of the given repository will be created for later retrieval. Each subsequent snapshot will store the changes between this snapshot and the last snapshot of the given repository. This means that only changed data are stored when doing subsequent snapshots. The default location where snapshots are stored is `$xp_home/snapshots`. A name of the snapshot will be given at snapshot-time, and returned in the snapshot-result.

To ease the process, we have provided a *snapshot* tool.

Restore

The restore rest-service accepts a JSON in this format:

```
{
  "snapshotName": "<snapshot-name>",
  "repository" : "<repository-id>"
}
```

The indices will be closed for the duration of a restore operation, meaning that no request will be accepted while the restore in running. To ease the process, we have provided a *restore* tool.

> **Warning:** Restoring a snapshot will restore data to the exact state of the indices at the snapshot-time, meaning all other changes will be lost.

Delete

The delete rest-service accepts a JSON in this format:

```
{
  "snapshotNames": ["name1", "name2"],
  "before" : "<timestamp>"
}
```

Deletes either all snapshots before timestamp, or given snapshots by name. To ease the process, we have provided a *delete-snapshots* tool.

Export and Import

Exporting and importing data in your Enonic XP installation is useful both for securing data and migrating between installations. Enonic XP ships with a set of tools (*Toolbox CLI*) to ease the operation of exporting and importing data from the system.

> **Caution:** At the moment, exporting and importing data can only be done to and from files on the same server running Enonic XP.

Content Export/Import vs System Dump/Load

Both enable you to export your data to a serialized format and import the serialized data into another instance. But, while the export/import focuses on a given content, the dump/load is used to export an entire system (all repositories and branches). This is used, for example, to export the entire system when doing an upgrade.

Content Export

The export operation will extract data for a given content URL and store it as XML in a sub-folder under `$XP_HOME/data/export`. The REST service for export is found at the following URL:

```
http://<host>:<port>/api/repo/export
```

The export REST service accepts a JSON in this format:

```
{
  "sourceRepoPath": "<source-repo-path>",
  "exportName": "<name>",
  "exportWithIds": <true|false>,
  "dryRun": <true|false>
}
```

To ease the process, we have provided an *export* tool.

Content Import

The import will take data from a given export directory and load it into Enonic XP at the desired content path. The REST service for import is found at the following URL:

```
http://<host>:<port>/api/repo/import
```

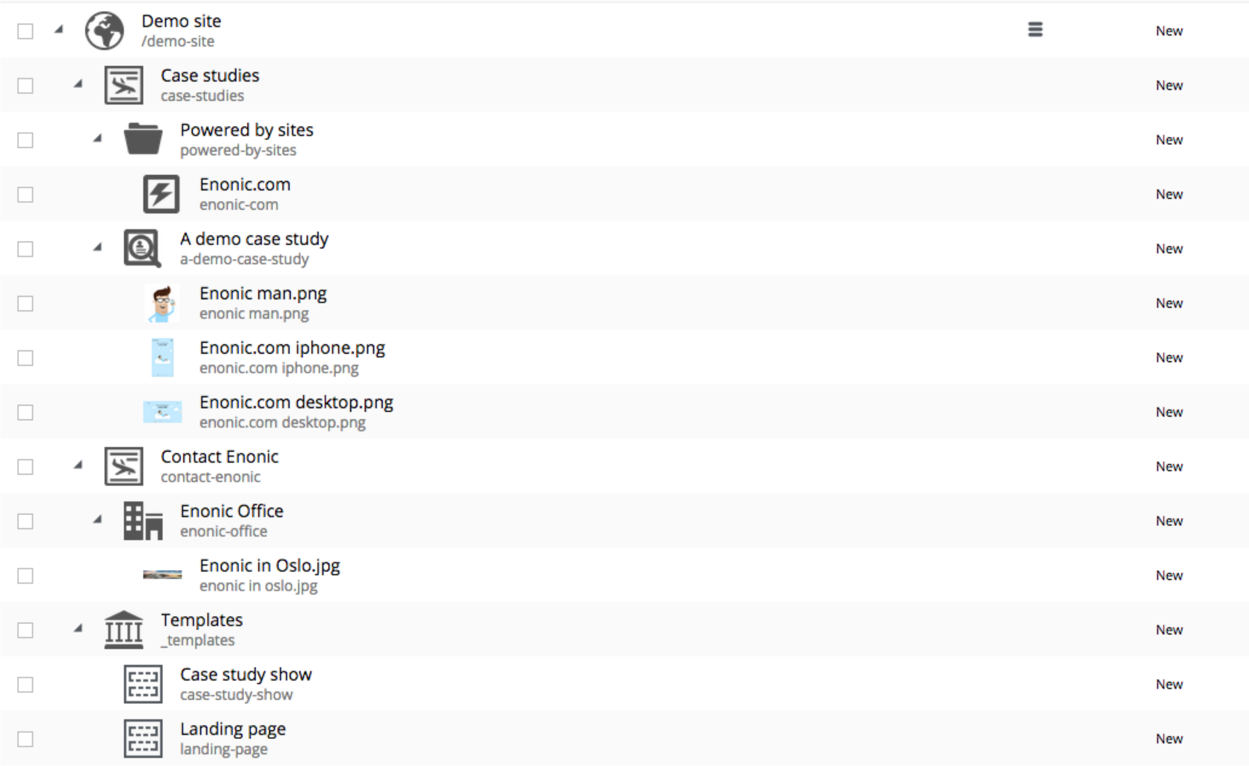
The import REST service accepts a JSON in this format:

```
{
  "exportName": "<name>",
  "targetRepoPath": "<target-repo-path>",
  "importWithIds": <true|false>,
  "importWithPermissions": <true|false>,
  "dryRun": <true|false>
  "xslSource": "<xsl-file-path>"
  "xslParams": {
    "<name>": "<value>"
  }
}
```

To ease the process, we have provided an *import* tool.

Content Export data structure

Let’s look at how this works. The following structure will be exported:



□	Demo site /demo-site	New
□	Case studies case-studies	New
□	Powered by sites powered-by-sites	New
□	Enonic.com enonic-com	New
□	A demo case study a-demo-case-study	New
□	Enonic man.png enonic man.png	New
□	Enonic.com iphone.png enonic.com iphone.png	New
□	Enonic.com desktop.png enonic.com desktop.png	New
□	Contact Enonic contact-enonic	New
□	Enonic Office enonic-office	New
□	Enonic in Oslo.jpg enonic in oslo.jpg	New
□	Templates _templates	New
□	Case study show case-study-show	New
□	Landing page landing-page	New

Run the export command:

```
$ ./toolbox.sh export -a su:password -s cms-repo:draft:/ -t myExport
```

Below is the resulting structure in the export folder `$XP_HOME/data/export/myExport`:

```
./content
./content/_
./content/_/node.xml
./content/demo-site
./content/demo-site/_
./content/demo-site/_/manualChildOrder.txt
./content/demo-site/_/node.xml
./content/demo-site/_templates
...
./content/demo-site/case-studies
./content/demo-site/case-studies/_
./content/demo-site/case-studies/_/node.xml
./content/demo-site/case-studies/a-demo-case-study
...
./content/demo-site/case-studies/a-demo-case-study/enonic man.png
./content/demo-site/case-studies/a-demo-case-study/enonic man.png/_
./content/demo-site/case-studies/a-demo-case-study/enonic man.png/_/bin
./content/demo-site/case-studies/a-demo-case-study/enonic man.png/_/bin/Enonic man.png
...
./content/demo-site/case-studies/powered-by-sites
...
./content/demo-site/contact-enonic
...
```

`content` The base folder of the export. All content in `cms-repo` has this as root path.

`content/_` All folders named _ are system folders for the data at the current level.

`content/_/node.xml` The definition of the node, e.g. all data for the current node

`content/demo-site` This is the site from the screenshot above.

`content/demo-site/_/manualChildOrder.txt` Our demo-site has manually ordered children, this file contains an ordered list of children.

`content/demo-site/case-studies` This 'case-studies' content is the first element in the site.

`content/demo-site/case-studies/a-demo-case-study/enonic man.png/_/bin` The `A demo case study` content has a binary attachment called `Enonic man.png`. The folder `_/bin` contains the actual binary files.

Changing export data

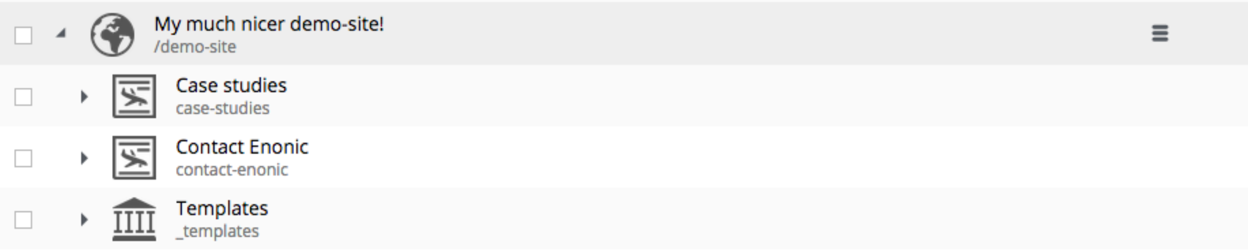
It is possible to make manual changes to the exported data before importing.

Using the above export as an example, the `demo-site` displayName can be changed to something more suitable:

```
myExport $ vi content/demo-site/_/node.xml

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<node xmlns="urn:enonic:xp:export:1.0">
  <id>2dfbdc41-af98-4b3c-a2a9-9dc4814d003a</id>
  <childOrder>_manualordervalue DESC</childOrder>
  <nodeType>content</nodeType>
  <data>
    <boolean name="valid">true</boolean>
    <string name="displayName">My much nicer demo-site!</string>
    <string name="type">portal:site</string>
    <string name="owner">user:system:su</string>
```

After some data has been changed, it can be imported again:

```
$ ./toolbox.sh import -a su:password -s myExport -t cms-repo:draft:/
```



My much nicer demo-site!
/demo-site

Case studies
case-studies

Contact Enonic
contact-enonic

Templates
_templates

Caution: Editing exported data is experimental at the moment and will potentially cause trouble if not done carefully. For exports without ids, references will be broken and must be fixed manually. When importing *with* ids onto existing data, renaming and changing manual order will not yet work as expected.

System Dump

The dump operation will extract data from your entire system and store it as XML in a sub-folder under `$XP_HOME/data/dump`. The REST service for export is found at the following URL:

```
http://<host>:<port>/api/system/dump
```

The dump REST service accepts a JSON in this format:

```
{
  "name": "<dump-name>"
}
```

To ease the process, we have provided a *dump* tool.

System Load

The load operation will take data from a given dump directory and load it into Enonic XP. The REST service for load is found at the following URL:

```
http://<host>:<port>/api/system/load
```

The export REST service accepts a JSON in this format:

```
{
  "name": "<dump-name>"
}
```

To ease the process, we have provided a *load* tool.

Troubleshooting

This document is an up-to-date list of known problems (and how to fix them) for our current release and development releases.

Wrong Java version

Verify that Java 1.8 (update 92 or higher) is installed and that this version is actually used.

Run *java -version* in the shell where you attempt to start Enonic XP:

```
$ java -version
java version "1.8.0_92"
Java(TM) SE Runtime Environment (build 1.8.0_92-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.92-b14, mixed mode)
```

The boot log will also output the version of Java that was actually used.

If the Java version does not match your expected version, make sure that the `JAVA_HOME` environment variable is set correctly. For OS X and Linux users - execute the following in your command line:

```
export JAVA_HOME=`/usr/libexec/java_home -v 1.8`
```

Optionally add the line to your `~/.properties` file to make the change persistent.

Check the *Troubleshooting Java* page for more Java help.

Port 8080 already taken

A lot of different web software defaults to port 8080. If you find that the log is complaining about this, simply identify the other software you have running on this port and stop it.

If shutting down other software that uses port 8080 is not an option, you may set a different port for Enonic XP. See *Configuration*.

Unexpected behavior

While frequently redeploying an app during development, some instability or unexpected behavior may be noticed. This can be caused by certain changes to the app files. For example, changing the app name in the build.gradle file, or deleting content import node.xml files. When this occurs, the project may need a clean build `gradle clean build`. Sometimes the app JAR file may need to be deleted from the $XP_HOME/deploy directory as well, and then replaced with the clean build JAR file.

Having two versions of the same app in the $XP_HOME/deploy folder will cause problems.

Cannot login after install

There could be a problem with file permissions on Windows if Enonic XP was unzipped and started from within the "My Documents" folder. This may allow XP to start, but the users cannot log in. The solution would be to unzip the Enonic XP distribution outside of the "My Documents" folder, or to manually change the file permissions.

Sending email with lib-mail not working

Check the installation's *Mail Configuration*.

Troubleshooting Java

Important: This documentation is based on OSX, Windows version coming later..

Check current JDK version

To see all installed JDK's on you environment, if any, type the following command in your terminal:

```
~/ $ /usr/libexec/java_home -V
Matching Java Virtual Machines (4):
    1.8.0_45, x86_64:  "Java SE 8"     /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Hor
    1.8.0_40, x86_64:  "Java SE 8"     /Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Hor
    1.8.0_20, x86_64:  "Java SE 8"     /Library/Java/JavaVirtualMachines/jdk1.8.0_20.jdk/Contents/Hor
    1.7.0_67, x86_64:  "Java SE 7"     /Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Hor
~/ $
```

If you have a JDK equal to or above version 1.8.0_92, but the `javac -version` points to another version, proceed to set the JAVA_HOME environment variable correctly.

Setting JAVA_HOME

If you don't have a JDK equal to or above version 1.8.0_92, you must install a newer version.

- Go to the JDK page and click the "JDK download" button. This button will always link to the latest version.
- Follow the installation instructions.

- After installation, proceed to check that the JAVA_HOME environment variable is set correctly.

To check the current JAVA_HOME environment variable:

```
~/ $ echo $JAVA_HOME
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home
~/ $
```

If this is not set correctly, you must set the correct one either for this terminal session, or in your `.profile` or `.bashrc` for all terminal sessions.

- To set the correct `JAVA_HOME` for the current terminal session; invoke the following command in your terminal. This command will set `JAVA_HOME` to the newest installed 1.8 JDK-version:

```
~/ $ export JAVA_HOME=`/usr/libexec/java_home -v 1.8`
```

- To set for all terminal sessions, add the entry to either your `~/.bash_profile` or `~/.profile.`

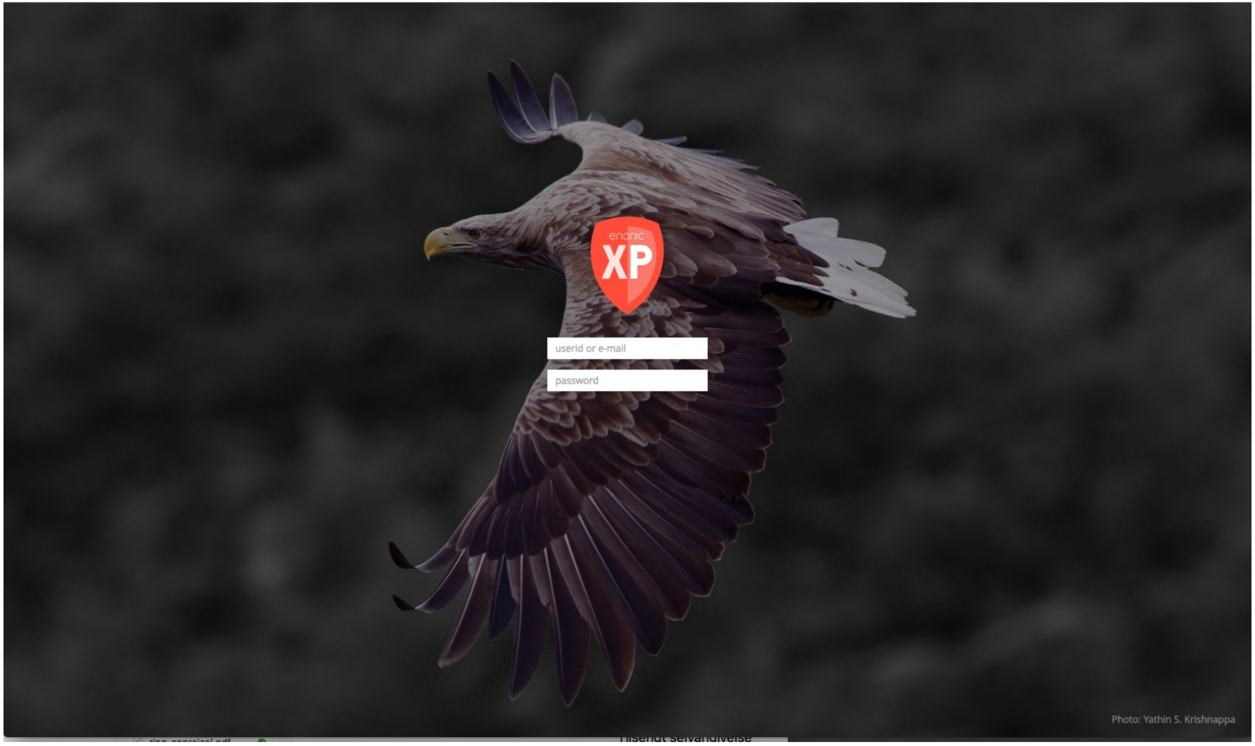
```
$ vim .bash_profile

export JAVA_HOME=$(/usr/libexec/java_home -v 1.8)

$ source .bash_profile

$ echo $JAVA_HOME
/Library/Java/JavaVirtualMachines/1.8.0_45.jdk/Contents/Home
```

CHAPTER 5

Admin Tools

Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

When visiting the Enonic XP admin interface - you will be required to log in. Unless otherwise configured, the standard login screen will be displayed. If this is a test-installation, you can login with “su” and “password”.

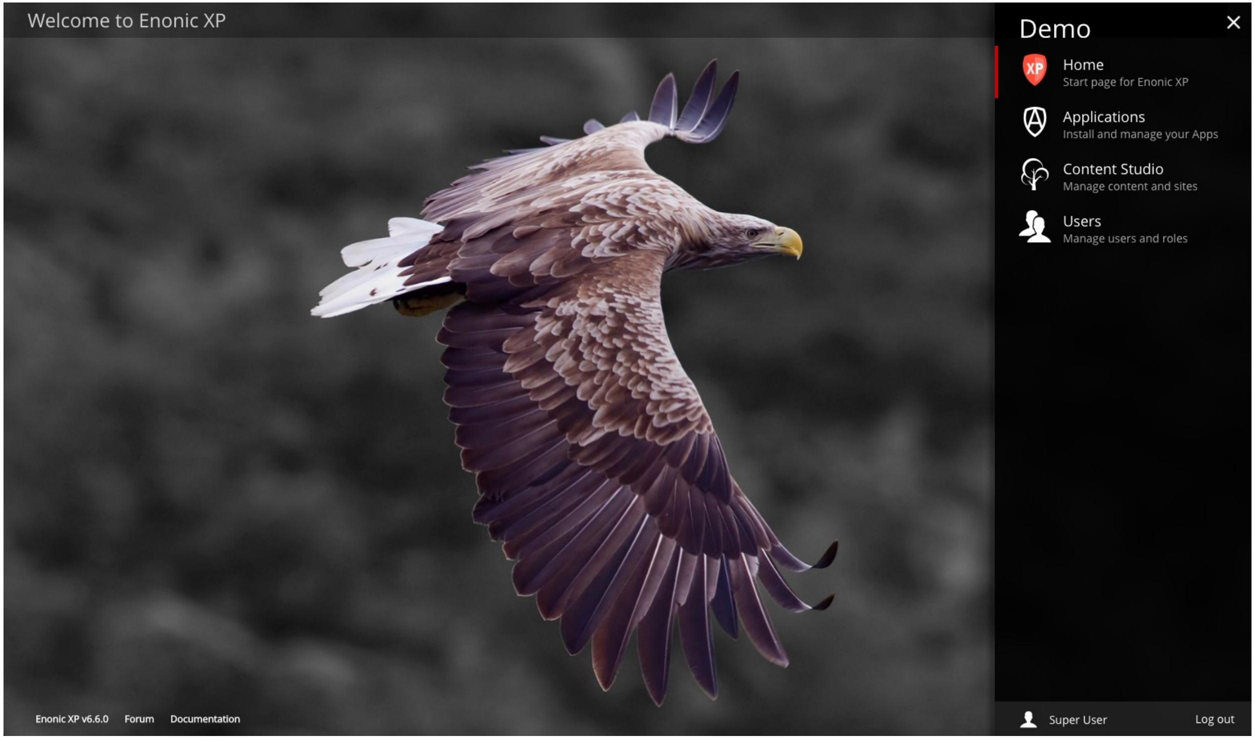


Home

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Upon successful login, the Home page appears as seen in the image below. The **Launcher Panel** is on the right and the space on the left is reserved for future use. The footer has the software version and links to the forum and documentation.

Future releases will include more functionality for the Home Tool.



Launcher panel

The launcher panel lists the installed admin tools in alphabetical order. These will open in a new tab when selected, but holding the left-click will cause the selected tool to open in the same tab. The bottom of the Launcher Panel has the name of the logged in user and a logout button.

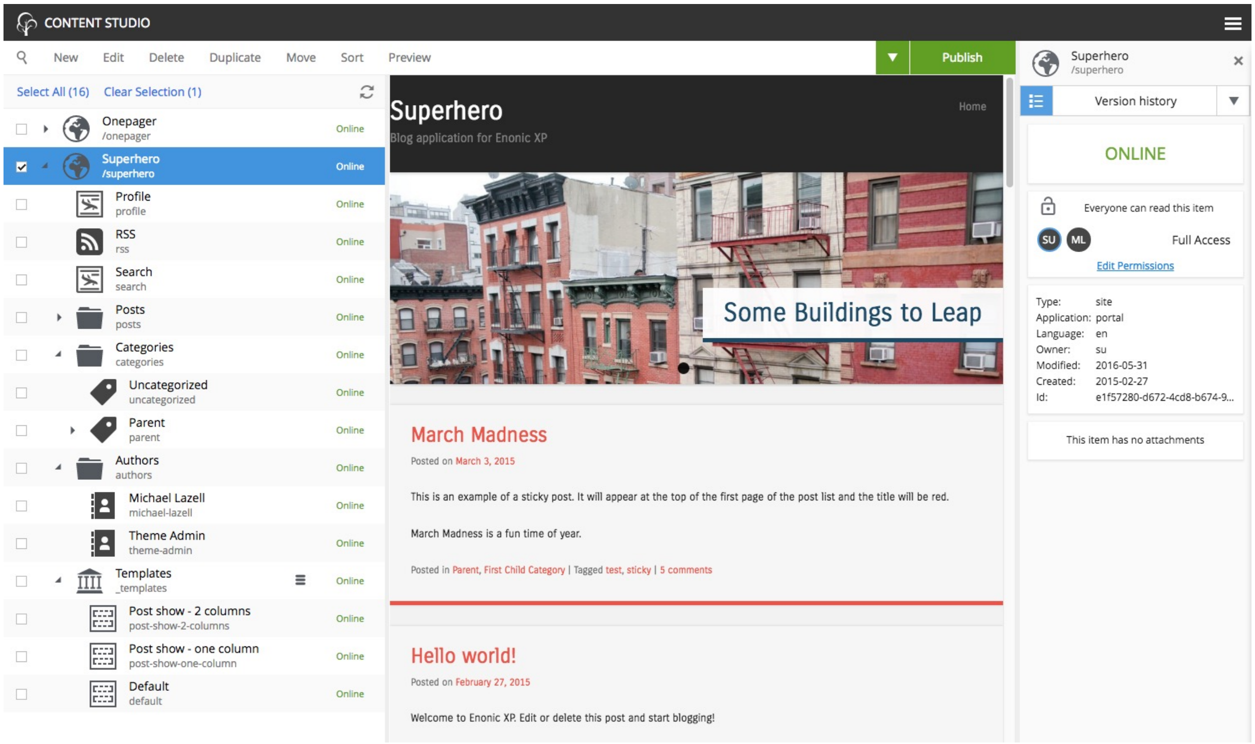
Content Studio

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The Content Studio is the Web CMS interface of Enonic XP. It is used for managing and publishing content. It also provides the tools for building and extending web applications from the components of the installed apps.

Opening the Content Studio reveals the *Browse View* with a menu bar, content tree grid, preview panel and details panel. The **Edit view** appears when content is created or edited. All actions and features will be described in the following pages.

Note: The Content Studio is fully responsive, so panels may stack or collapse if the screen is small or narrow. This documentation will assume a large desktop view.



Posted on March 3, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

March Madness is a fun time of year.

Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Posted on February 27, 2015

Welcome to Enonic XP. Edit or delete this post and start blogging!

Everyone can read this item

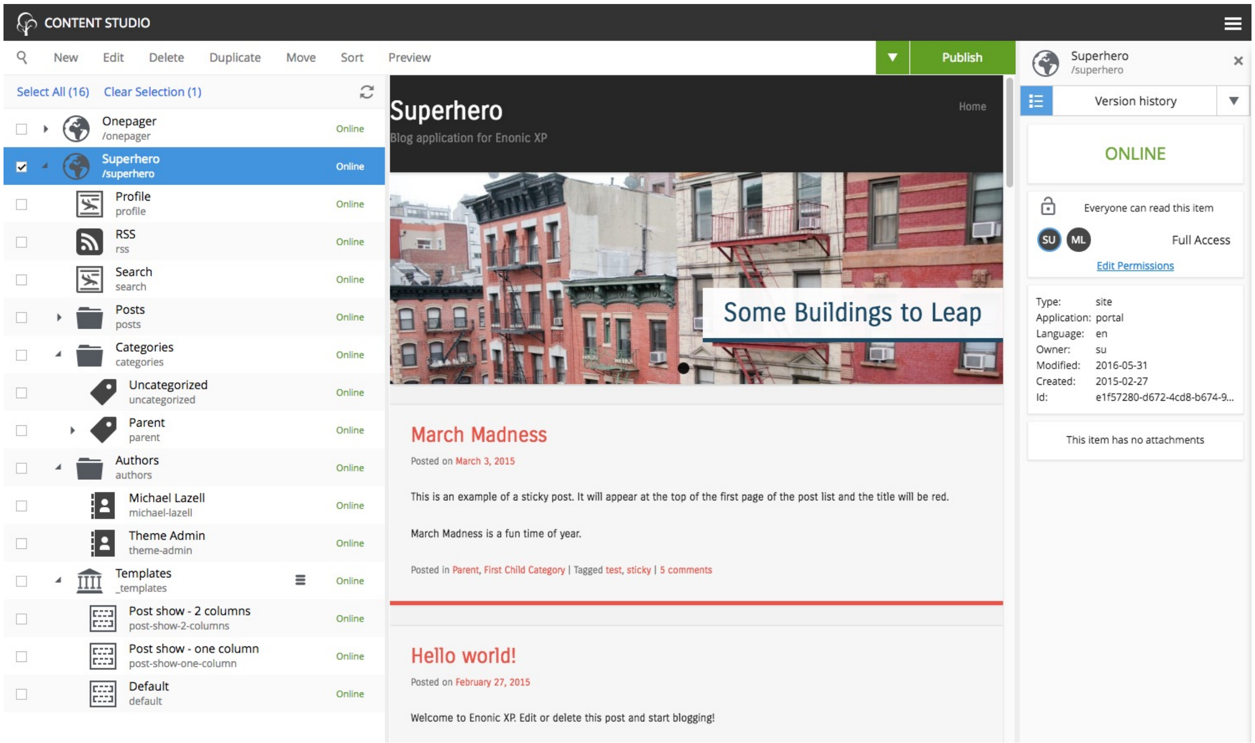
Type: site
Application: portal
Language: en
Owner: su
Modified: 2016-05-31
Created: 2015-02-27
Id: e1f57280-d672-4cd8-b674-9...

This item has no attachments

Browse View

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The **Browse view** is the default page of the Content Studio. It consists of a menu bar on top and two or three panels below it. On the left is the *Content tree grid* where all the sites and content can be found. To the right of the tree grid is the *Preview panel* which displays a preview of the selected content. On the far right is the *Detail panel* that may be toggled on and off with the icon that has three small squares next to horizontal bars.


CONTENT STUDIO
New
Edit
Delete
Duplicate
Move
Sort
Preview
Publish
Select All (16)
Clear Selection (1)
Onepager /onepager Online
Superhero /superhero Online
Profile profile Online
RSS rss Online
Search search Online
Posts posts Online
Categories categories Online
Uncategorized uncategorized Online
Parent parent Online
Authors authors Online
Michael Lazell michael-lazell Online
Theme Admin theme-admin Online
Templates _templates Online
Post show - 2 columns post-show-2-columns Online
Post show - one column post-show-one-column Online
Default default Online
Superhero
Blog application for Enonic XP
Home
Some Buildings to Leap
March Madness
Hello world!
Superhero /superhero
Version history
ONLINE
Everyone can read this item
SU
ML
Full Access
Edit Permissions

Posted on March 3, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

March Madness is a fun time of year.

Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Posted on February 27, 2015

Welcome to Enonic XP. Edit or delete this post and start blogging!

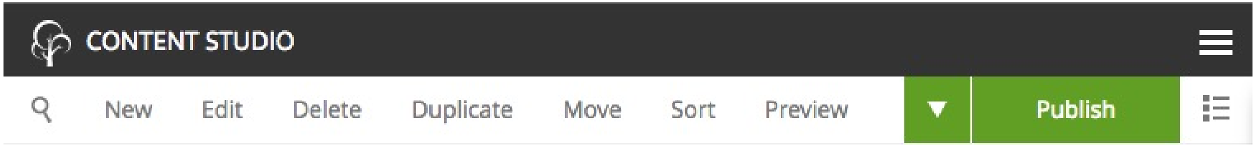
Type: site
Application: portal
Language: en
Owner: su
Modified: 2016-05-31
Created: 2015-02-27
Id: e1f57280-d672-4cd8-b674-9...

This item has no attachments

Toolbar

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The toolbar in the browse view has buttons for New, Edit, Delete, Duplicate, Move, Sort, Preview and Publish. These options are also available in the context menu that appears when right-clicking on a content in the tree grid panel.

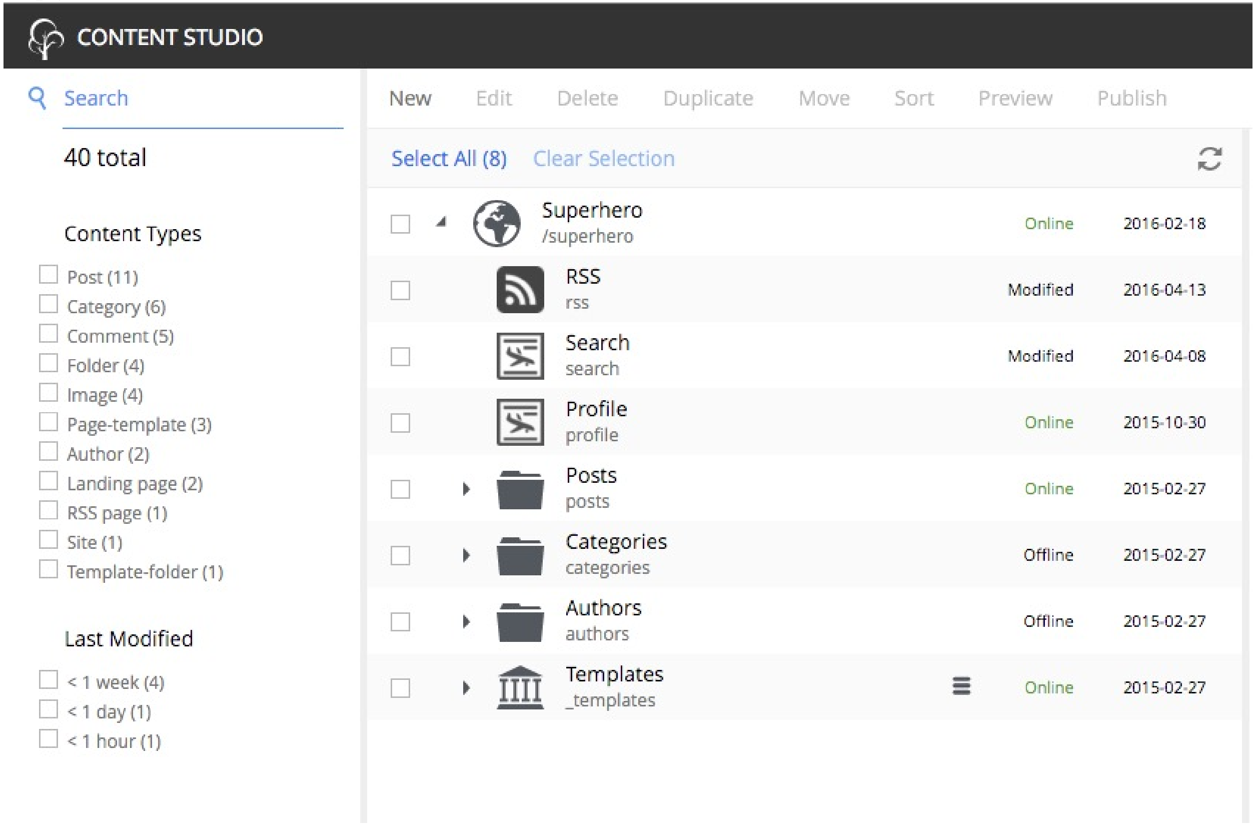


- **New**: Opens the *Create Content* dialogue for creating new content.
- **Edit**: Opens the content for editing in a new internal tab. See *Edit View*.
- **Delete**: Opens the **Delete item** dialogue for the selected content. Read about *Delete*.
- **Duplicate**: Makes a copy of the selected content. See *Duplicate*.
- **Move**: Opens the **Move item with children** dialogue. See *Move content*.
- **Sort**: Opens the **Sort items** dialogue where child items of the selected content can be sorted. See *Sorting content*.
- **Preview**: Opens the selected content in preview mode, in a new browser tab. If more than one content is selected, each will open in its own tab. Only content that has a supporting page template can be previewed. Read more about *Page Templates*.
- **Publish**: Opens the **Publishing Wizard** dialogue. Read more about *Publishing content*.

Search Panel

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The search panel can be toggled with the magnifying glass icon at the far left of the menu bar. When opened, it has a text search field and categories to limit search results by Content Types and Last Modified. All content will match an empty search field.



40 total

Content Types

□ Post (11)
□ Category (6)
□ Comment (5)
□ Folder (4)
□ Image (4)
□ Page-template (3)
□ Author (2)
□ Landing page (2)
□ RSS page (1)
□ Site (1)
□ Template-folder (1)

Last Modified

□ < 1 week (4)
□ < 1 day (1)
□ < 1 hour (1)

□	Superhero /superhero	Online	2016-02-18
□	RSS rss	Modified	2016-04-13
□	Search search	Modified	2016-04-08
□	Profile profile	Online	2015-10-30
□	Posts posts	Online	2015-02-27
□	Categories categories	Offline	2015-02-27
□	Authors authors	Offline	2015-02-27
□	Templates _templates	Online	2015-02-27

Search results are instantly updated in the tree grid as search parameters are changed. Children of matching content will also exist in the tree grid but won't be visible unless the parent is expanded with the triangle.

The number of hits will appear below the search field, along with a **Clear** button that will remove all search parameters.

Each content type that has at least one matching content will appear in the Content Types list with the number of matches in parenthesis (). Checking the box next to a content type will remove the results that do not match that type. Multiple content types can be "checked".

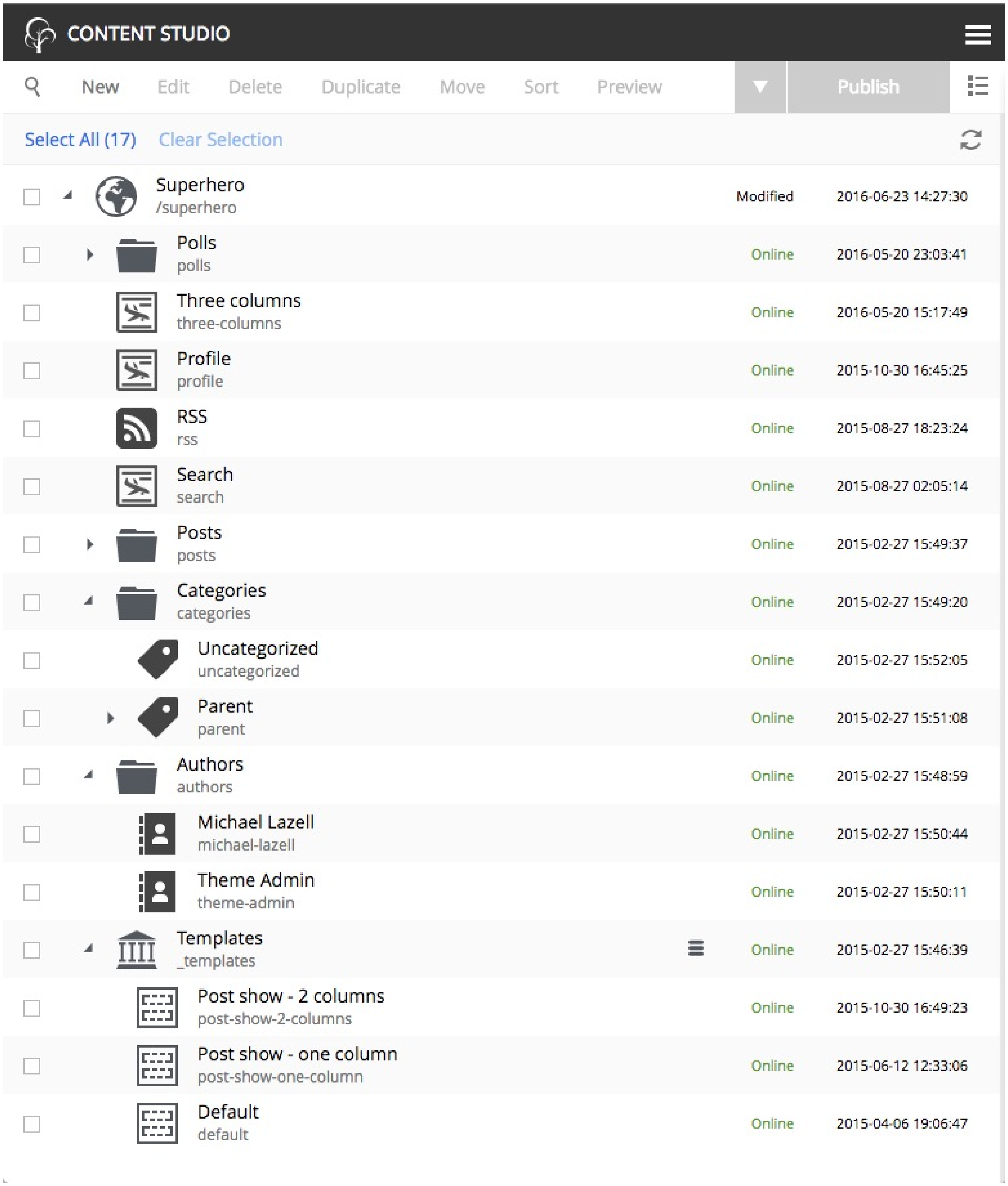
The Last Modified list works in a similar manner. It is used to limit results to content that has been modified in the last month, week, day or hour. This list may not appear if other search parameters have limited results to content that was modified more than a month ago.

Tip: Search results remain in the tree grid when the search panel is closed. Don't forget to clear the search when finished with it.

Content tree grid

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The content tree lists all the content that the logged in user has *read access* to. Content can be organized and stored in a hierarchy with child content collapsed under parent content. Parent content will have a gray triangle to the left of the content icon that can be clicked to expand or collapse the view of its children. The content tree can also be navigated with the keyboard arrow keys. The top of the content tree has buttons to select all displayed content, clear selected content and refresh the tree.



□	Superhero /superhero	Modified	2016-06-23 14:27:30
□	Polls polls	Online	2016-05-20 23:03:41
□	Three columns three-columns	Online	2016-05-20 15:17:49
□	Profile profile	Online	2015-10-30 16:45:25
□	RSS rss	Online	2015-08-27 18:23:24
□	Search search	Online	2015-08-27 02:05:14
□	Posts posts	Online	2015-02-27 15:49:37
□	Categories categories	Online	2015-02-27 15:49:20
□	Uncategorized uncategorized	Online	2015-02-27 15:52:05
□	Parent parent	Online	2015-02-27 15:51:08
□	Authors authors	Online	2015-02-27 15:48:59
□	Michael Lazell michael-lazell	Online	2015-02-27 15:50:44
□	Theme Admin theme-admin	Online	2015-02-27 15:50:11
□	Templates _templates	Online	2015-02-27 15:46:39
□	Post show - 2 columns post-show-2-columns	Online	2015-10-30 16:49:23
□	Post show - one column post-show-one-column	Online	2015-06-12 12:33:06
□	Default default	Online	2015-04-06 19:06:47

Information about the content is displayed in three columns. The first column has an icon based on the type of content. A gray triangle will appear to the left of the icon if the content has children content. The first column also has a display name and under this is the path name. The URL path to any content can be determined by following the tree grid path names from parent to child. The second column shows the status of the content which will be Offline, Online or Modified. The third column shows the date that content was last modified.

Checkboxes to the left of each content in the tree can be used to select more than one content. Group actions available in the menu bar will be performed on all the selected content. Some actions can only be performed on one content at a time.

Preview panel

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

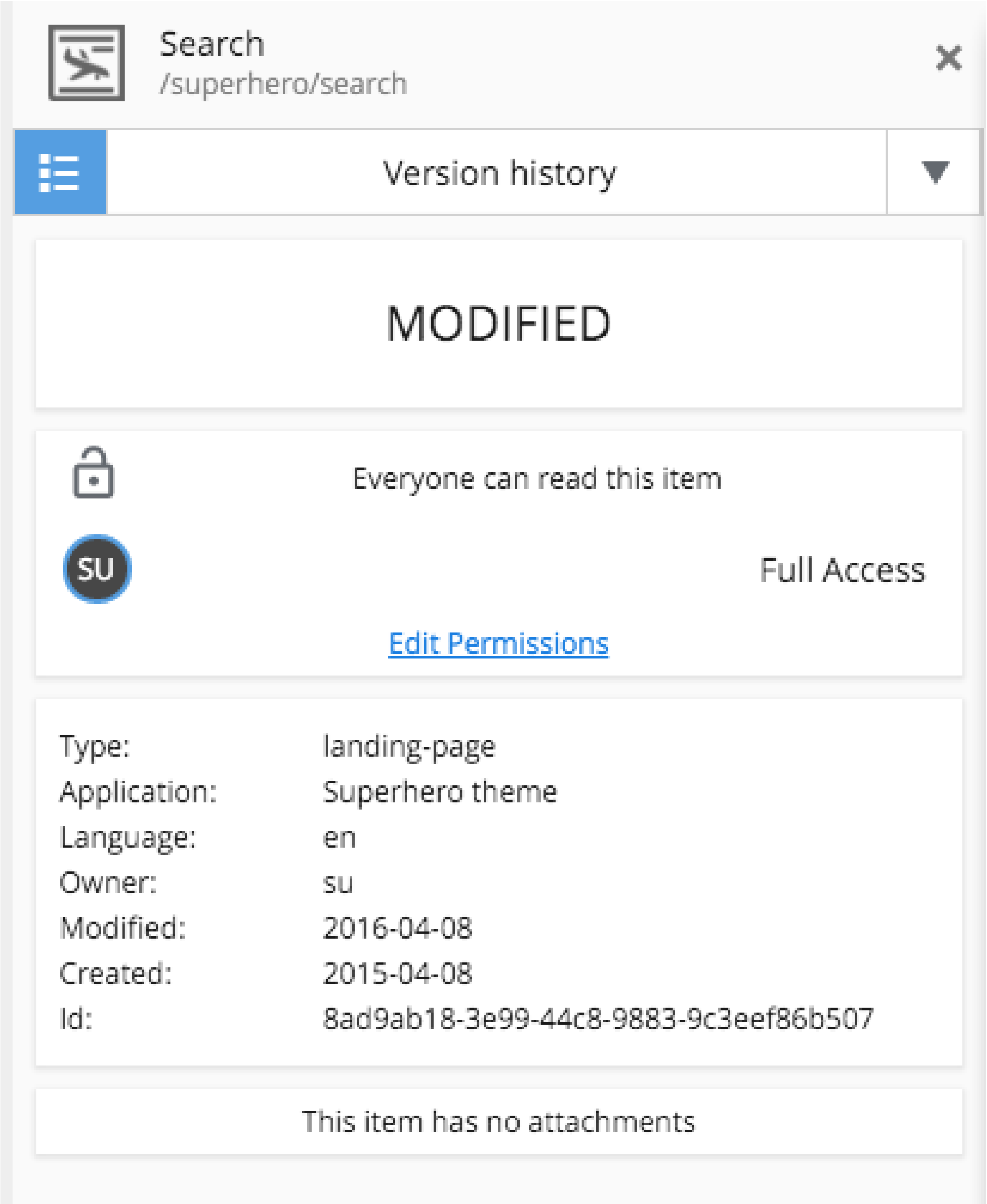
A preview of the selected content appears in the middle of the page if a site **page template** supports the type of content selected. This is a working preview so links will take you to other pages and highlight the new page content in the content tree grid. The **Preview** button in the toolbar will open the page in a new browser tab. When more than one content is selected in the content tree grid, the preview panel is replaced with a list of the selected content.

Note: Only the **Draft** version of content is visible in the preview panel.

Detail panel

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

This panel shows basic information about whichever content is selected in the tree grid. The content type icon is displayed at the top along with the Display Name of the content and the content path.



MODIFIED

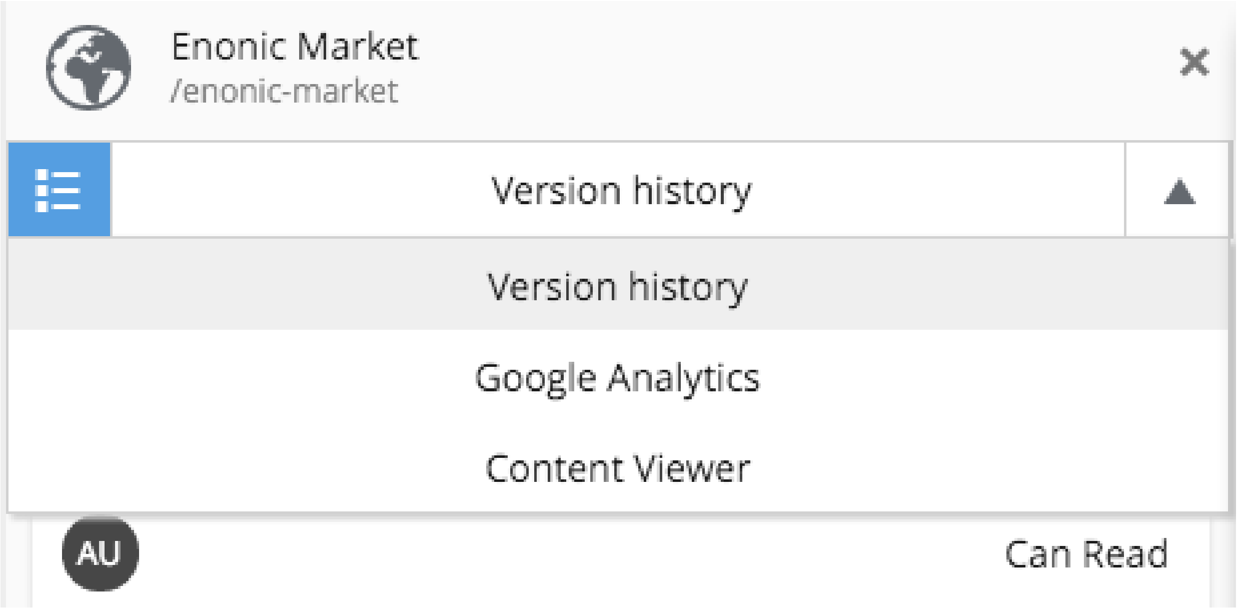
Everyone can read this item

Full Access

Type:	landing-page
Application:	Superhero theme
Language:	en
Owner:	su
Modified:	2016-04-08
Created:	2015-04-08
Id:	8ad9ab18-3e99-44c8-9883-9c3eef86b507

This item has no attachments

Next is the **widget selector** control with the installed widgets listed in a dropdown. The **Version history** widget is built-in and others can be added with the Applications admin tool. When a widget is selected, everything below it in the details panel is replaced with whatever the widget is designed to display. To the left of the widget selector is a button ☰ that restores the detail panel.



Next in the detail panel is the content status (Online, Offline, or Modified) followed by the permissions section. The permissions section has three parts. First it shows an unlock icon 🔓 if the content has the role **Everyone**, which means the content is viewable without authentication. Second, it lists the users that have access to the selected content at each access level, for example, “Can Read” or “Full Access”. Third, it has a link to edit permissions if the logged in user has the **Content Manager Administrator** role or other applied permissions to edit the content.

The next part of the detail panel shows basic information about the selected content, including type, application, language, owner, timestamps and ID. Finally, if the content has any file attachments, they are listed here.

Widgets

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Widgets are apps that add functionality to the Content Studio. As of version 6.8, widgets are limited to the *Detail panel* where they display custom information about whichever content is selected in the tree view. Installed widgets are listed in a dropdown selector at the top of the panel.

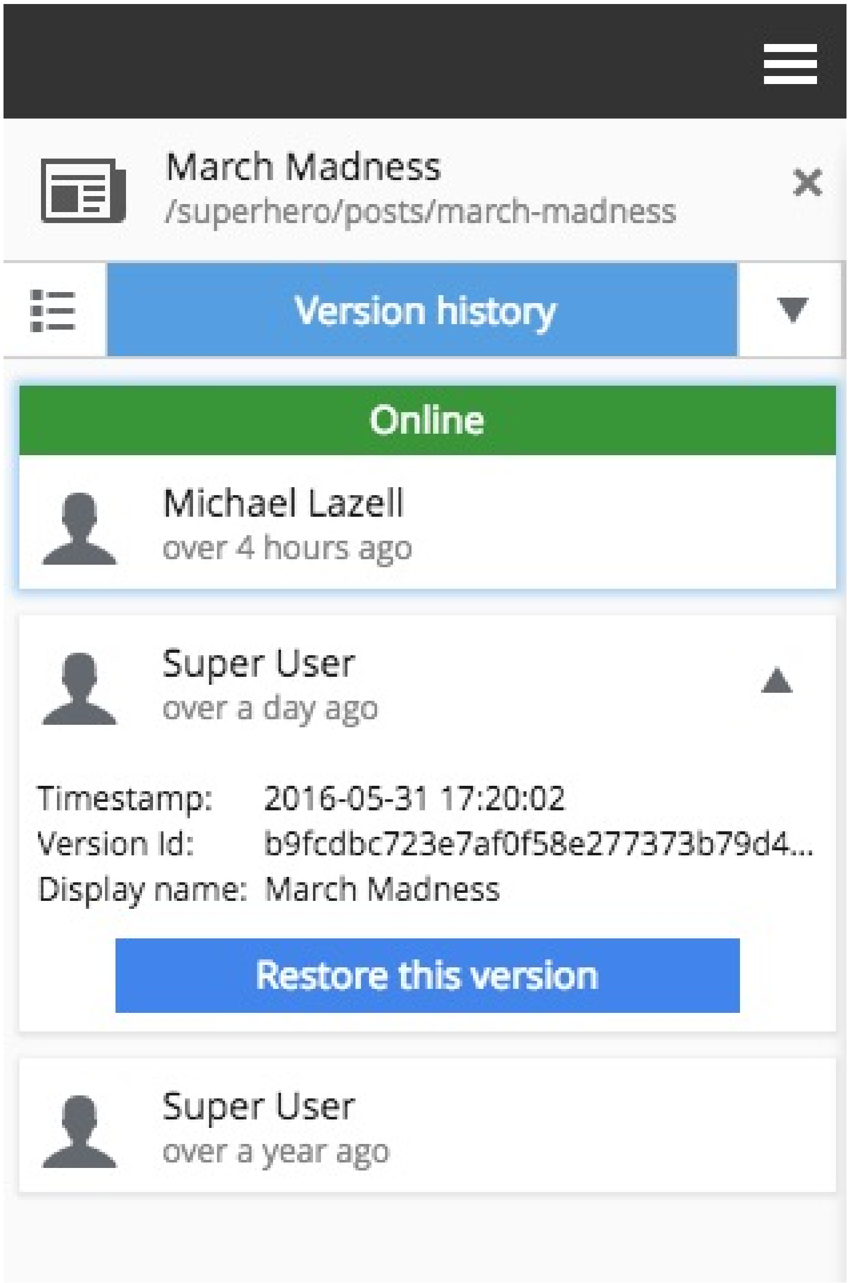
There are two built-in widgets.

Version history

Every time a content is saved with changes, a new version is created. The **Version history** widget displays a list of each version for a selected content. This widget also allows the user to switch between versions of a content.

Each version item in the list displays the name of the user who made the changes and the length of time since the change was made. If the content has been published then the published version will be marked as “Online” with a green heading.

Selecting one of the version items will expand it to reveal the timestamp, version id, and the display name of the content.



Timestamp: 2016-05-31 17:20:02
Version Id: b9fcdbc723e7af0f58e277373b79d4...
Display name: March Madness

Active version and online version Every content has one **active version**. A published content will also have an online version. The active version is the one shown in the preview and the one that will be loaded into the content editor when the content is edited. When a content is published, the active version is the one that goes online so it will be both the active and online version.

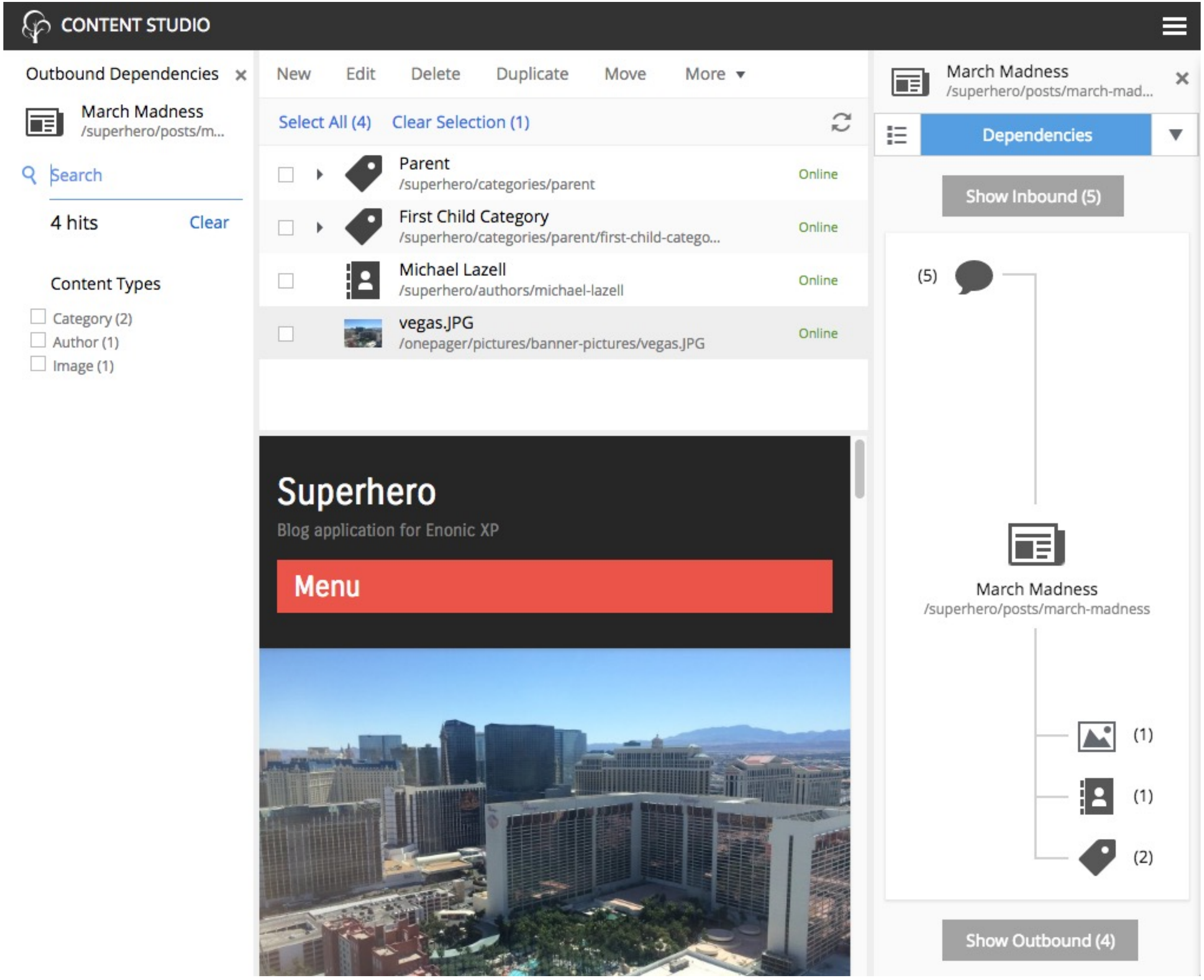
The active version can be changed by restoring a different version. If the active version is older than the published version, it will be listed as “Out-of-date”. The active version will be listed as “Modified” if it is newer than the online version.

Restoring a version Any version can be set as the **active version** by selecting it and clicking the button labeled “Restore this version”. Doing so will not change the published (online) version. This means that a previously published version of the content will remain online when a version is restored. A content must be published before the restored (active) version goes online.

Dependencies

Content can be configured to use content of other types. For example, an article content might use a ContentSelector for adding pre-defined categories. The dependencies widget makes it easy to find all the content that uses, or is used by, the selected content.

Select the Dependencies widget in the details panel and then select a content in the tree grid. The icon, name, and path of the selected content will appear in the center of the details panel. An icon for each type of inbound dependency is listed at the top-left. Icons for each type of outbound dependency are listed at the bottom-right. Next to each content type icon is the number of dependent items of that type.



□	Parent /superhero/categories/parent	Online
□	First Child Category /superhero/categories/parent/first-child-catego...	Online
□	Michael Lazell /superhero/authors/michael-lazell	Online
□	vegas.JPG /onepager/pictures/banner-pictures/vegas.JPG	Online

Inbound dependencies are other content items that use the selected content. **Outbound** dependencies are the content items that are used by the selected content. For example, the image above shows a Post content named March Madness. This Post has five inbound Comment items that reference the March Madness post. The Post itself references one image content, one Author content, and two Category contents. Therefore, the image, Author and Categories are outbound.

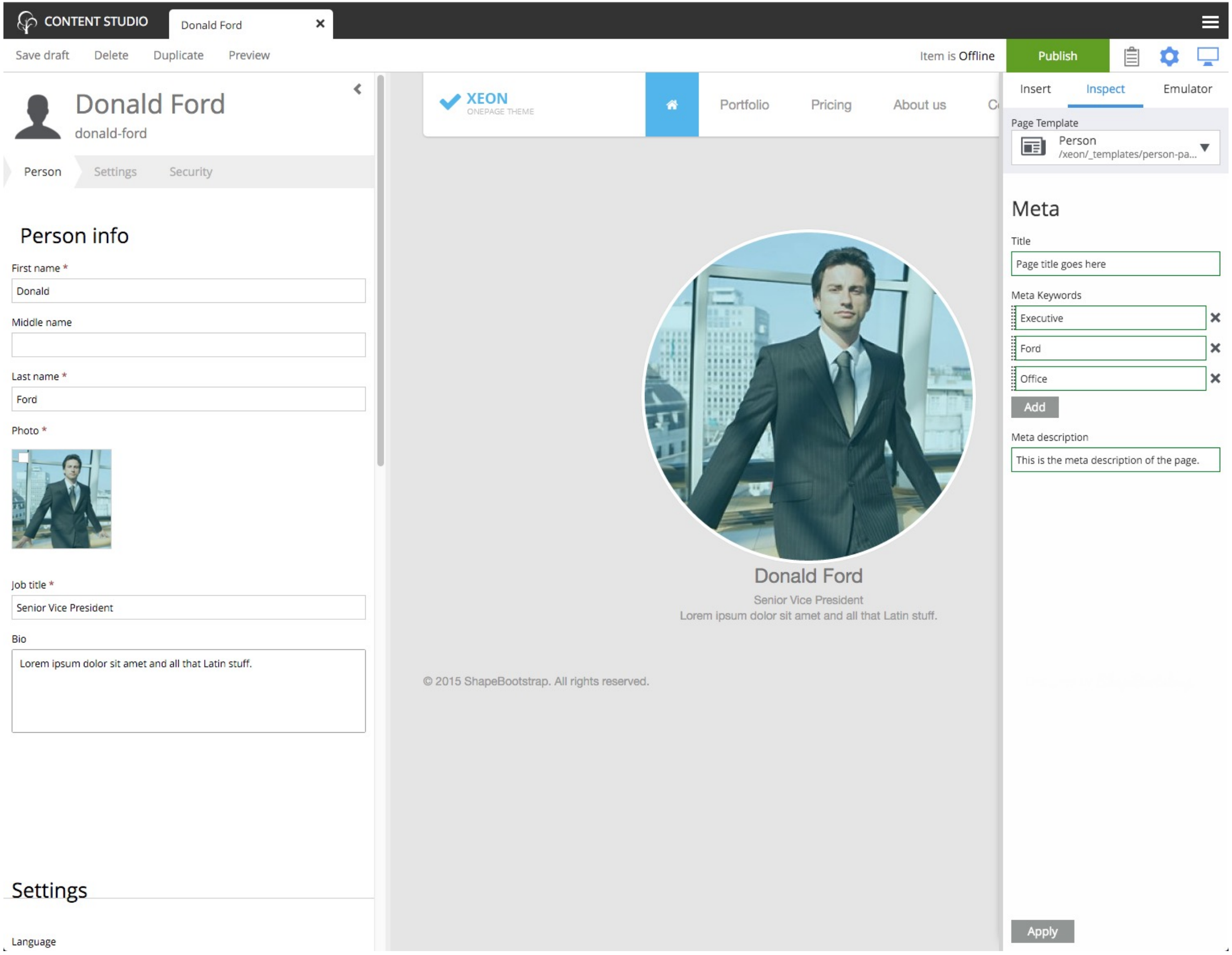
Inbound and outbound content can be viewed by clicking the respective buttons at the top and bottom of the dependencies widget. This will open the search panel on the left and filter the tree grid to show only the selected dependency type.

Edit View

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The **Edit view** exits as an internal tab within the Content Studio page. It appears when a content is created or edited. Multiple content tabs can be open at the same time. Users can switch between tabs by clicking them. The *Browse View* can be reached by clicking the words "Content Studio" at the top-left of the page and the content tabs will remain where they are. The tab will have a red circle with an exclamation mark if the content is not valid due to missing required fields. Invalid content can be saved in the **draft** branch but it cannot be published.

The edit view has several parts. On the left is the content editor panel. The center is where the page editor appears for content that has a page template. The inspection panel is on the right.



Donald Ford
donald-ford

Person info

First name *
Donald

Middle name

Last name *
Ford

Photo *

Job title *
Senior Vice President

Bio
Lorem ipsum dolor sit amet and all that Latin stuff.

Settings

Language

Donald Ford
Senior Vice President
Lorem ipsum dolor sit amet and all that Latin stuff.

© 2015 ShapeBootstrap. All rights reserved.

Page Template
Person
/xeon/_templates/person-pa...

Meta

Title
Page title goes here

Meta Keywords
Executive
Ford
Office

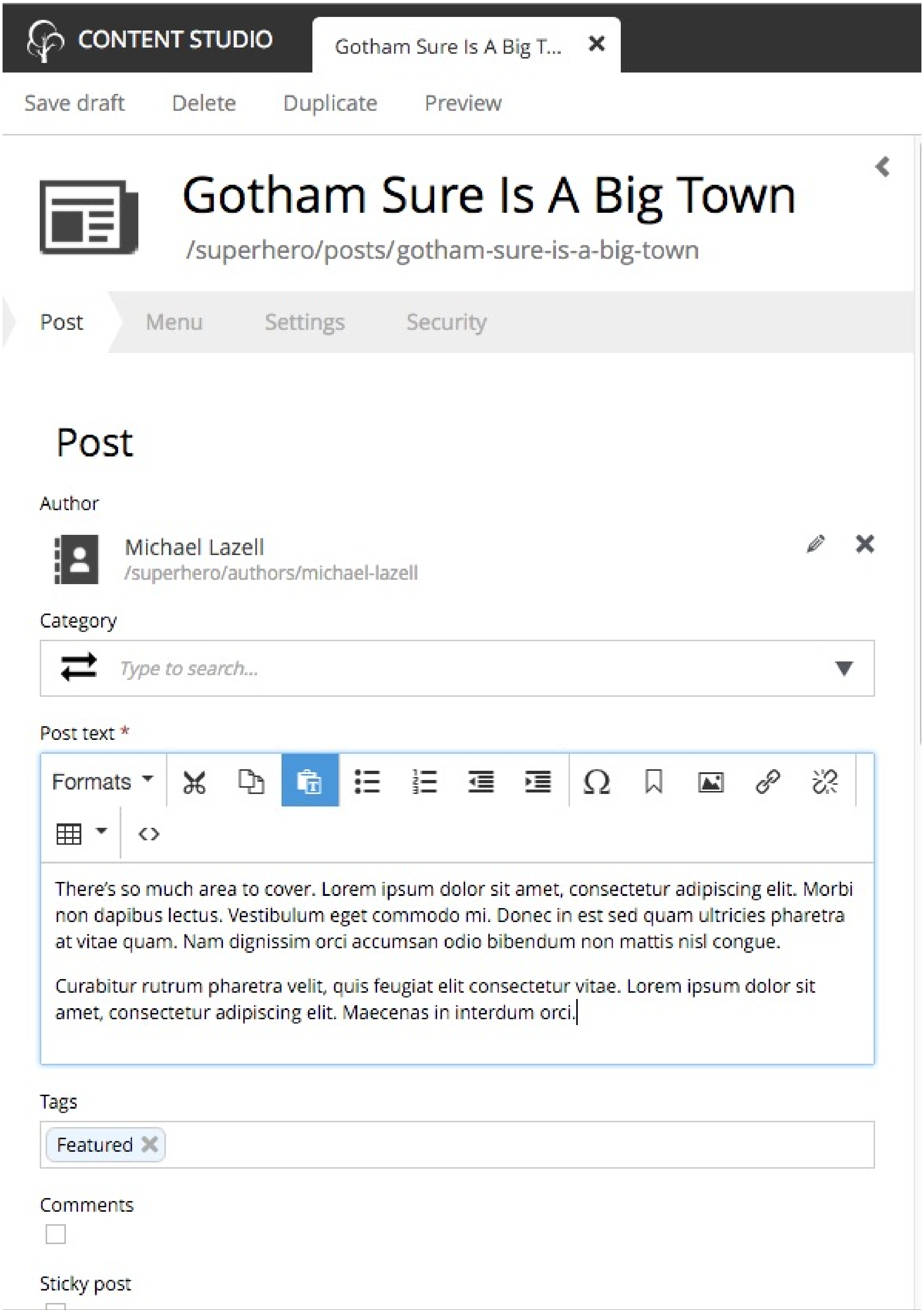
Meta description
This is the meta description of the page.

Content Editor panel

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

This panel appears on the left side of the page when content is created or edited. It is where the form appears in which content data is entered. The structure and fields in the form are determined by the **content type** which is defined in the application code. The content editor panel can be collapsed to give the page editor more room by clicking the arrow icon on the top right of the panel.

The top of the content editor panel has the content type icon, the Display Name, and the content path name. The default icon can be replaced with an image file by clicking on it. As the Display Name is entered, the path name will automatically be filled in with a URL-friendly version. The path will not update automatically once the content has been published. This is to prevent accidental breaking of external hard-coded links. The path name can always be changed manually. It is also possible for the Display Name to be generated automatically while other fields in the form are filled in if the content type was set up that way in the application code.



Gotham Sure Is A Big Town

/superhero/posts/gotham-sure-is-a-big-town

Post

Author

Michael Lazell
/superhero/authors/michael-lazell

Category

Type to search...

Post text *

There’s so much area to cover. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi non dapibus lectus. Vestibulum eget commodo mi. Donec in est sed quam ultricies pharetra at vitae quam. Nam dignissim orci accumsan odio bibendum non mattis nisl congue.

Curabitur rutrum pharetra velit, quis feugiat elit consectetur vitae. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas in interdum orci.

Tags

Featured

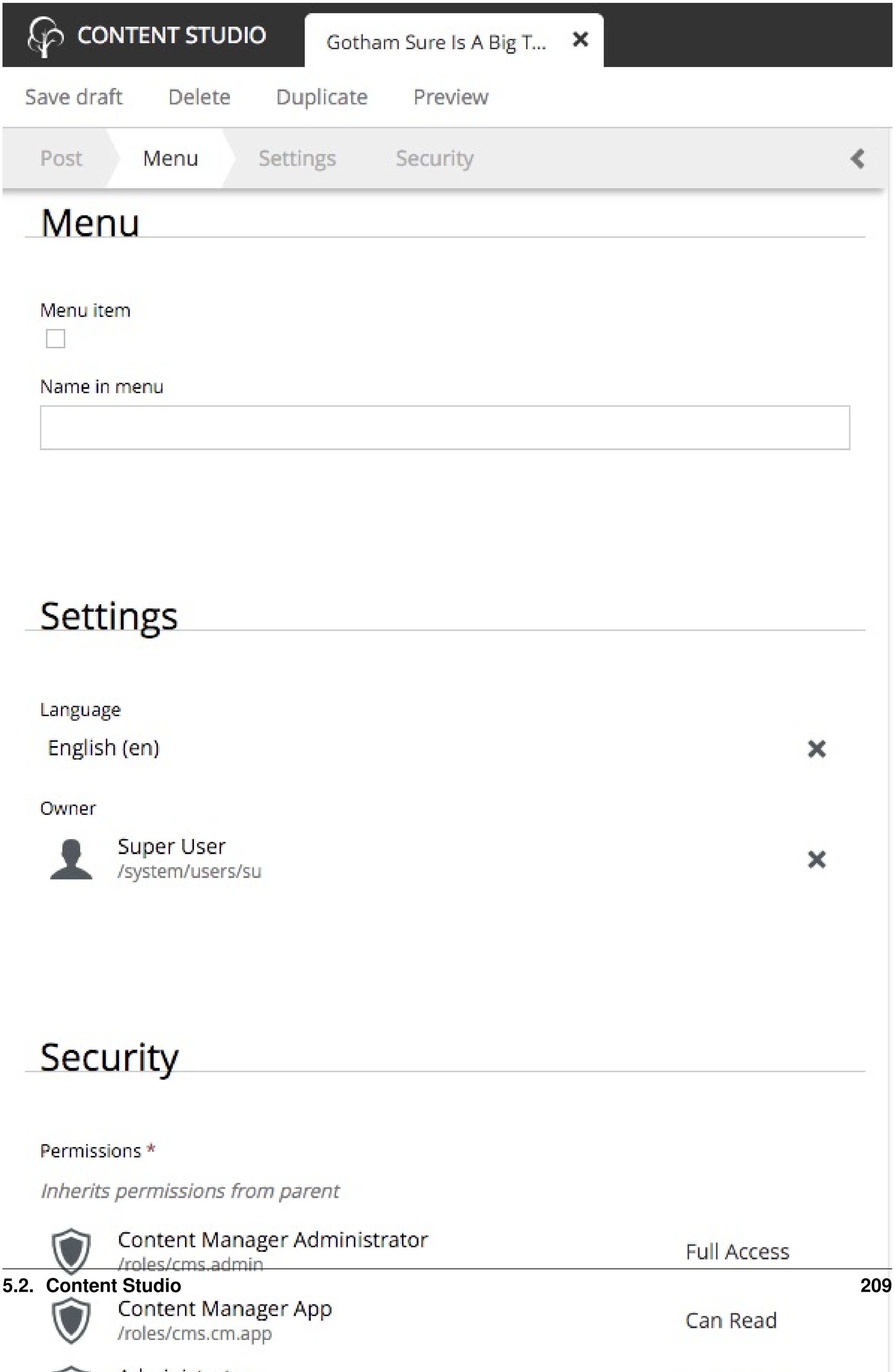
Comments

□

Sticky post

□

Next is a ribbon with buttons that correspond to different sections of the content form. All content will have buttons for the content type name, **Settings**, and **Security**. Other items may be in the ribbon if x-data was added to the content type in the application code. Clicking on an item in the ribbon will scroll the content editor to the corresponding section of the form.



Menu

Menu item

□

Name in menu

Settings

Language

English (en) ✕

Owner

Super User
/system/users/su ✕

Security

Permissions *

Inherits permissions from parent

Content Manager Administrator /roles/cms.admin	Full Access
Content Manager App /roles/cms.cm.app	Can Read

The content data section is next and the form fields will depend on how the content type was set up.

The **Settings** area is where the content's Language and Owner are set. The **Language** will be inherited from the parent content if it was set there. The Owner will be the logged in user who created the content, but it could be changed if the current user has the right permissions.

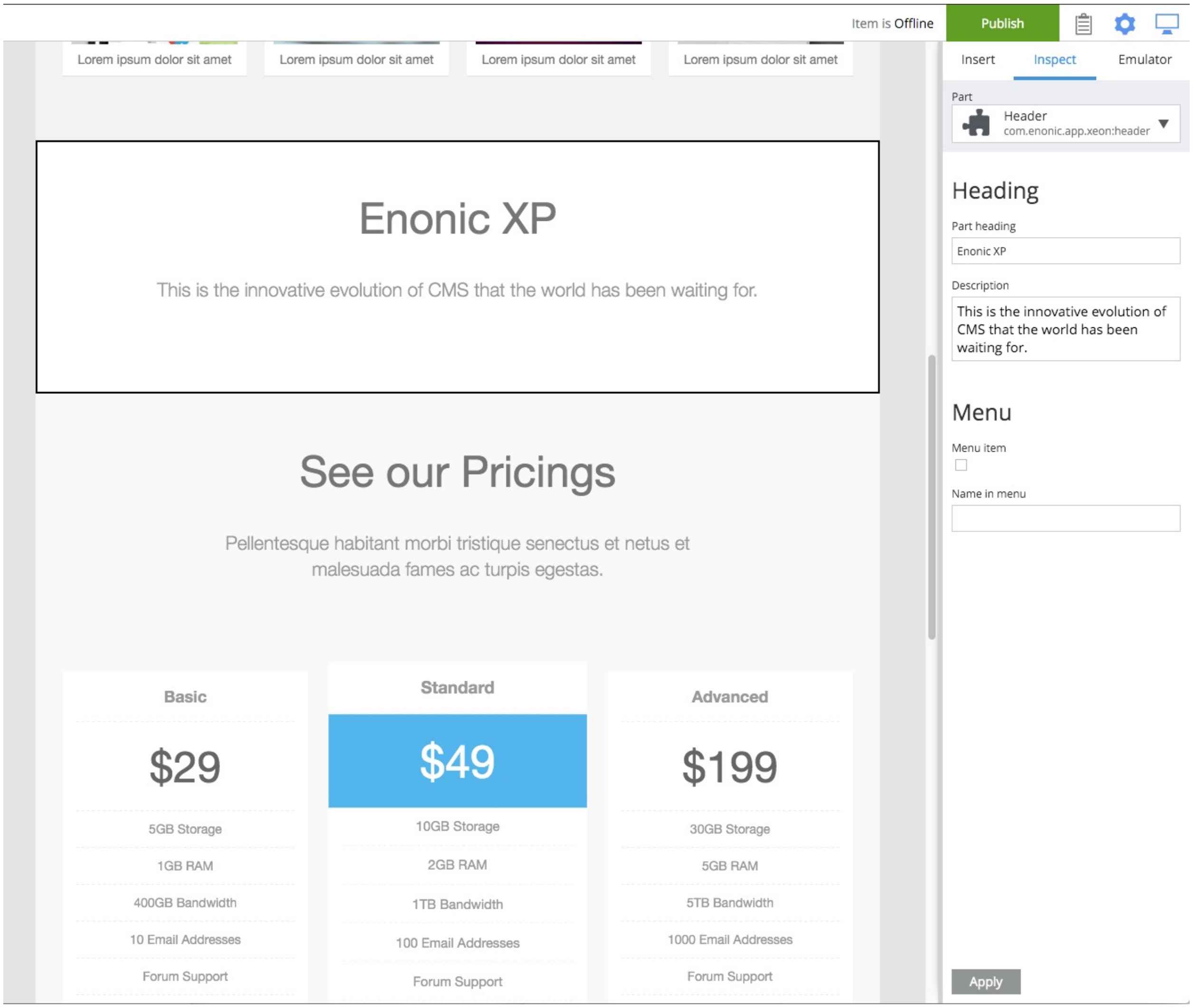
The **Security** section is where the content's permissions are set. Content will inherit the permissions of the parent content when created. Users, Groups and Roles are principals that can be added to the content. Clicking on any item here will expand it and show what permissions the principal has. Read more about *Content Security*.

Page Editor panel

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The page editor is a modifiable preview of the content page that will open automatically when a content that has a supporting page template is created or edited. It is used in conjunction with the *Inspection Panel* to add, remove, and move components around the page with drag and drop. The page editor can be closed and reopened with the icon on the far right of the toolbar that looks like a computer monitor.

A component can be selected in the page editor by clicking on it. This will highlight the selected component and show its configuration in the inspect panel. Right-clicking a component will open a context menu with various actions that can be performed on the component. These options may include **Select parent**, **Insert** another component, **Inspect** its configuration, **Reset** the component's configuration, **Remove** the component, **Duplicate** it, and **Create fragment** from the component.



Part: Header com.enonic.app.xeon:header

Heading

Part heading: Enonic XP

Description: This is the innovative evolution of CMS that the world has been waiting for.

Menu

Menu item: □

Name in menu:

The page editor will be closed when the type of content being edited does not have a supporting page template. However, it can be opened and a dropdown selector will appear which can be used to add a **page component** to render a page. Read more about this in the *Page Templates* section.

Inspection Panel

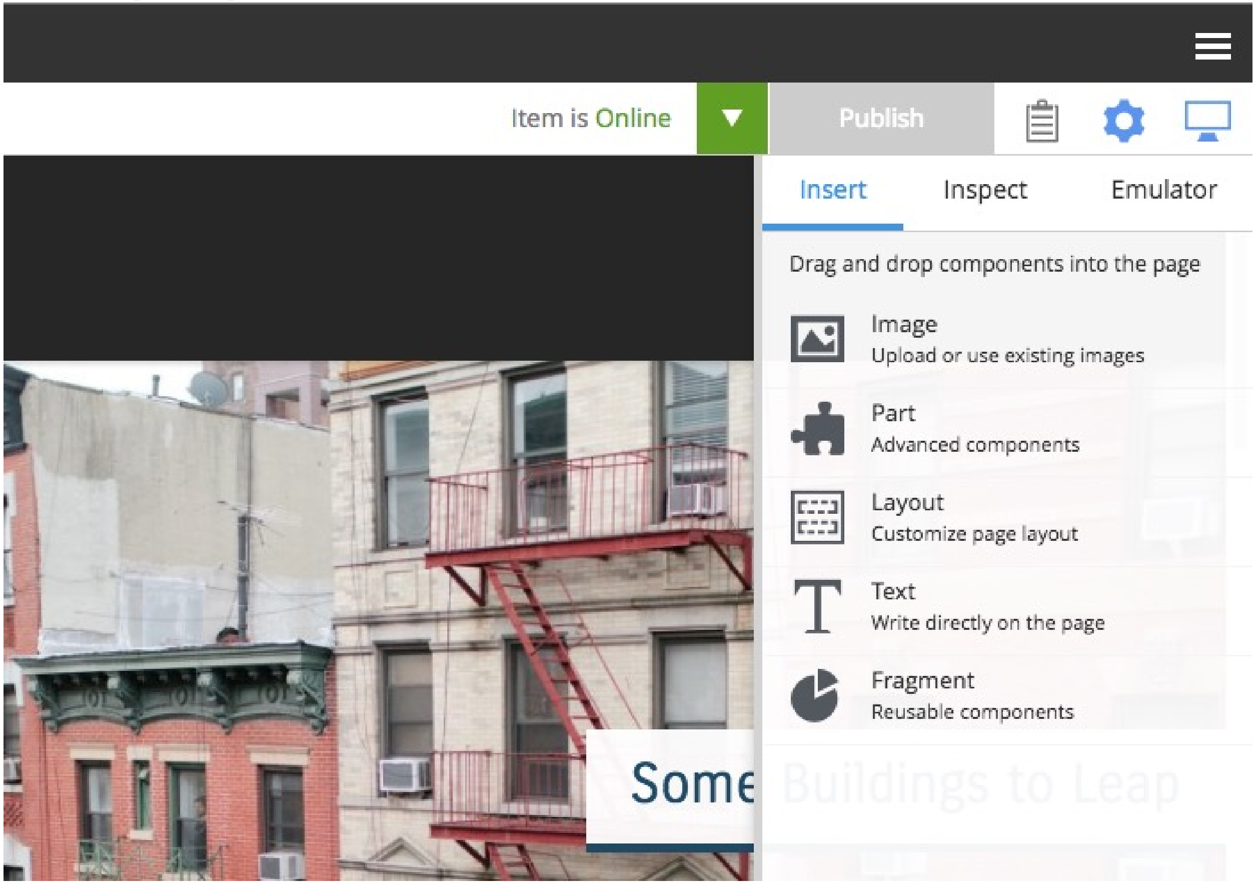
Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

This panel is a multi-function tool that can be toggled open and closed with the cog icon at the right of the toolbar. It is used in conjunction with the **Page Editor** to add and configure components on the page and to emulate various device sizes. The inspection panel has three tabs: Insert, Inspect, and Emulator.

Insert tab

This tab has a list of *Component types* that can be added to the page in a two step process. The first step is to drag and drop a component type placeholder to a region in the page editor. The second step is to select the specific component from a dropdown selector in the placeholder.

For example, to add a part component called “Categories” to the page, simply click the part icon (puzzle piece) and drag it to a region in the page editor. A red circle appears when the component is dragged over an area where it cannot be dropped. A green checkbox appears when dragged over valid locations and a blue box shows where the component will land. Once the part placeholder is dropped, it will have a combo-box where the “Categories” part can be selected from the list. When a component placeholder is selected, the inspection panel will also show a combo-box with a list of available part components.



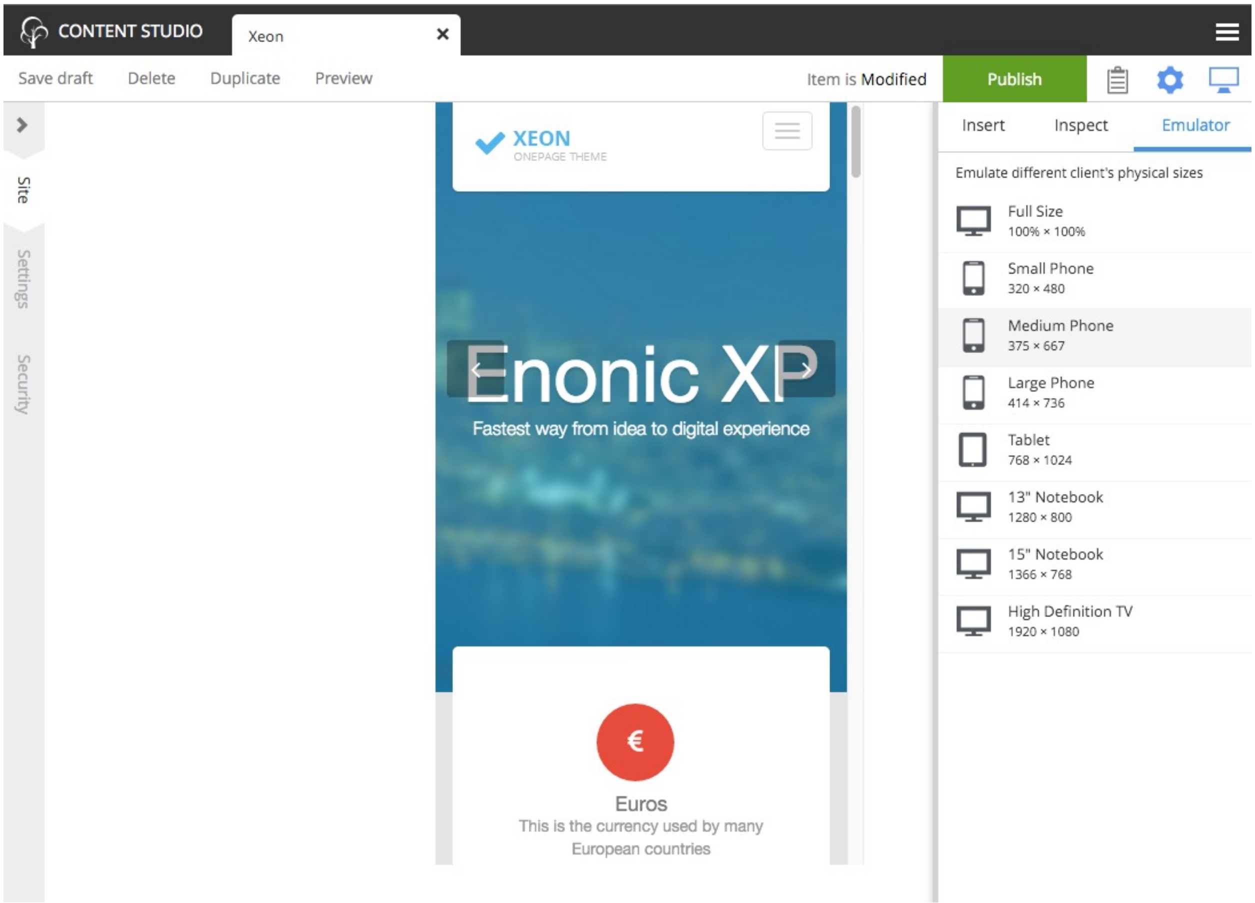
Inspect tab

This is where components are configured. The inspection panel displays form inputs matching the configuration of whichever component is selected in the page editor. Some components won’t have any configuration settings. The page will not update as configuration values change until the content is saved or when changes are applied with the button at the bottom of the panel.

The inspect tab is also used for changing the default page template of a selected content and choosing a page component to render content that does not have a supporting page template. Read more about this in *Page Templates*.

Emulator tab

This tab has buttons for emulating various sized devices. The page editor will shrink and expand to fit the selected device size.



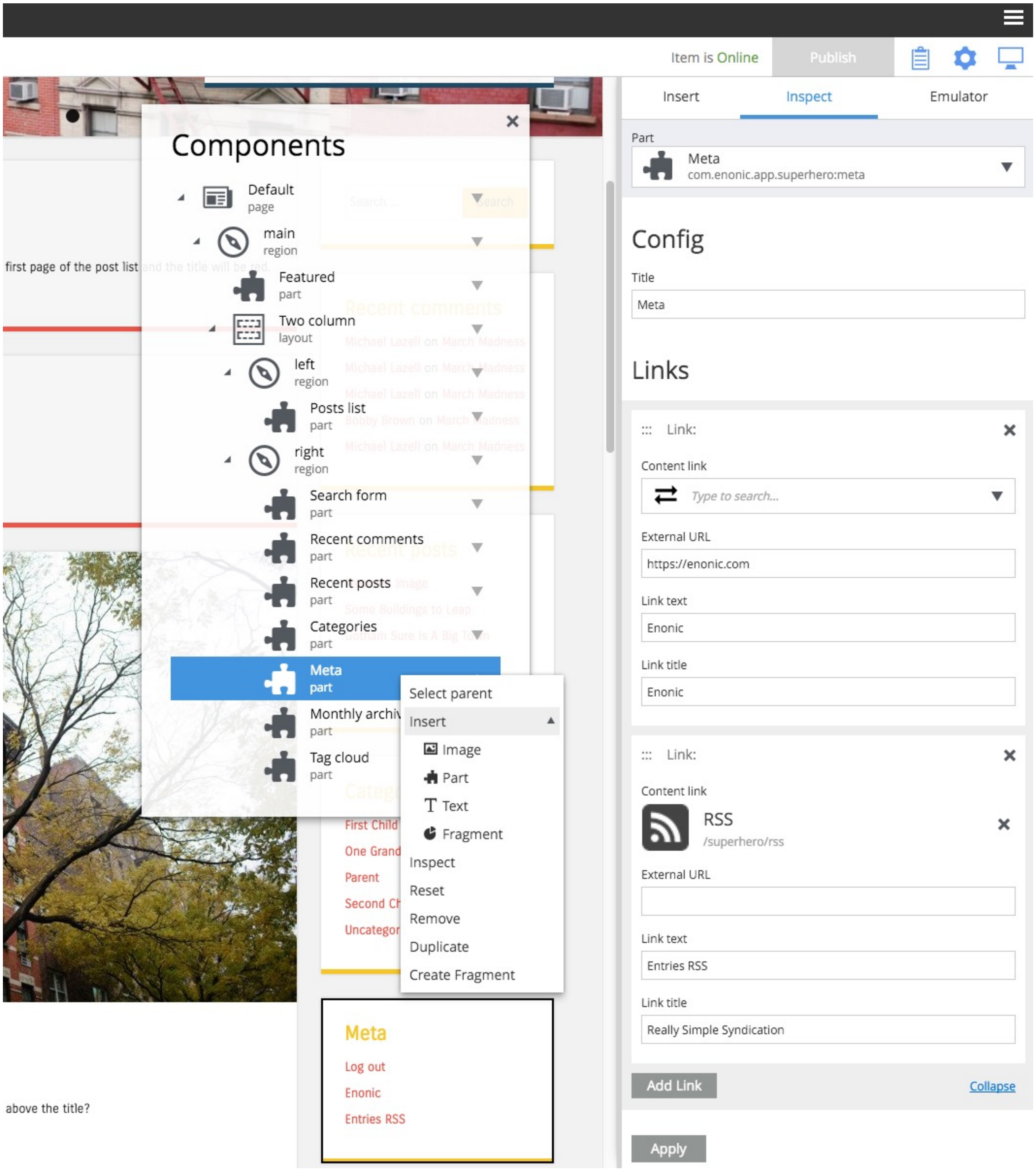
Component View

Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

This tool opens in a draggable internal window activated by the clipboard icon in the toolbar. It displays a hierarchical tree representation of all the components and regions on the page, including the page itself. This tool is extremely valuable in situations where an unconfigured component does not render or is very small and difficult to find in the page editor.

Selecting a component in the tree will highlight the component in the page editor and display its configuration in the inspection panel's **Inspect tab**. The triangle on the right of each component will open a menu of options that can be selected for the component. The menu options available depend on the component selected, but may include: **Insert** for inserting other components, **Reset** to reset the component's configuration, **Remove**, **Duplicate** and **Create fragment**.

In the image below, a part named **Meta** is selected in the Components view and the part is highlighted with a black border at the bottom of the page editor. The Meta part's configuration is visible in the inspect panel on the right.



Part
Meta
com.enonic.app.superhero:meta

Config

Title: Meta

Links

Link:
Content link: Type to search...
External URL: https://enonic.com
Link text: Enonic
Link title: Enonic

Link:
Content link: RSS /superhero/rss
External URL:
Link text: Entries RSS
Link title: Really Simple Syndication

Content actions

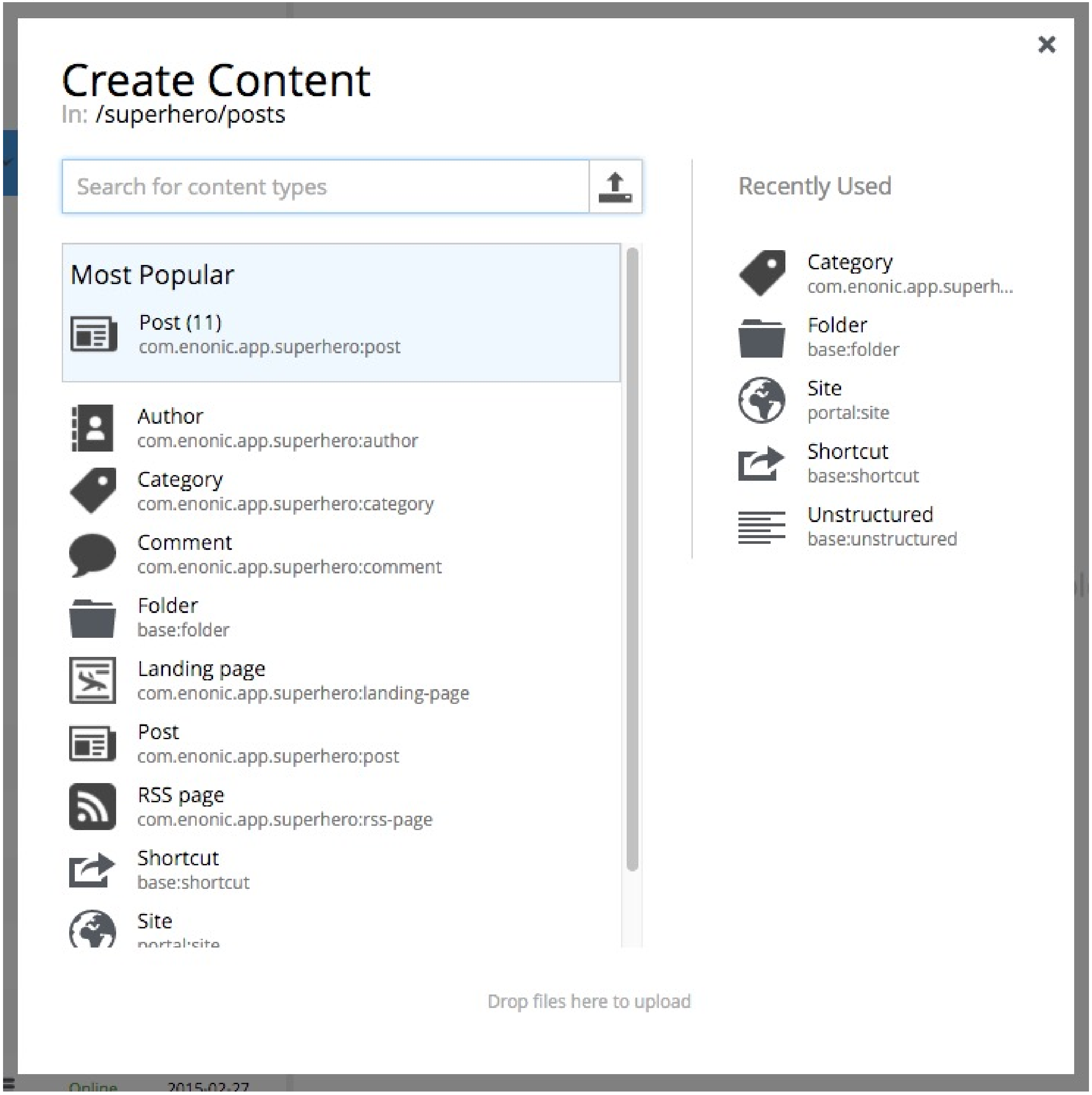
Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Content can be created, edited, deleted, duplicated, moved, sorted and published.

Create Content

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

All content created with the Content Studio begins with the **Create Content** dialogue. This dialogue has two functions. First it assists the user with finding the type of content to be created. Second, it opens the edit view in a new internal tab.



Create Content

In: /superhero/posts

Search for content types

Most Popular

Post (11)
com.enonic.app.superhero:post

Author
com.enonic.app.superhero:author

Category
com.enonic.app.superhero:category

Comment
com.enonic.app.superhero:comment

Folder
base:folder

Landing page
com.enonic.app.superhero:landing-page

Post
com.enonic.app.superhero:post

RSS page
com.enonic.app.superhero:rss-page

Shortcut
base:shortcut

Site
portal:site

Recently Used

Category
com.enonic.app.superh...

Folder
base:folder

Site
portal:site

Shortcut
base:shortcut

Unstructured
base:unstructured

Drop files here to upload

There are three ways to open the **Create Content** dialogue in the Content Studio. The first method is with the shortcut - **alt + n**. The other methods require selecting the **New** button in the toolbar or in the context menu that appears when right-clicking an existing content in the tree grid.

Content will be created as a child of whichever content was selected in the tree grid when the Create Content dialogue was opened. If no content was selected, the new content will be created at the root of the repository. Usually, only **Site** contents are created at the root.

Search field

The search field can be used to quickly find a content type.

File and image upload

To the right of the search field is an upload button represented by an icon with an up arrow. Clicking this icon will open the user's file browser so one or more files can be selected from the file system. The type of content created will depend on the type of file selected. Files can also be dragged and dropped onto the Create Content dialogue. A new edit view tab will open for each selected file so that meta-data can be entered.

Most Popular

If the parent content has one or more existing child items, the most used type of child content will appear with a blue background at the top of the content type list. The number of this type of child content is in parenthesis.

Content type list

All content types of the installed applications are listed on the left side of the Create Content dialogue. The list is instantly updated as characters are typed into the search field. Each item in the list has the content type icon, Display Name of the content type, and the full app name of the content type. It is possible for two applications to have a content type with the same Display Name, so the full app name would be helpful to tell them apart.

Recently Used

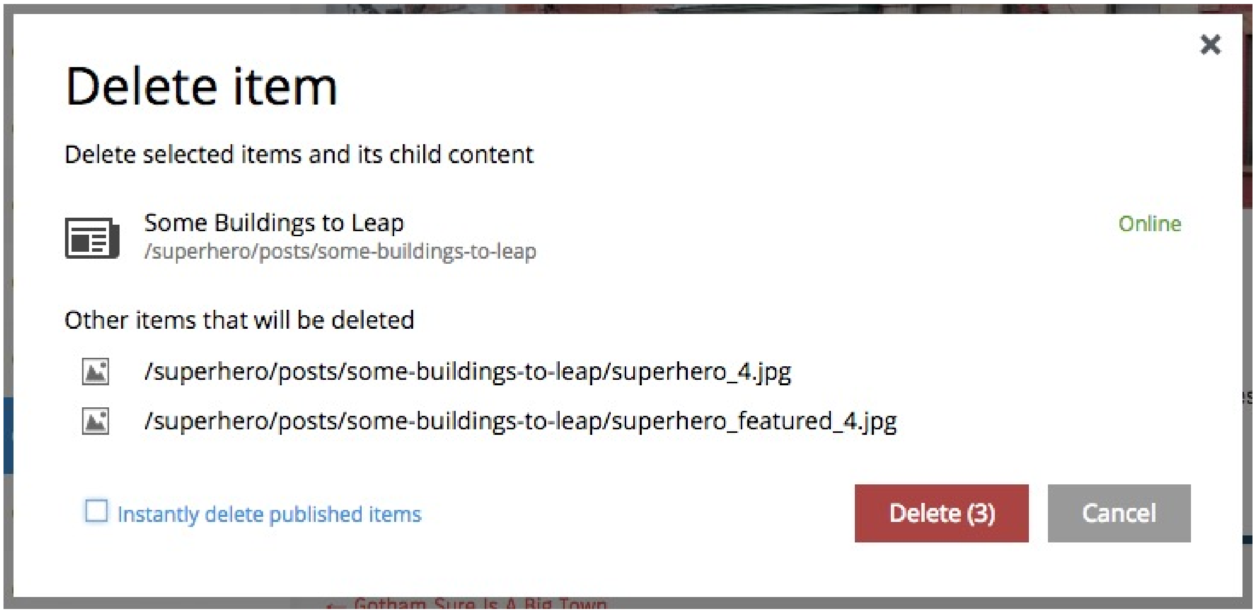
The right side of the dialogue will list the recently used types of content.

Delete

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Content is deleted through the **Delete item** dialogue. This dialogue can be accessed by the **Delete** button in the toolbar of both browse view and edit view. It can also be opened from the context menu by right-clicking a content in the tree grid. The selected content is listed along with its status: Online or Offline. If the content is a parent then the children content are listed and will also be deleted. The Delete button shows the total number of items that will be removed. More than one content can be selected for deletion from the tree grid in browse view.

Offline content is removed immediately. Content that is **online** can be removed immediately by checking the box labeled “Instantly delete published items”. If this box is not checked then the content will only be marked for deletion. Its status will change to **pending delete** and its name will have a line through it. Content that is pending delete will still be visible outside the Content Studio. Content that is pending delete can be removed and taken offline by publishing or unpublishing it, or deleting it again and checking the box to “Instantly delete published items”.



Delete item

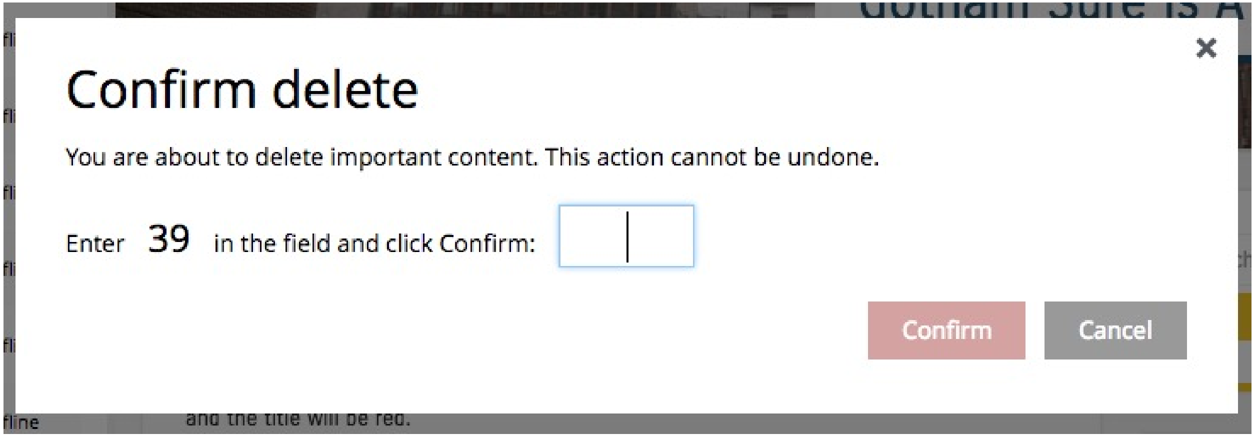
Delete selected items and its child content

Some Buildings to Leap
/superhero/posts/some-buildings-to-leap
Online

Other items that will be deleted

/superhero/posts/some-buildings-to-leap/superhero_4.jpg
/superhero/posts/some-buildings-to-leap/superhero_featured_4.jpg

An extra layer of protection kicks in before bulk items or a **site** content can be deleted. The "Confirm delete" dialogue appears to warn that this action cannot be undone. The number of content items to be deleted must be entered before the deletion will occur.



Confirm delete

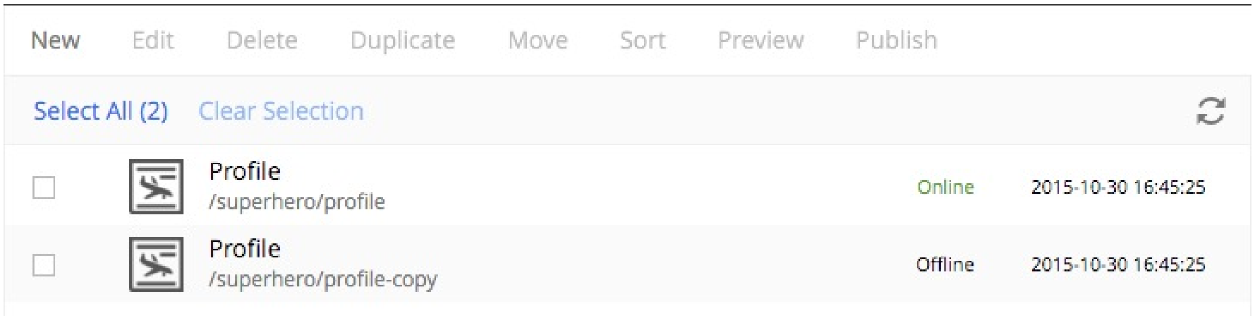
You are about to delete important content. This action cannot be undone.

Enter 39 in the field and click Confirm:

Duplicate

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

This action will make a copy of the selected content and add "-copy" to the duplicate's path name. The only other difference between the original and duplicate will be the content ID. Even the **Created** and **Modified** dates will be the same. This option is disabled when more than one content is selected in the tree grid.

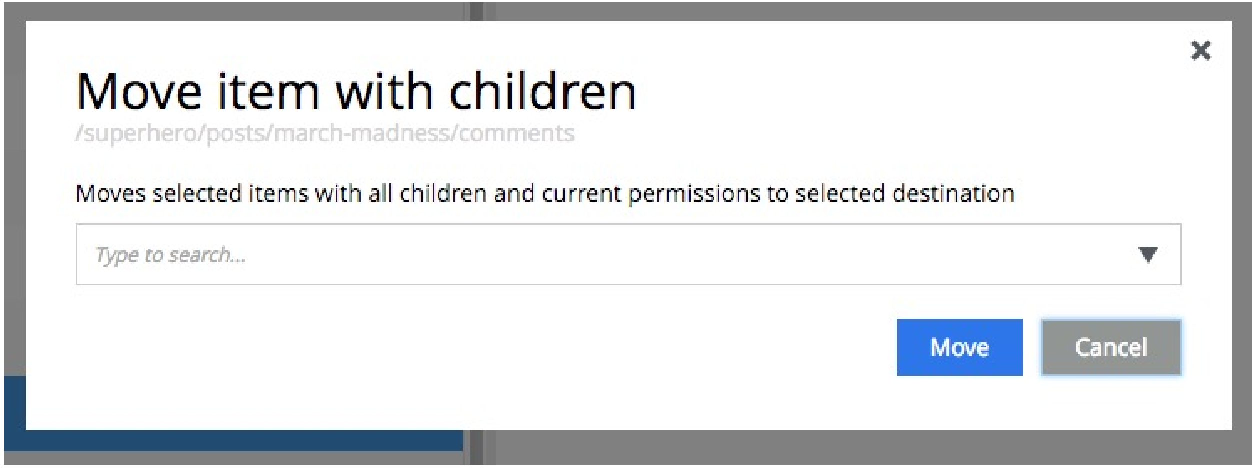


□	Profile /superhero/profile	Online	2015-10-30 16:45:25
□	Profile /superhero/profile-copy	Offline	2015-10-30 16:45:25

Move content

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The toolbar's "Move" button opens the **Move item with children** dialogue. This feature moves selected items with all children and current permissions to another place in the tree grid. Type to search, or use the dropdown arrow, to find the new parent content where the selected items will be moved to. If the "Move" button in the dialogue is clicked without choosing a parent content then the items will be moved to the root of the content tree. Content cannot be moved to a location where another content already exists with the same path name.


Move item with children

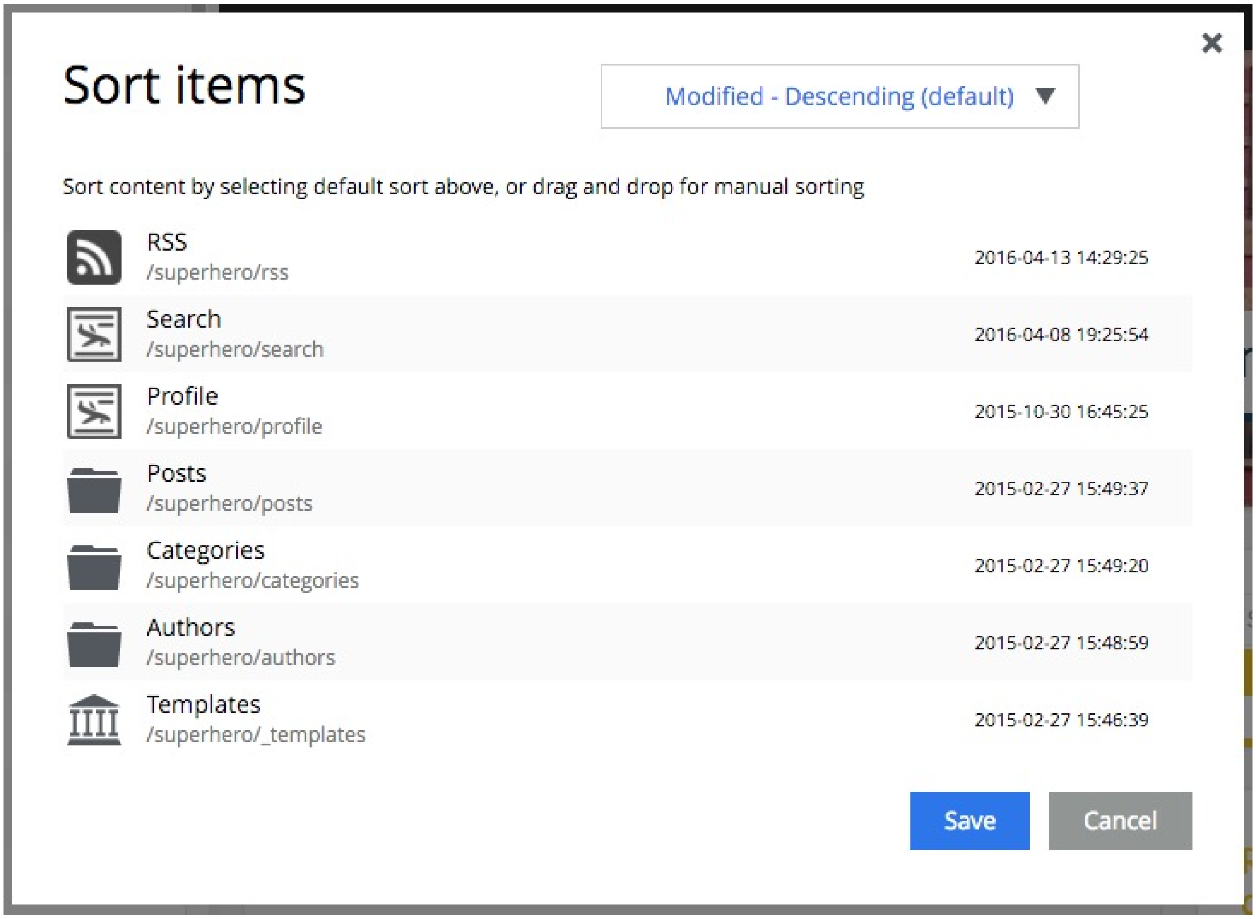
/superhero/posts/march-madness/comments

Moves selected items with all children and current permissions to selected destination

Sorting content

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Child items of any content can be sorted in a number of ways through the **Sort items** dialogue. It is accessed with the **Sort** button in the toolbar or context menu, or by clicking a "sort" icon that appears for previously sorted content.



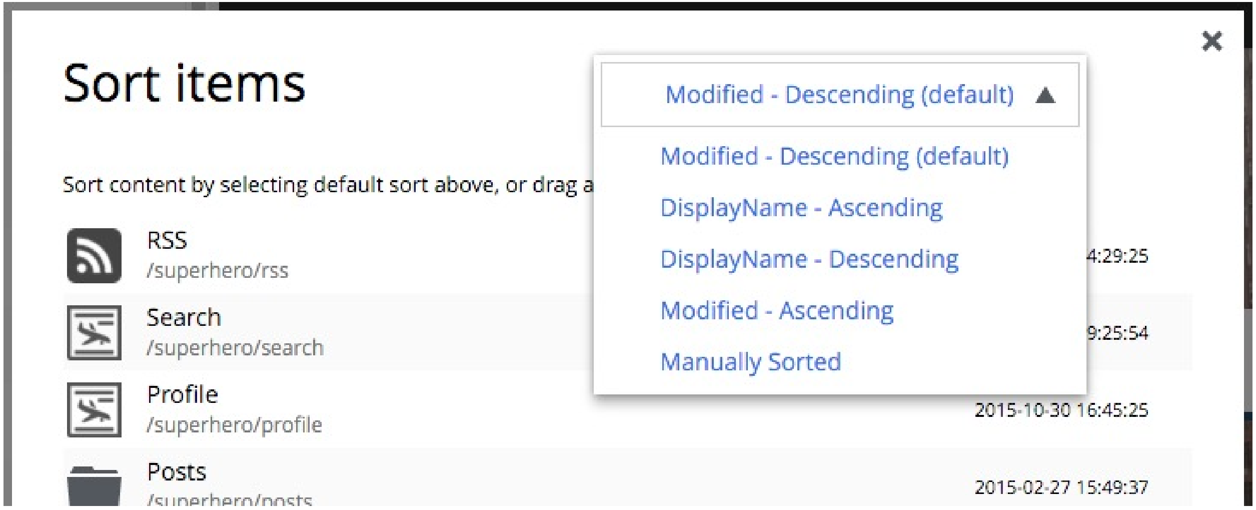
Sort items

Modified - Descending (default) ▼

Sort content by selecting default sort above, or drag and drop for manual sorting

RSS /superhero/rss	2016-04-13 14:29:25
Search /superhero/search	2016-04-08 19:25:54
Profile /superhero/profile	2015-10-30 16:45:25
Posts /superhero/posts	2015-02-27 15:49:37
Categories /superhero/categories	2015-02-27 15:49:20
Authors /superhero/authors	2015-02-27 15:48:59
Templates /superhero/_templates	2015-02-27 15:46:39

The default sorting is by the **modified** time in descending order. The other options are **modified** time ascending, **DisplayName** (ascending or descending), and **Manually Sorted**. To manually sort items, simply drag and drop them in the **Sort items** dialogue.



Sort items

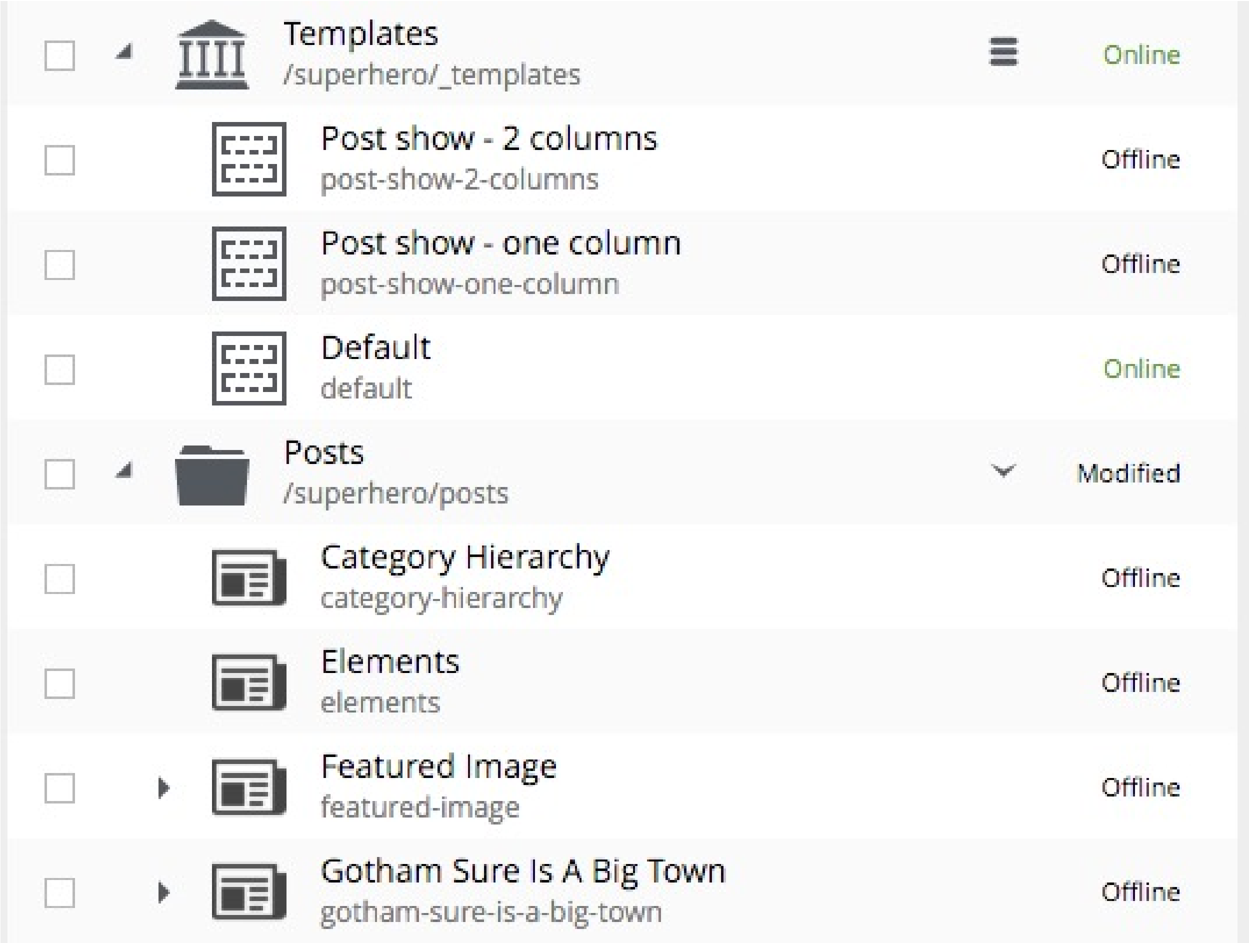
Modified - Descending (default) ▲

- Modified - Descending (default)
- DisplayName - Ascending
- DisplayName - Descending
- Modified - Ascending
- Manually Sorted

Sort content by selecting default sort above, or drag a

RSS /superhero/rss	4:29:25
Search /superhero/search	9:25:54
Profile /superhero/profile	2015-10-30 16:45:25
Posts /superhero/posts	2015-02-27 15:49:37

Once items are sorted, the parent content will have an arrow in the browse view tree grid pointed up or down to denote ascending or descending. If the content is manually sorted, an icon with three horizontal bars will appear. Clicking the bars or arrow icon will open the **Sort items** dialogue. In the image below, the Templates folder has manual sorting and the Posts folder is sorted by DisplayName descending.



Templates	/superhero/_templates	Online
Post show - 2 columns	post-show-2-columns	Offline
Post show - one column	post-show-one-column	Offline
Default	default	Online
Posts	/superhero/posts	Modified
Category Hierarchy	category-hierarchy	Offline
Elements	elements	Offline
Featured Image	featured-image	Offline
Gotham Sure Is A Big Town	gotham-sure-is-a-big-town	Offline

Publishing content

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Publishing is a simple but important concept for working with content in Enonic XP. The basic principal of publishing is that it makes content viewable to others outside of the Content Studio.

Draft and master branch

All content created with the Content Studio exists in the **draft** branch with the status **offline**. Content in the draft branch can be edited, changed, and previewed until it is ready to go online. Every time a content is saved with changes, a new version is created. (See *Version history*) When a content is published, the active version is copied from the **draft** branch to the **master** branch. Only content in the master branch can be accessed by others outside of the Content Studio, subject to the contents' security settings.

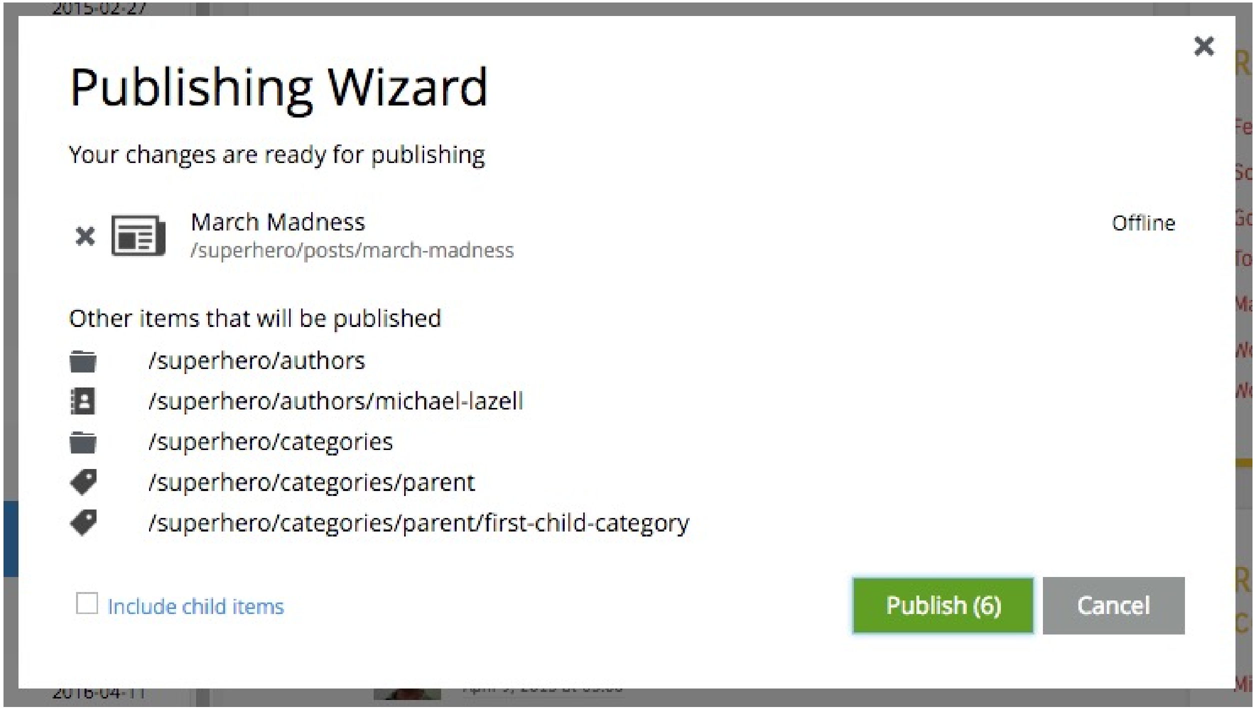
Content status

Published content will have the status **online** while content that has not yet been published will be **offline**. When changes to a published content are saved, the new version becomes the active version but the version that is online is not changed. The status of the new active version will be **modified** and this content will need to be published again before the changes will be visible outside of the Content Studio.

When a published content is “deleted”, the “Delete item” dialogue offers a checkbox to “Instantly delete published items”. If this box is not checked then the content’s status will be **pending delete** and it will still be visible outside of the Content Studio. Content that is pending delete must be published, unpublished, or “instantly deleted” before it is actually removed from the master branch.

Publishing wizard

Content is published through the **Publishing Wizard** dialogue. When a content is selected for publishing, its parents and all the related content will be published with it. For example, in the image below, a **Post** content named “March madness” was selected for publishing. This post has two related **Category** contents and a related **Author** content. Therefore, the categories and author will be published with the March Madness post and the parent folders of the categories and author will also be published. All items that will be published with the selected content are listed in the publishing wizard.



Publishing Wizard

Your changes are ready for publishing

March Madness
/superhero/posts/march-madness
Offline

Other items that will be published

/superhero/authors
/superhero/authors/michael-lazell
/superhero/categories
/superhero/categories/parent
/superhero/categories/parent/first-child-category

☐ Include child items

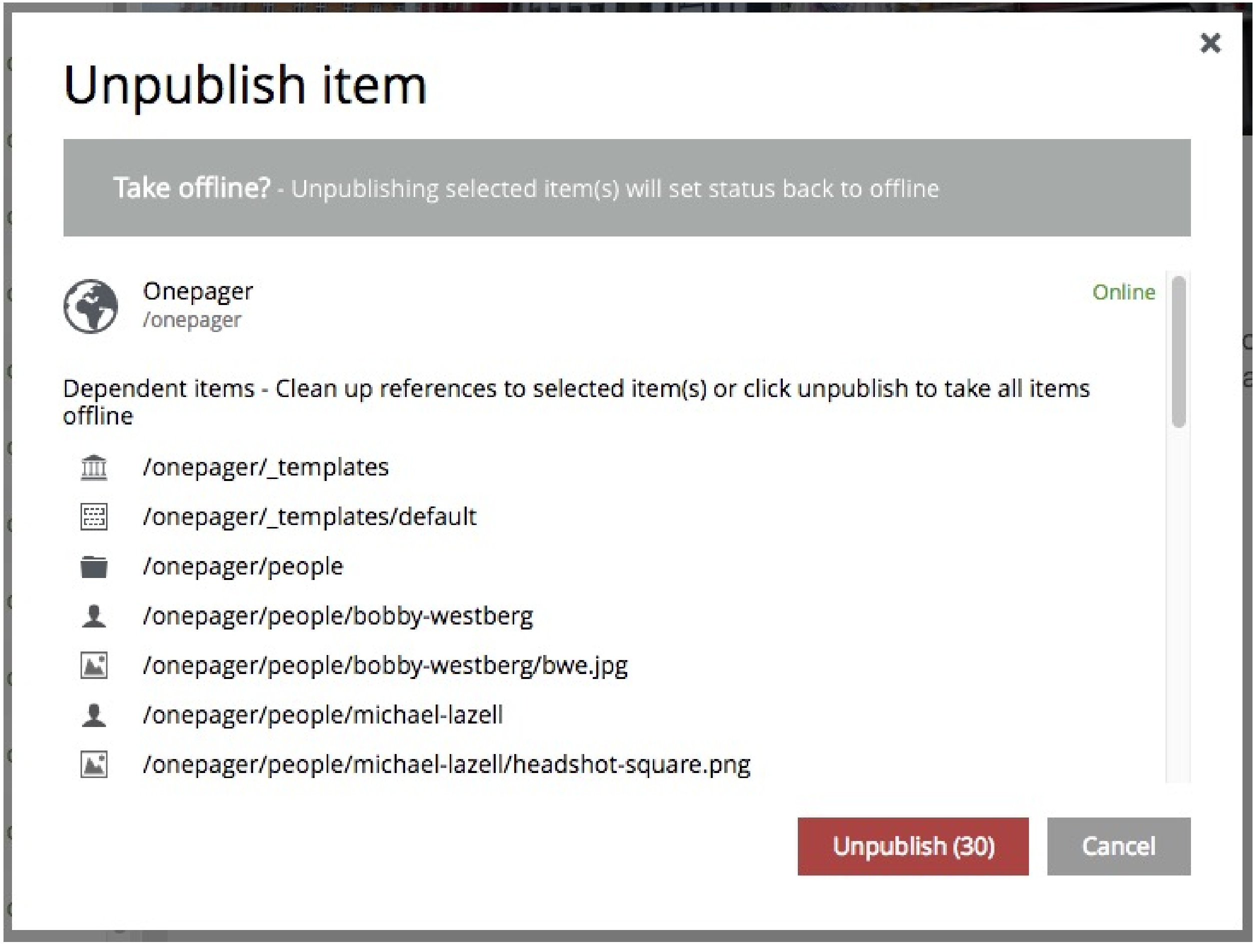
If the selected content has children then these items can be included by checking the box labeled “Include child items”. The total number of items that will be published is displayed on the **Publish** button. The green “Publish” button has a menu option for “Publish tree” which simply opens the dialogue with the “Include child items” box checked.

Unpublish content

Previously published content can be taken offline with the “Unpublish” feature. The “Unpublish” dialogue can be opened from the Publish menu in the toolbar or by right-clicking the content in the tree grid and selecting “Unpublish” from the context menu. All of the content’s children will be listed and unpublished along with the selected content.

The total number of content items that will be taken offline will appear in parenthesis in the red “Unpublish” button at the bottom of the dialogue.

Content that has the status **Pending delete** will be removed and taken offline when unpublished.



Unpublish item

Take offline? - Unpublishing selected item(s) will set status back to offline

Onepager
/onepager
Online

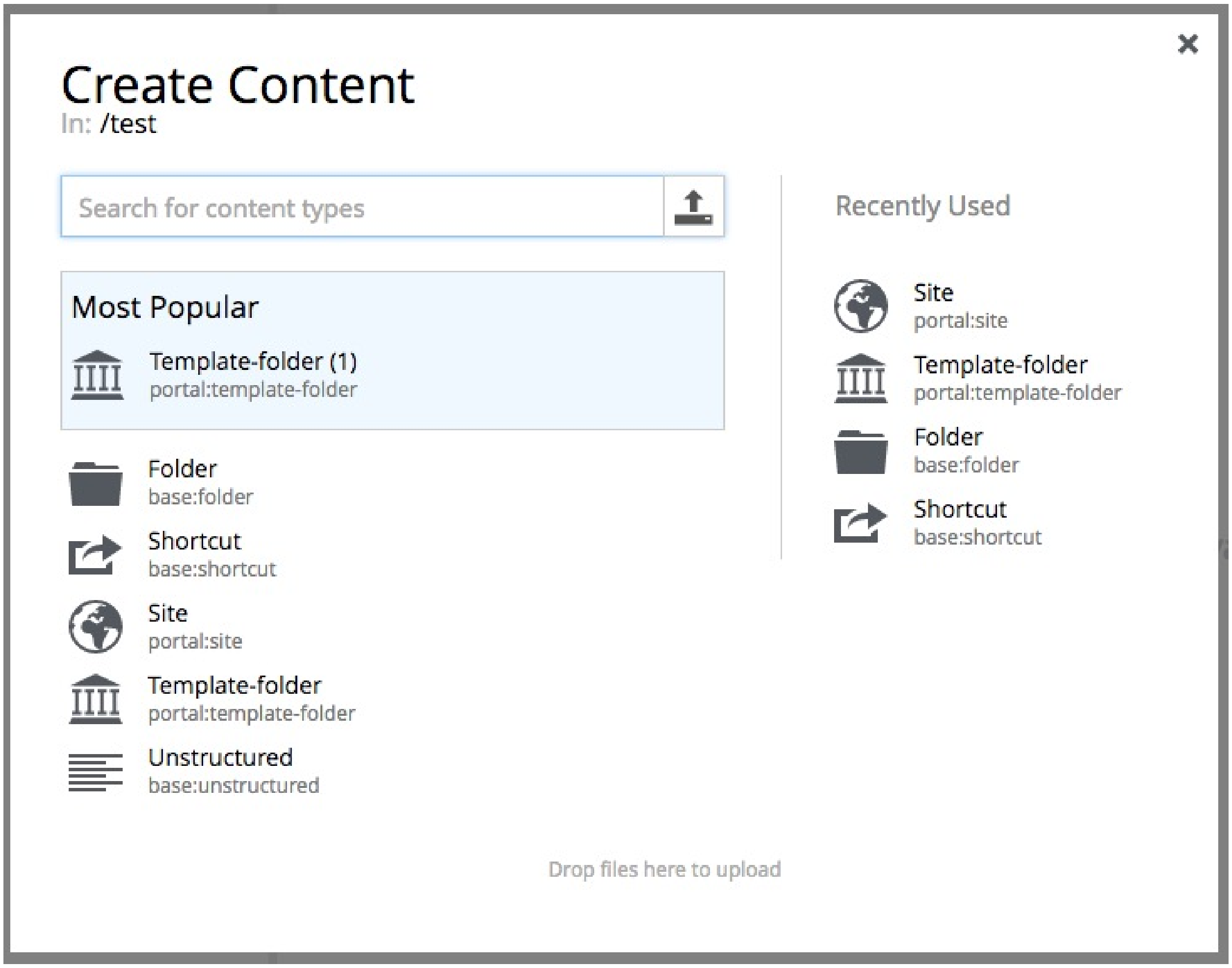
Dependent items - Clean up references to selected item(s) or click unpublish to take all items offline

/onepager/_templates
/onepager/_templates/default
/onepager/people
/onepager/people/bobby-westberg
/onepager/people/bobby-westberg/bwe.jpg
/onepager/people/michael-lazell
/onepager/people/michael-lazell/headshot-square.png

Content Types

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

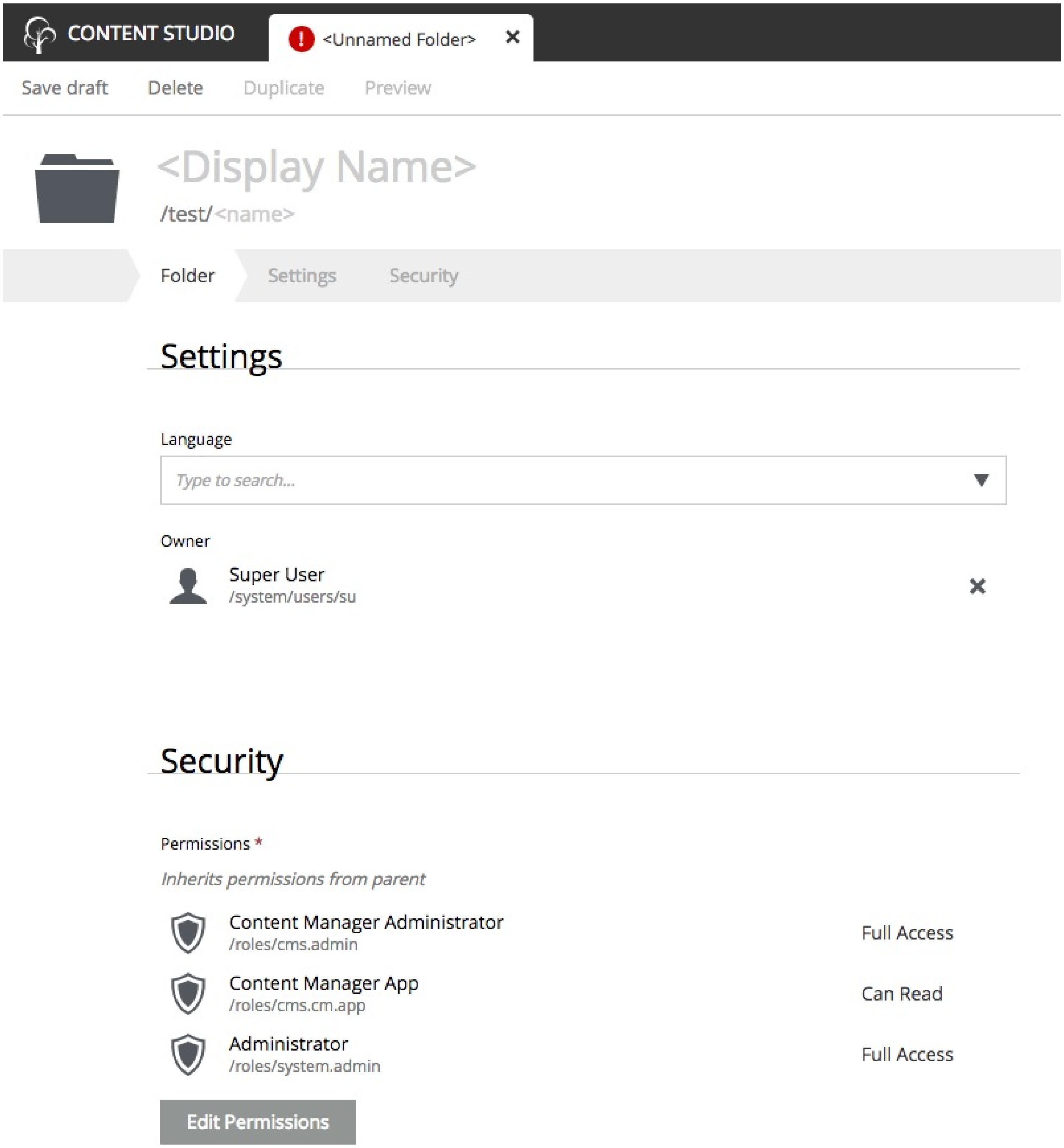
Some content types are built into enonic XP. A basic understanding of these will be essential to building sites with the Content Studio.



Folder

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Content of type **Folder** is only used to organize other content. Folders have no data fields other than the display name and path name. They also have the **Settings** and **Security** that all other content has.



Settings

Language

Type to search...

Owner

Super User
/system/users/su

Security

Permissions *

Inherits permissions from parent

Content Manager Administrator /roles/cms.admin	Full Access
Content Manager App /roles/cms.cm.app	Can Read
Administrator /roles/system.admin	Full Access

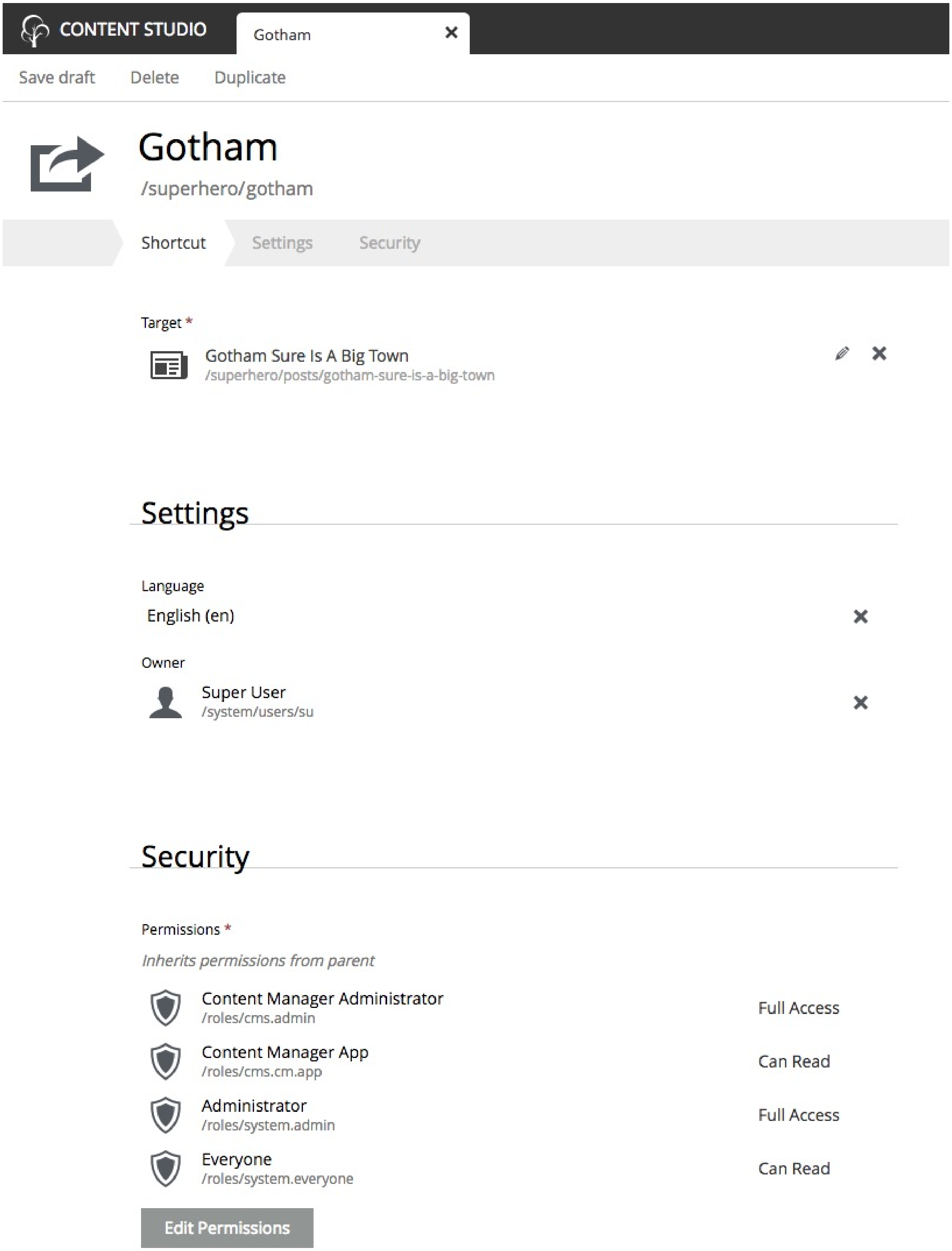
Shortcut

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Shortcuts create a content path that will redirect to another content. This allows content deep within a site to be accessed with a short URL.

For example, a content in the “Posts” folder named “Gotham sure is a big town” would normally be reached at a URL with `superhero/posts/gotham-sure-is-a-big-town`. But a shortcut as a first child of “Superhero” would make it available at `superhero/gotham`.

Each shortcut has a required **Target** field. The selected target content can be edited with the pencil icon.



Gotham

/superhero/gotham

Target *

Gotham Sure Is A Big Town
/superhero/posts/gotham-sure-is-a-big-town

Settings

Language

English (en)

Owner

Super User
/system/users/su

Security

Permissions *

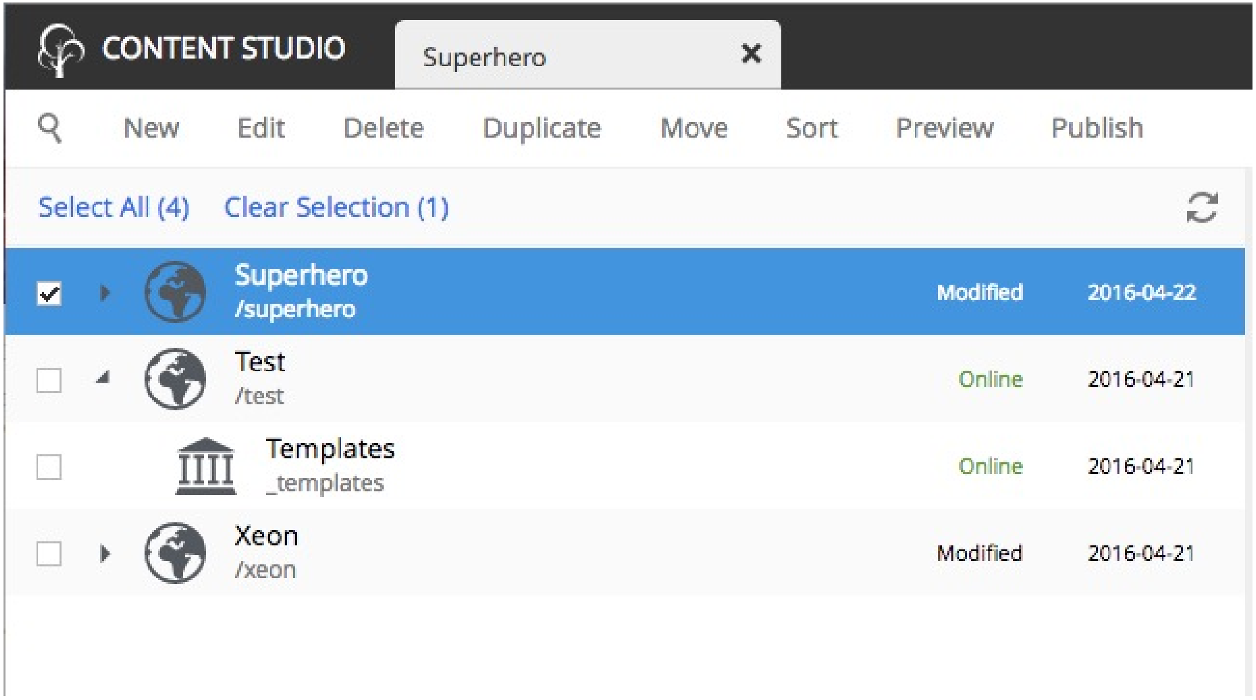
Inherits permissions from parent

Content Manager Administrator /roles/cms.admin	Full Access
Content Manager App /roles/cms.cm.app	Can Read
Administrator /roles/system.admin	Full Access
Everyone /roles/system.everyone	Can Read

Site Content

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Sites are usually created at the root level of the tree grid from a special built-in content type represented by a globe icon. Site contents have a textarea field for the site description and a dropdown selector for adding and configuring applications. Every site content will have a **Templates** folder which is automatically created along with the site content.

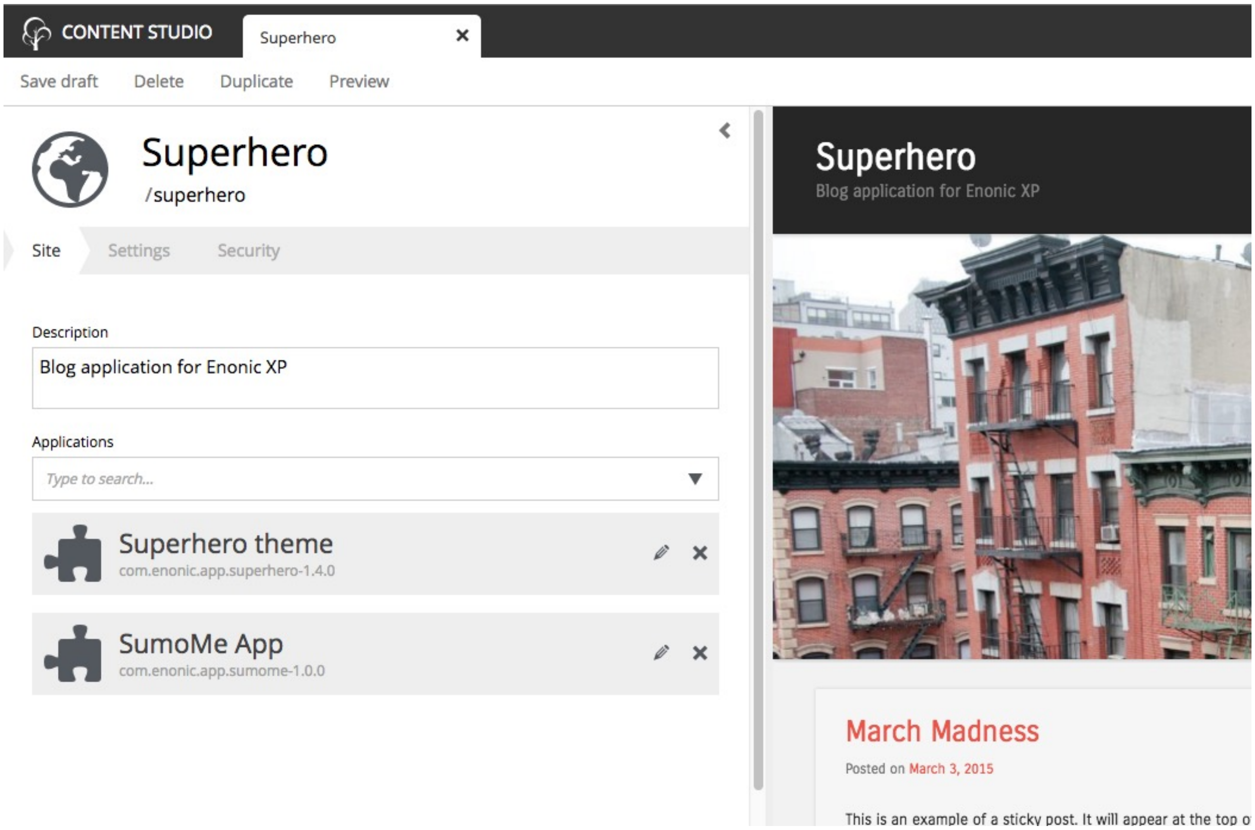


	Name	Status	Date
☑	Superhero /superhero	Modified	2016-04-22
□	Test /test	Online	2016-04-21
□	Templates _templates	Online	2016-04-21
□	Xeon /xeon	Modified	2016-04-21

Applications

Applications contain all the code behind a website. Apps are installed with the *Applications* tool and apps are used by adding them to a Site content. Some apps will have components for building a site and others will only add functionality to existing sites. Sites will need at least one app that has at least one page component before any rendering can happen.

Edit a Site content and add the desired apps with the **Applications** dropdown selector. The image below shows the edit view for the Superhero Site content. This site has two applications: one is the **Superhero theme** which was used to build the site and the other is the **SumoMe App** that adds some functionality.



Superhero
/superhero

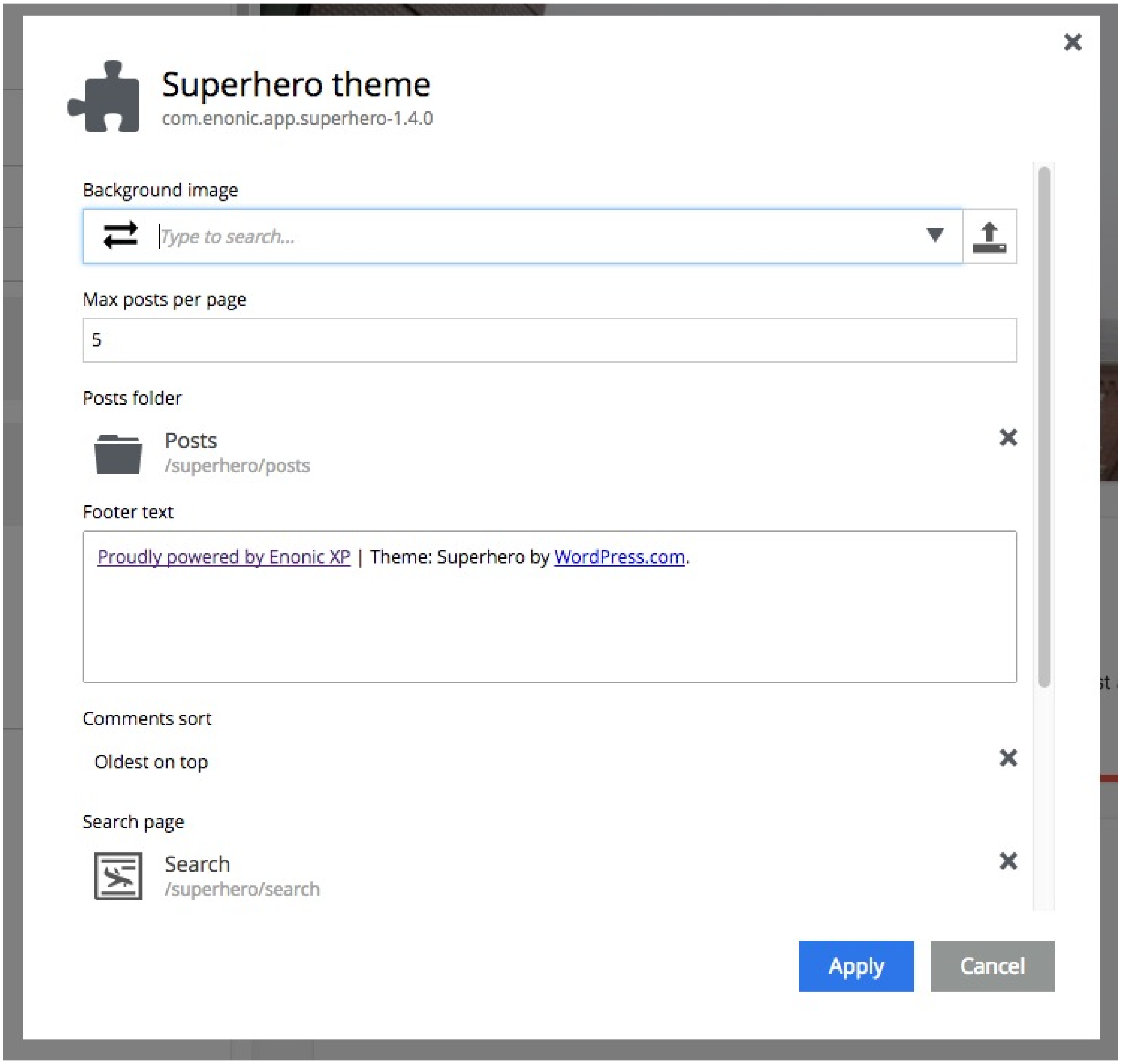
Description: Blog application for Enonic XP

Applications:
- Superhero theme — com.enonic.app.superhero-1.4.0
- SumoMe App — com.enonic.app.sumome-1.0.0

Superhero
Blog application for Enonic XP

Configuration

Apps that have been added to a Site are listed in the content edit panel with the display name and app name. The X icon removes the app from the site and the pencil icon opens the application configuration dialogue with the current values. The configuration options available are defined in the application code.



Superhero theme
com.enonic.app.superhero-1.4.0

Background image
Type to search...

Max posts per page
5

Posts folder
Posts
/superhero/posts

Footer text
Proudly powered by Enonic XP | Theme: Superhero by WordPress.com.

Comments sort
Oldest on top

Search page
Search
/superhero/search

Page Templates

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

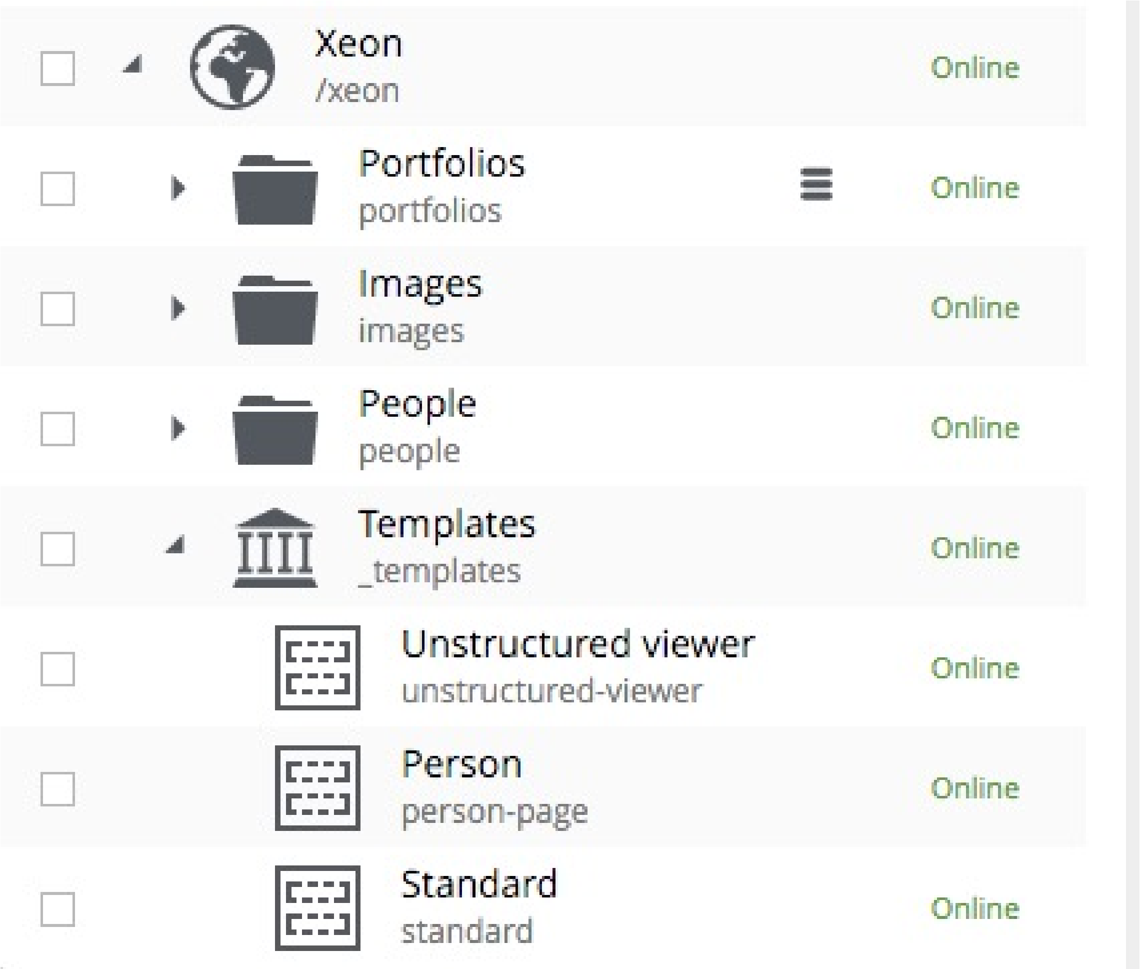
Page templates are special content that enable other content to be rendered as pages. Page templates must support one or more content types. When a request endpoint matches a content path, the content's supporting template will be used to render it. The *Page Editor panel* is used to build page templates by placing the desired components into regions on the page in the desired locations.

Page components

Page components are defined in the application code and contain the basic HTML structure of all rendered pages. Each page template uses one page component. They usually contain the page header, footer and menu. Most page components will have one or more regions where other components (parts, layouts, etc.) can be placed with the page editor. A single page component can be used by any number of page templates.

Creating page templates

Page templates are content that can only be created in a site's **Templates** folder. Create a new template content here and choose which content types will use it for rendering with the dropdown selector labeled **Supports**. A dropdown selector on the right side of the page is used for choosing the page component. Once a page component is selected, the page preview will be visible in the page editor. Use the **inspect panel** or the **component view** to add components to the region(s) in the page.



□ Xeon /xeon Online

□ Portfolios portfolios Online

□ Images images Online

□ People people Online

□ Templates _templates Online

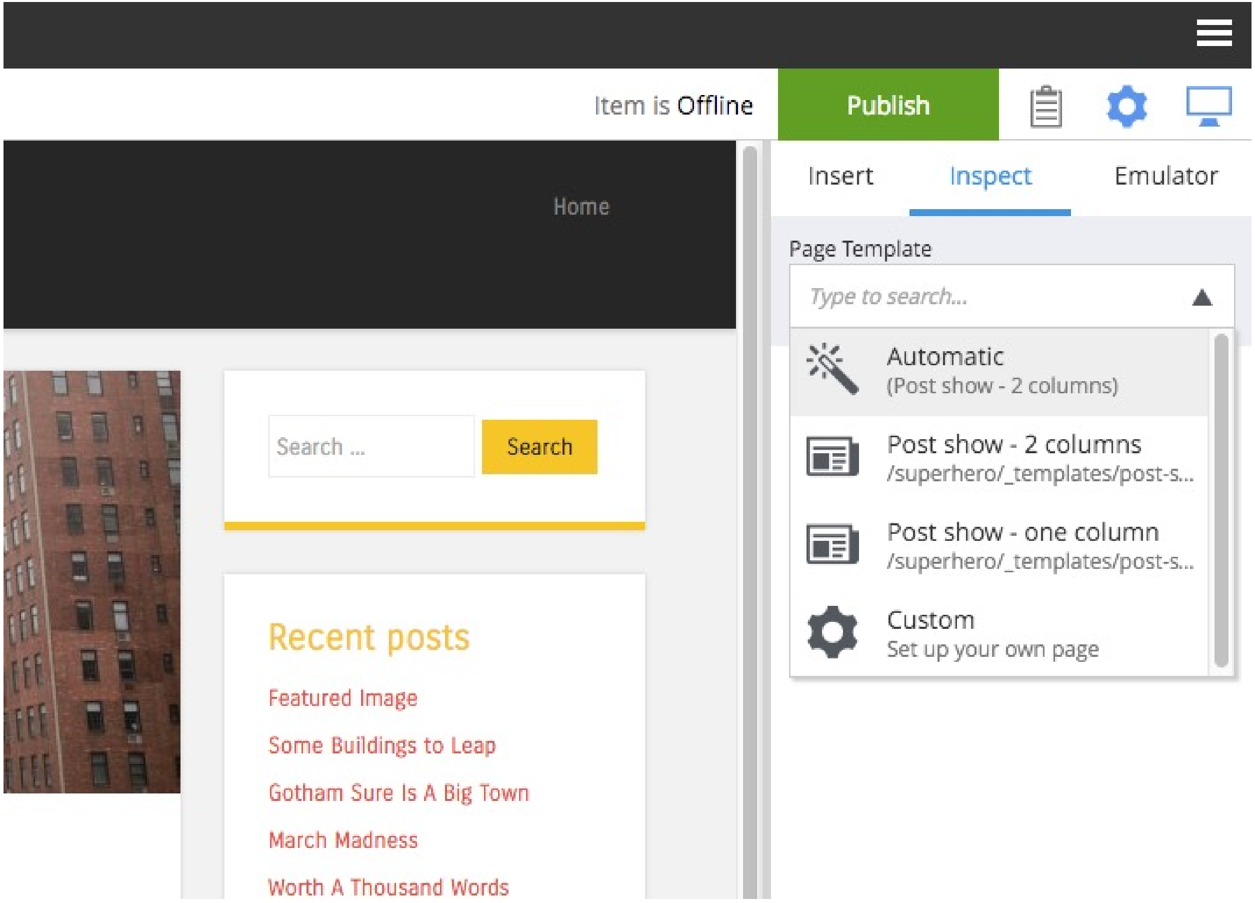
□ Unstructured viewer unstructured-viewer Online

□ Person person-page Online

□ Standard standard Online

Customizing content

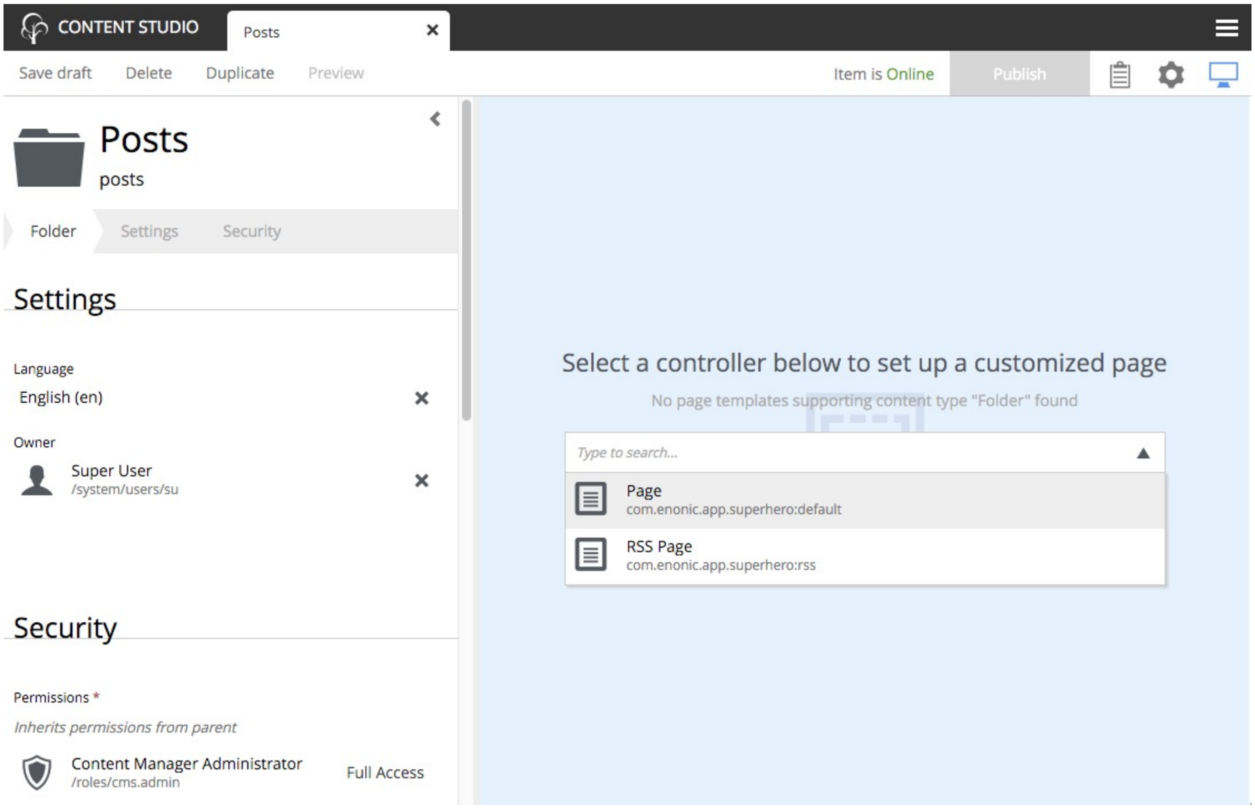
More than one page template can support the same content type. In this situation, the template that appears first in the Templates folder will be used to render the content by default. But individual content can be manually configured to use any template that supports its type. For example, in the image below there is a content type called "Post" and two templates that support the Post type, first is "Post show - 2 columns" and then "Post show - 1 column". The "Post show - 2 columns" will be used automatically by all Post content. To force an individual Post content to use the 1 column template, edit the content and select the 1 column template in the **Page Template** dropdown at the top of the **Inspect panel**.



Individual content can also be customized to render differently. To customize a content, edit it and click anywhere on the page in the page editor. A context menu appears with a "Customize" button. Once that button is clicked, components on the page can be moved with drag and drop, or removed from the page and other components can be added.

Rendering other content

Even content that has no supporting templates can be rendered as a page. For example, if a site has a **Folder** content called "Articles" and it has child **Article** content then it might be desirable to render the "Articles" folder as a page with a list of the articles that it contains. This is achieved by adding a page component to the unsupported content with the page editor. Edit the content and open the page editor. Select a page component from the dropdown selector and then add components (parts, layouts, etc.) to the page.



Settings

Language: English (en)

Owner: Super User /system/users/su

Security

Permissions *

Inherits permissions from parent

Content Manager Administrator /roles/cms.admin — Full Access

Select a controller below to set up a customized page

No page templates supporting content type "Folder" found

Type to search...

Page com.enonic.app.superhero:default

RSS Page com.enonic.app.superhero:rss

Templates folder

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The **Templates folder** is just like a regular folder except that it has a special icon and it can only contain **Page-template** content. When a Site content is created, a Templates folder is automatically created with it as a child of the Site.

Unstructured

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Unstructured content cannot be edited in the Content Studio except for the display name, the content name, and the settings and security. Content of this type is meant to be used by applications to store data when the structure is not known. Form entries are often stored as unstructured content to avoid the need to create a custom content type for each form on a site. The stored data cannot be viewed in the Content Studio without a custom page component or page template that supports the Unstructured content type.

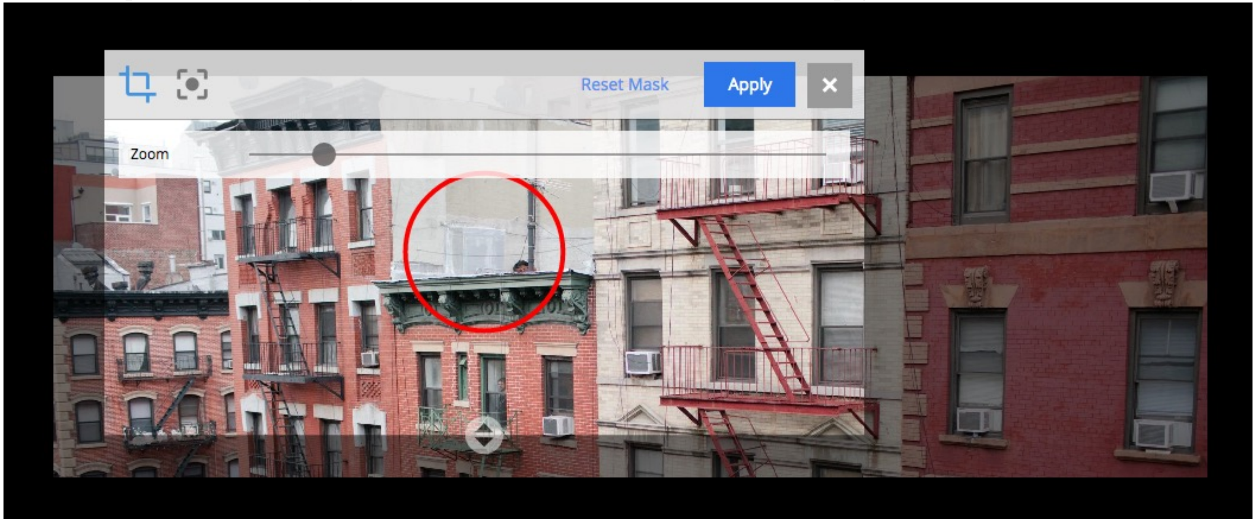
Image content

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

An image content is created when an image file is uploaded. Storing images as content allows them to be indexed for searches and have the same language and security settings as other content. Image content items have fields for Caption, Artist, Copyright, Tags and Text. Image content also has fields that are automatically filled in with any Exif data the image file contains. The image file itself can be swapped out with the upload button or by dragging a file onto the image. Images can be cropped and a focal point can be added in the editor.

Cropping

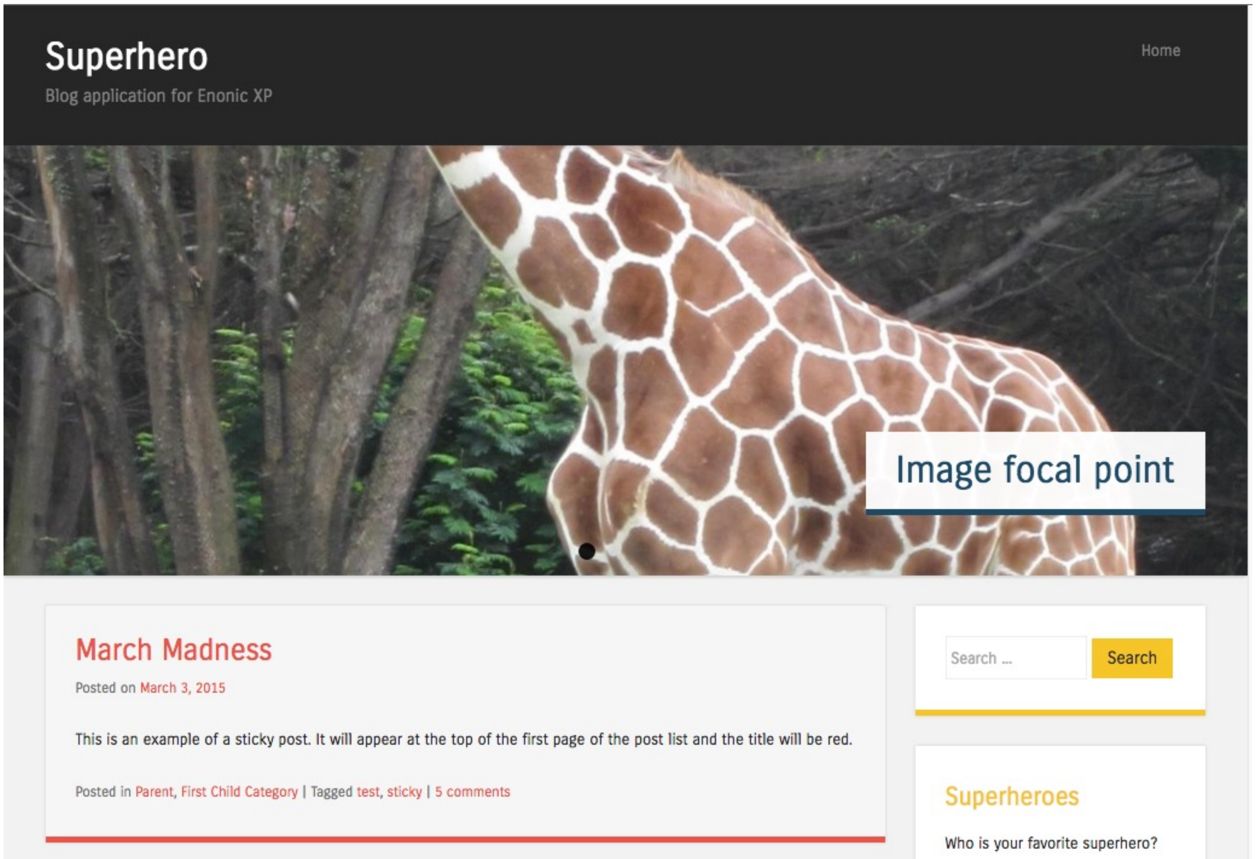
Clicking the crop icon (above the image) will darken the page outside of the image preview and the editing tools. The zoom slider will make the preview larger and parts of the image will extend into the dark regions of the page. The image can be moved by clicking and dragging it around. The aspect ratio can be changed by clicking and dragging the circle (arrows icon) at the bottom of the picture. Make the necessary adjustments so that the part of the picture you want to keep is within the highlighted area. The **Apply** button will restore the page to normal edit mode.



Focal point

Images can be displayed on a web page with a different aspect ratio than the original. When this happens, the top and bottom or the left and right edges of the picture will be automatically cropped. This can cause the subject of the image to be lost. For example, the heads of people in a portrait image could be cut off when the image is rendered with a landscape ratio. Setting a focal point on an image ensures that the subject will always be in the picture, no matter the ratio used to render it.

Click the focal point icon. A red circle appears in the center of the image preview. Click on the part of the picture that you want to always keep in frame and then click the Apply button. Once a focal point is set, its location will be marked with a red circle when the content is in edit mode.



March Madness

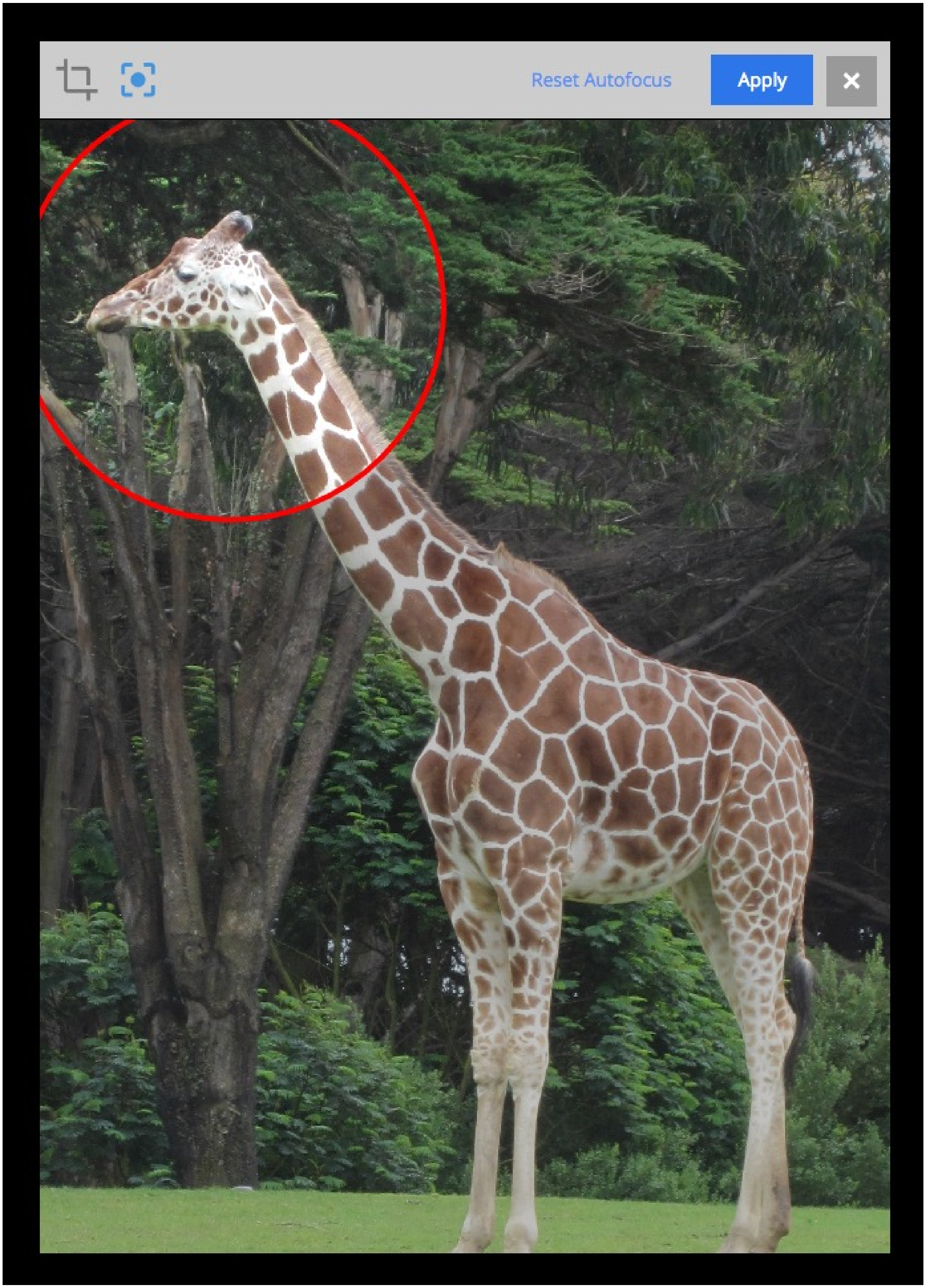
Posted on March 3, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

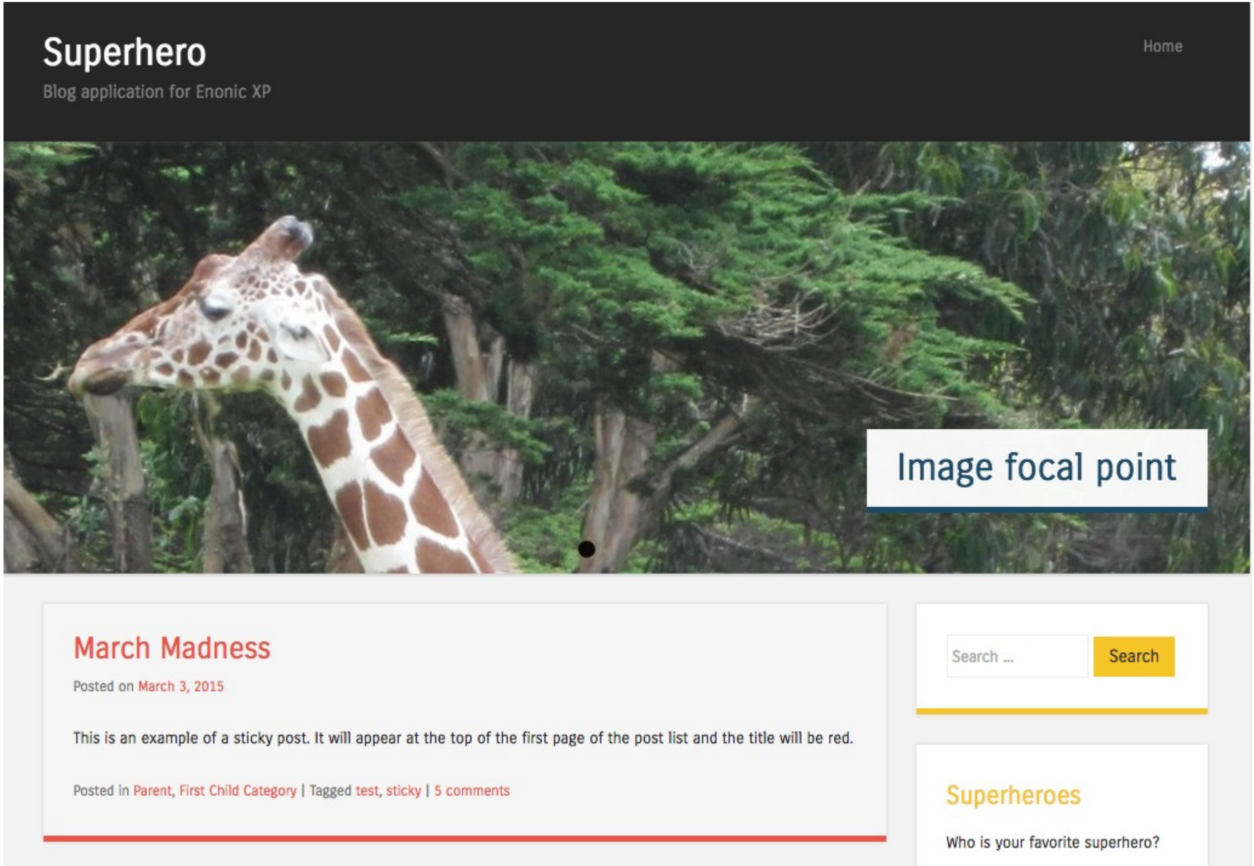
Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Who is your favorite superhero?

In the image above, the original picture has a tall aspect ratio. No focal point is set.


Reset Autofocus
Apply
×

Setting the focal point.



March Madness

Posted on March 3, 2015

This is an example of a sticky post. It will appear at the top of the first page of the post list and the title will be red.

Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

Superheroes

Who is your favorite superhero?

The giraffe's head remains in frame with the focal point set.

Input types

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Some of the input types will be familiar from standard web forms. Other input types are specific for editing content and configurations in Enonic XP. Most inputs can be navigated and operated with the keyboard. Inputs can have their own configurations which are defined in the application code and affect how they work.

All inputs have some common features. For example:

- Each input has a label.
- Each input can have an optional help text.
- Required fields are marked with a red asterisk.
- Input fields may be repeatable so that they can have multiple values. Repeatable inputs will have an **Add** button below the field.

Standard input types

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

These are the familiar input types that are used all over the web. They function the same as you would expect.

TextLine

A text line is used for capturing a single line of text. In the image below, the first TextLine is required and marked with a red star. The second allows multiple entries under the same label. When multiple values are enabled, the order of the entries can be changed by clicking the dotted area to the left of the entry box and dragging them up or down.

TextLine inputs may be configured with regular expressions to allow only valid values with a specified structure. The box becomes red when the value does not match the regular expression.

TextArea

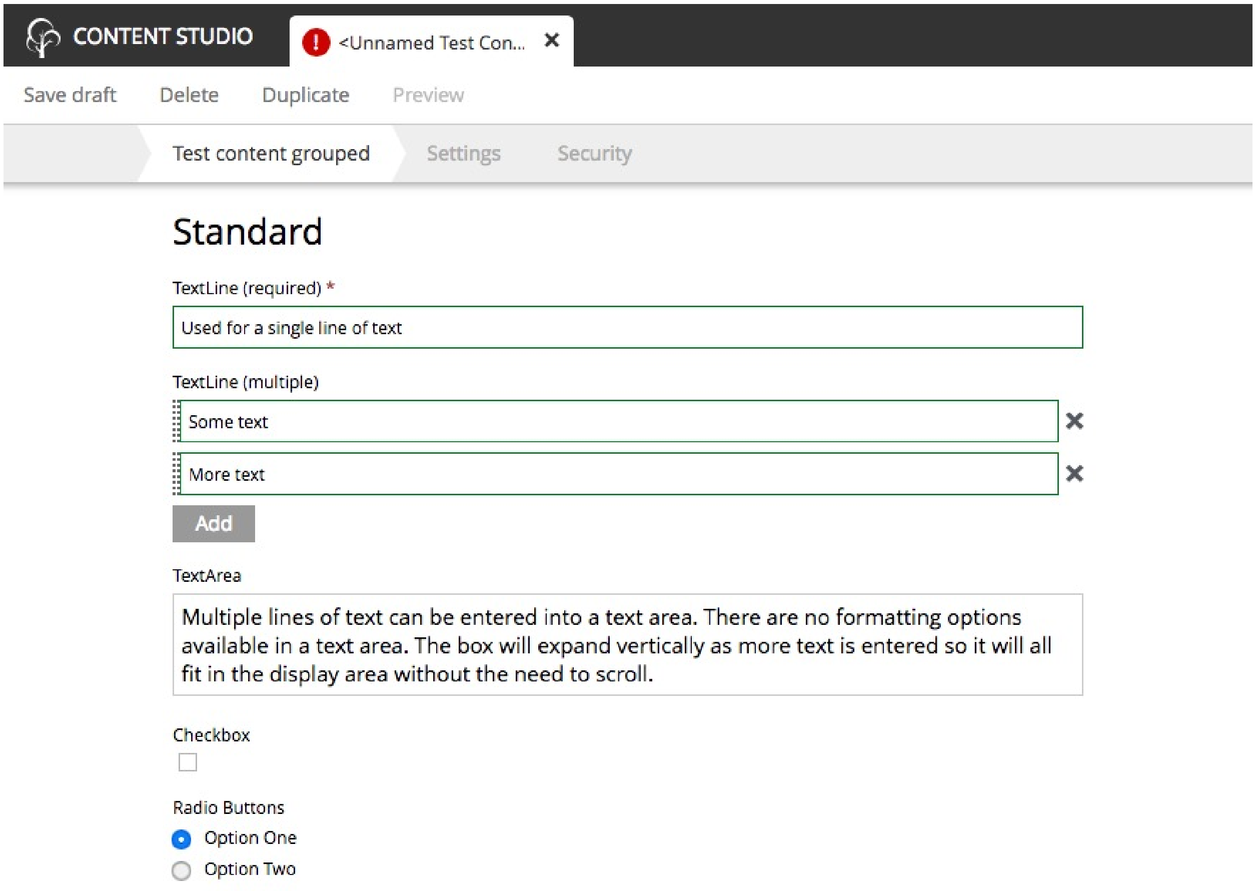
The text area is used when the expected value to be entered is too large for a text line. Multiple lines of text are allowed. There are no formatting options available for a text area. The box will expand vertically as more text is entered.

CheckBox

The checkbox can be toggled between **checked** or **unchecked** by clicking the box or the label with the mouse or by pressing the space-bar when it has focus. Checkbox position in relation to its label is configurable.

RadioButtons

This type of input allows the user to select one of several options. When one option is selected, a previously selected option will be unselected. The options can be navigated with the keyboard arrows.



Standard

TextLine (required) *

Used for a single line of text

TextLine (multiple)

Some text

More text

TextArea

Multiple lines of text can be entered into a text area. There are no formatting options available in a text area. The box will expand vertically as more text is entered so it will all fit in the display area without the need to scroll.

Checkbox

□

Radio Buttons

● Option One

○ Option Two

Selector input types

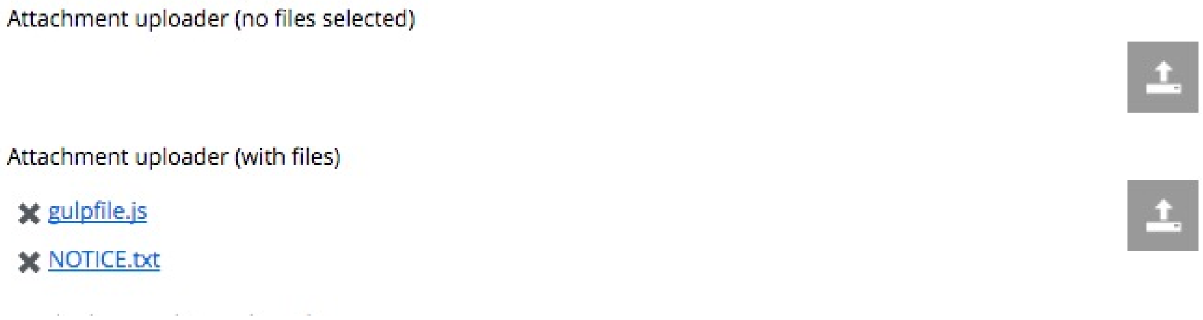
Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

These input types allow selections from a list of options.

Attachment uploader

Files can be selected for upload with this input type. The uploaded files will exist as attachments to the content being edited rather than as a content of their own.

Clicking the upload button to the right of the label will open your machine's native file browser where one or more files may be selected. Attached files are listed by name and may be removed by clicking the X to the left. The files may also be downloaded by clicking the file name.



Attachment uploader (no files selected)

Attachment uploader (with files)

✖ gulpfile.js

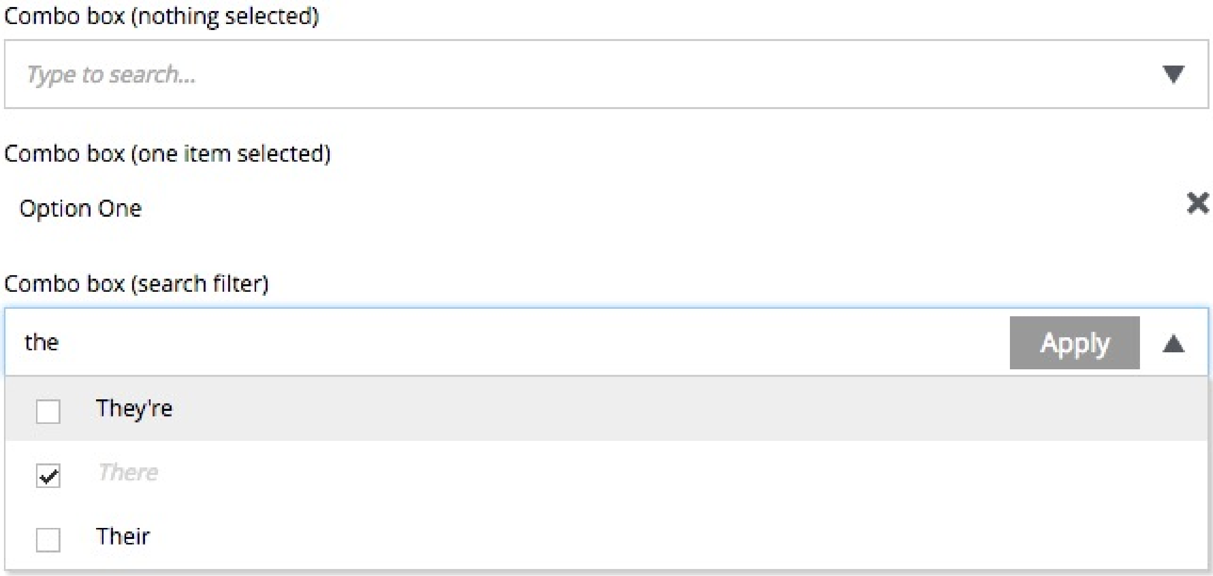
✖ NOTICE.txt

ComboBox

This input type allows the selection of one or more predefined options. Clicking the down arrow in the right of the box will open a list of options. Typing in the box will filter the options in the list.

If the input is configured for more than one selection then a checkbox will appear to the left of each option. The **Apply** button appears in the box when changes are made and it must be used to accept the changes.

Selected options can be removed by clicking the X to the right of the selection text.



Combo box (nothing selected)

Type to search...

Combo box (one item selected)

Option One ✖

Combo box (search filter)

the

□ They're

☑ There

□ Their

Content selector

Content selectors allow the selection of one ore more existing content items. They can be configured in the application code to list only specific types of content and/or content that exists at a certain path of the content tree.

Content selectors behave much like combo boxes. The down arrow in the right of the box opens the list. Typing in the search field will filter the content by name. When more than one content has been selected, the order my be changed by clicking and dragging the dotted area to the left of each content. Any selected content may be opened for editing by clicking the pencil icon. Selected content can be removed with the X icon.

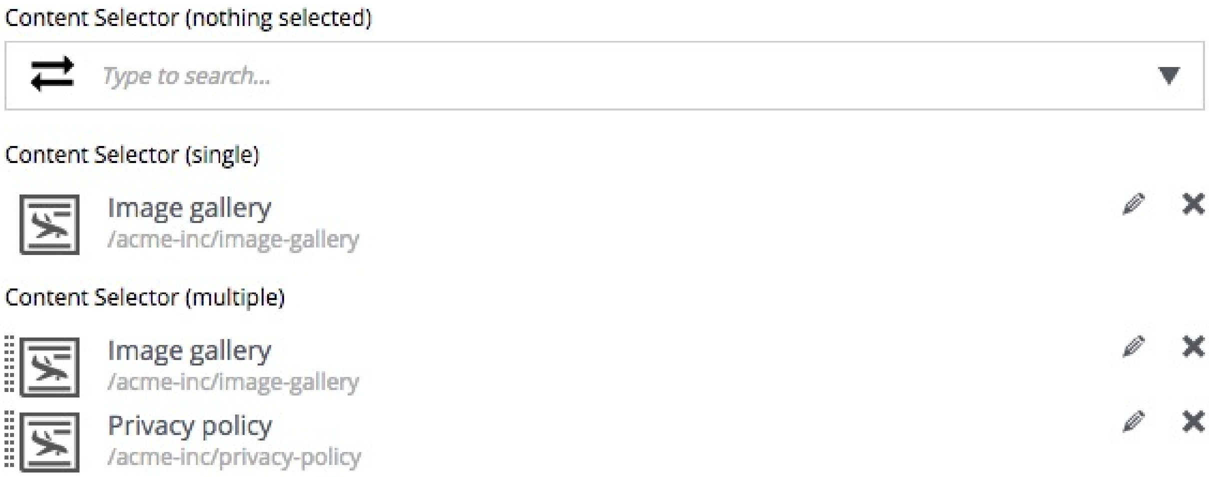
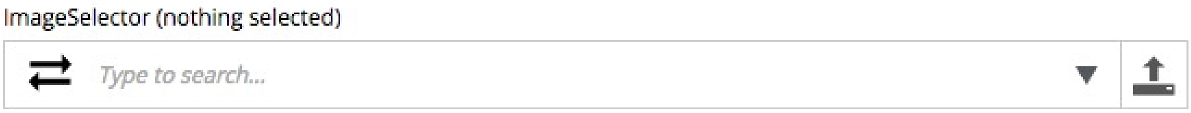
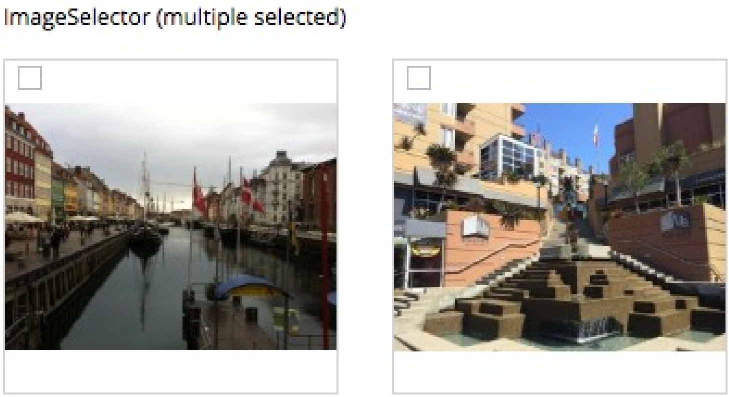


Image selector

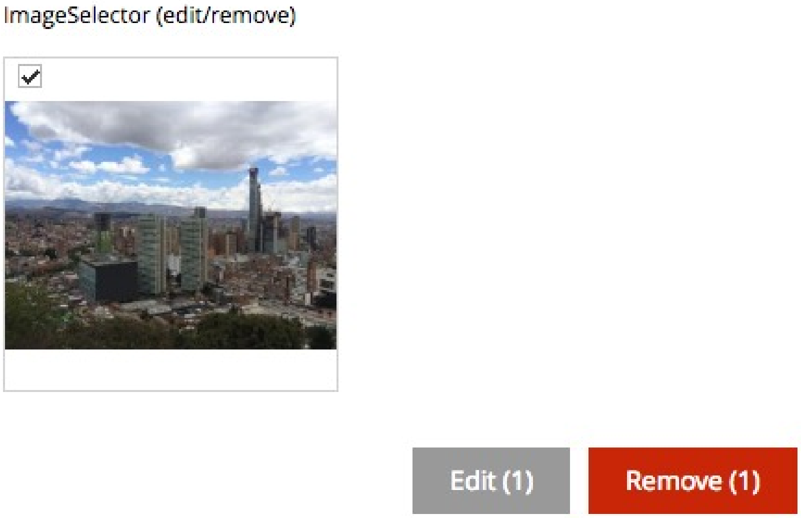
Image selectors are much like content selectors except that they allow only image content to be selected. They also allow new image content to be created and selected by clicking the upload button to the right of the input. When multiple images are selected, the order can be changed by clicking and dragging the image previews. Clicking on a selected image will reveal **Edit** and **Remove** buttons.



ImageSelector (nothing selected)



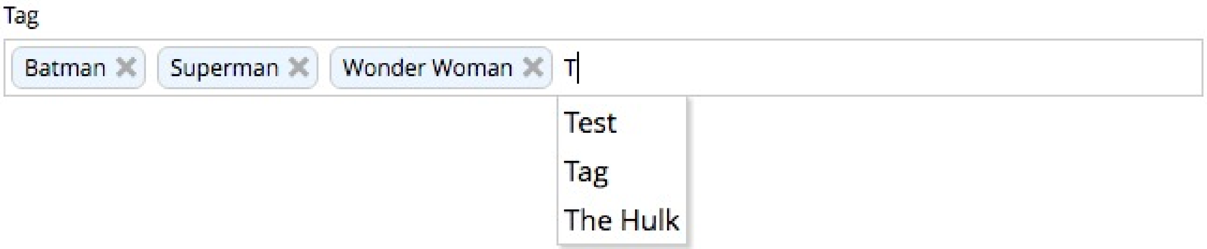
ImageSelector (multiple selected)



ImageSelector (edit/remove)

Tag

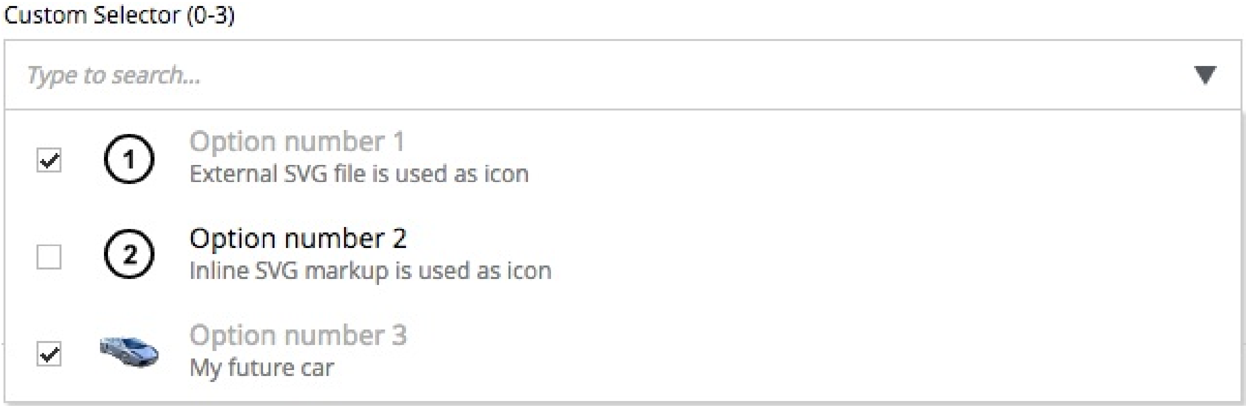
Tags allow the selection of previously entered tags and also allow the creation of new tag options. Start typing in the box to reveal matching options. If the desired tag is found in the list, use the arrow keys or click to select it. If the desired tag has not been entered before, simply finish typing and press **enter/return**. Added tags can be removed by clicking the X next to each tag.



Tag

Custom selector

Behavior of Custom selector is similar to that of Content selector except that the list of options in the dropdown is built based on custom data source. Application developer must create a service that returns JSON in correct format required by the input. Name of this service must then be specified inside the input configuration in the application code. Each option in the JSON must have a unique Id and display name (required), and, optionally, description and thumbnail (either as an external URL or inline SVG).



Custom Selector (0-3)

Type to search...

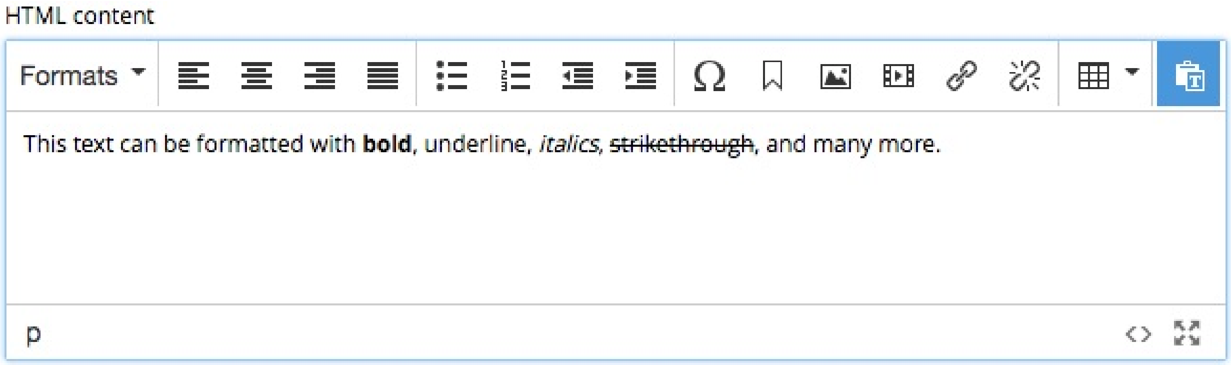
- ☑ Option number 1 — External SVG file is used as icon
- □ Option number 2 — Inline SVG markup is used as icon
- ☑ Option number 3 — My future car

Html Area input type

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

There are three input types for entering text. The Text Line and Text Area are covered in the *Standard input types* page. This section will cover the HtmlArea.

The HtmlArea is a special input type that allows text to be formatted in various ways. Code and iFrames can be embedded with macros. Images and content links can also be inserted. The editing features of the HtmlArea are the same for the Text component, except that the toolbar for the text component appears at the top of the page.



HTML content

This text can be formatted with **bold**, underline, *italics*, ~~strikethrough~~, and many more.

Toolbar

The toolbar has buttons for various formatting options as well as inserting things like pictures, links, special characters and more.

Formats The “**Formats**” button opens a menu with submenus for **Headings**, **Inline**, **Blocks**, and **Alignment**.

The **Heading** submenu has options for turning a line of text into an Html heading. These options range from h1 (largest) to h6 (smallest).

The **Inline** submenu has options for converting text to bold, italic, underline, strikethrough, superscript, subscript and code. The code option makes text monospace.

The **Blocks** submenu will wrap a block of text in HTML elements for Paragraph <p>, Blockquote <blockquote>, Div <div>, and Pre <pre>.

The **Alignment** submenu has options to align a block of text to the left, center, right, or justify.

Alignment The next four icons can be used to change alignment of selected element: left-align, center, right-align or justify.

Lists The next two icons will turn text into a bullet list or a numbered list. Pressing “enter/return” will make a new list item and pressing it a second time will end the list. Use shift + enter/return to make a new line within the list item. A sub-list can be created with the indent button.

Indent The next buttons will decrease and increase indent for the selected text. These buttons will also increase or decrease the level of a list item.

Special character This button opens a menu with 250 special characters. Selecting one will insert it at the cursor’s location.

Anchor Anchors are places in the text that links can send the user to. If a link references an anchor on the same page then the page will scroll up or down to the location of the anchor. The anchor button in the toolbar opens the **Insert Anchor** dialog where the name of the anchor is entered. The anchor name will be used as the value of the “id” attribute, so it should be lower case without spaces.

Insert/Edit image This button opens the **Insert Image** dialog. An existing image content can be selected from the “Image” selector, or a new image can be uploaded by clicking the upload button. Once an image is selected, some formatting options appear. The image can be floated to the left or right so that text wraps around it. The image can also be centered or set to full width. A checkbox allows you to keep the image at its original size. A **Cropping effect** selector has options for various aspect ratios. A caption can be entered at the bottom.

Insert macro This button opens the **Insert Macro** dialog, which contains a selector for choosing a macro. Macros allow all sorts of things to be inserted into the input, such as iframes, YouTube videos, Twitter Tweets, etc. There two built-in macros and others can be added with applications. Once a macro is selected, a form appears with inputs for the macro’s configuration settings.

Insert/Edit Link This button opens the **Insert Link** dialog. You can select existing text in the HTML Area before opening the dialog or write it directly inside the dialog. You can link to a content item, external URL, trigger download or a new email.

Unlink Pressing this button will remove a link from an element.

Table This button expands a dropdown menu enabling you to insert a new table, manage table properties or add/delete columns/rows in existing table.

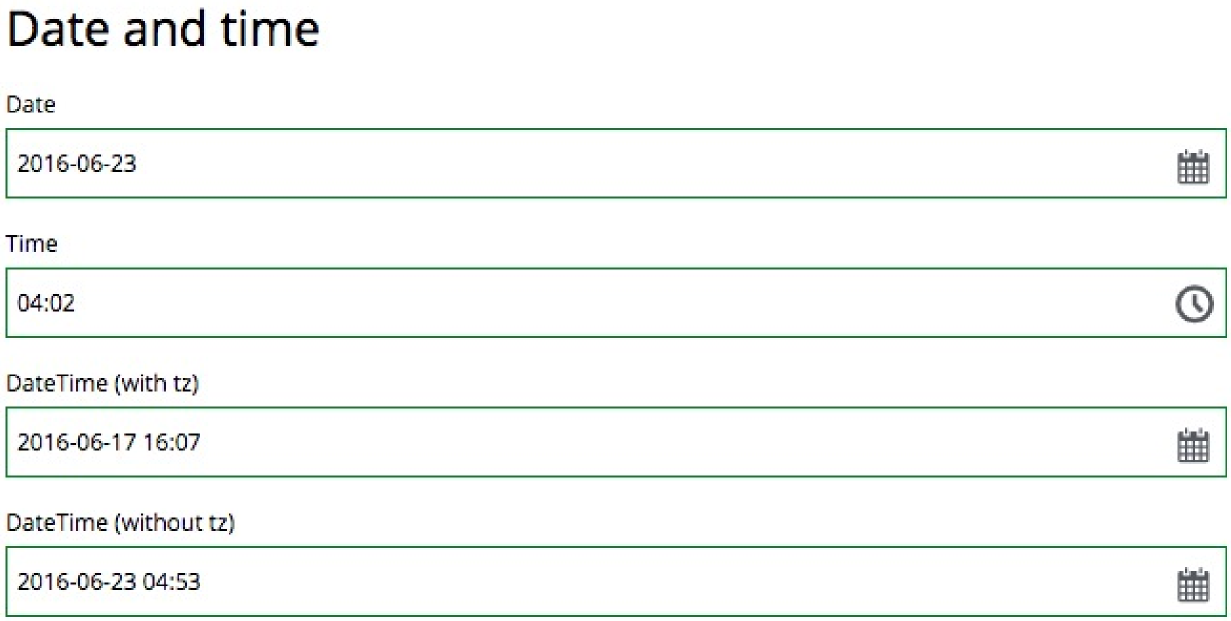
Paste as text The paste button is actually a toggle setting that controls how text is pasted. The icon is highlighted blue with the default mode which removes the source formatting from text when it is pasted. Click the icon to change modes. With plain text disabled, any formatting applied on the source of the text will come with it.

Tip: Although HTML Area comes with default toolbar, it can be customized to exclude specific (or all) tools and/or include other tools that are supported. Read about this in description of *HtmlArea* input config.

Date and time input types

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

There are three input types for entering dates and times: **Date**, **Time**, and **DateTime**. Each input has a date/time-picker tool. Values can also be manually typed into the input fields. The input turns red if the value is not a valid format. These inputs can have default values (set in the application code) and the defaults can even be relative to the date and time that the form item was created.



Date and time

Date

2016-06-23

Time

04:02

DateTime (with tz)

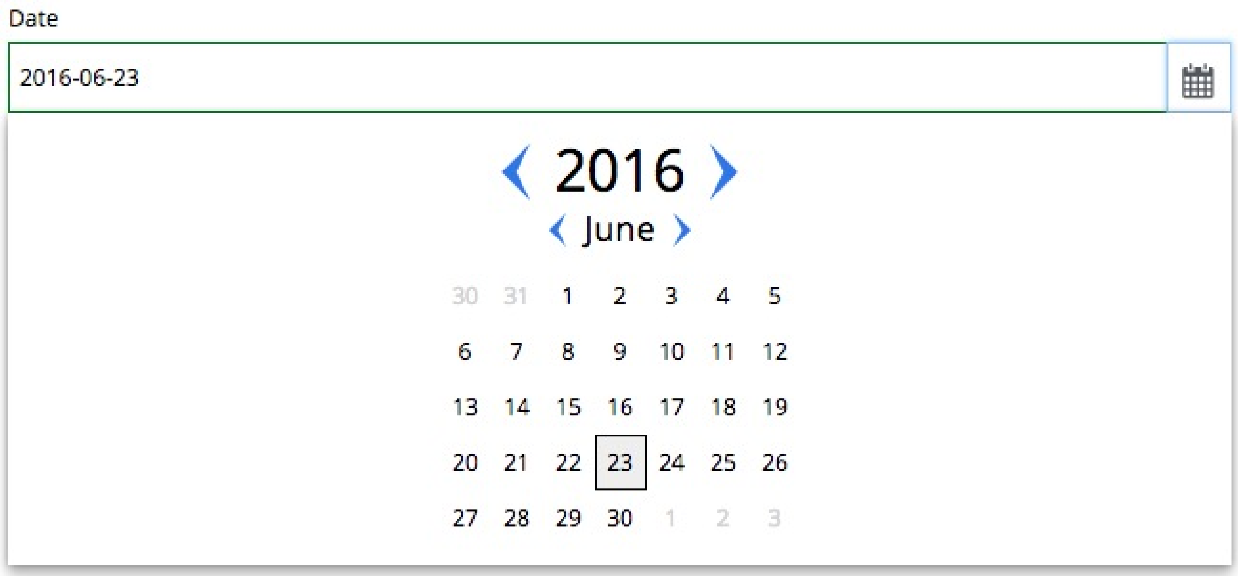
2016-06-17 16:07

DateTime (without tz)

2016-06-23 04:53

Date

The date input filed allows text to be entered in the format of YYYY-MM-DD. The input block turns red when the date is entered with an invalid format. The button on the right opens a date-picker tool. Date inputs can be configured to have a default value. The default date could even be relative to the current date, for example, one month from the time that the content with thie input was created.



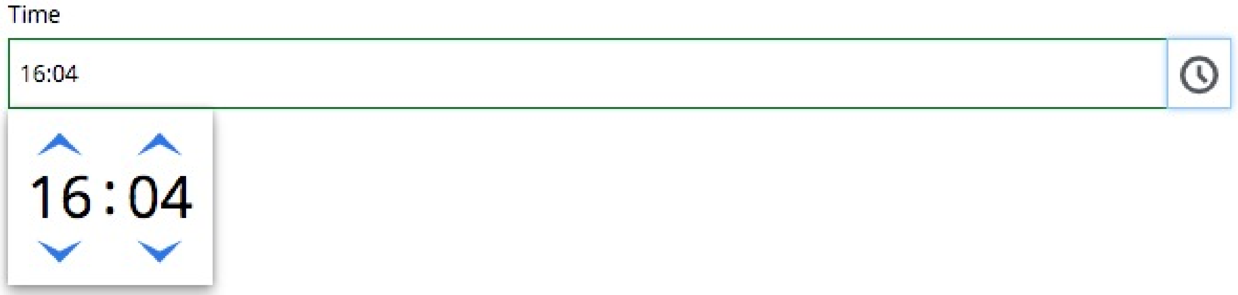
Date

2016-06-23

30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3

Time

The time can be entered manually in the 24 hour format HH:MM. Invalid entries will turn the input red. A button in the right of the form will open a time-picker tool for easily selecting the desired time. The input could be configured with a default value.



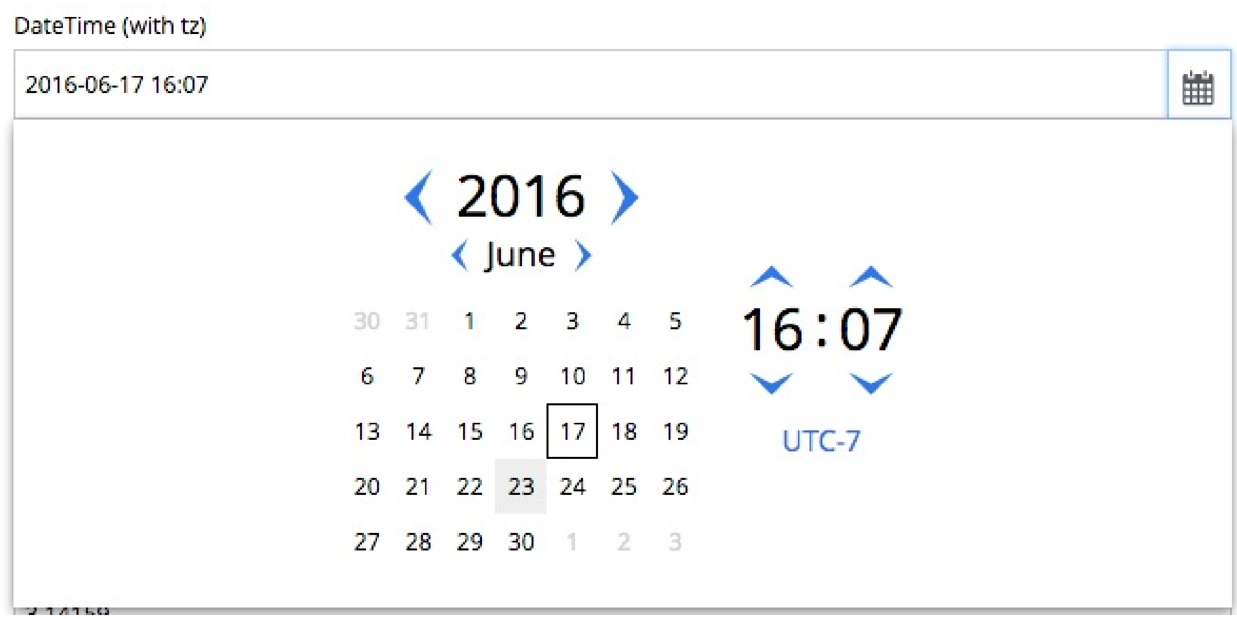
Time

16:04

DateTime

The DateTime input contains both the date and the time. A value can be entered manually in the format "YYYY-MM-DD HH:MM". A date-time picker tool can be opened with the button on the right side fo the input.

The input can be configured to include the timezone. The timezone will be the same as that of the Enonic installation server and it is not editable in the data-time picker tool. However, the timezone can be changed by manually entering a date in ISO 8601 format 2016-06-17T12:59+03:00. A default value can be set.



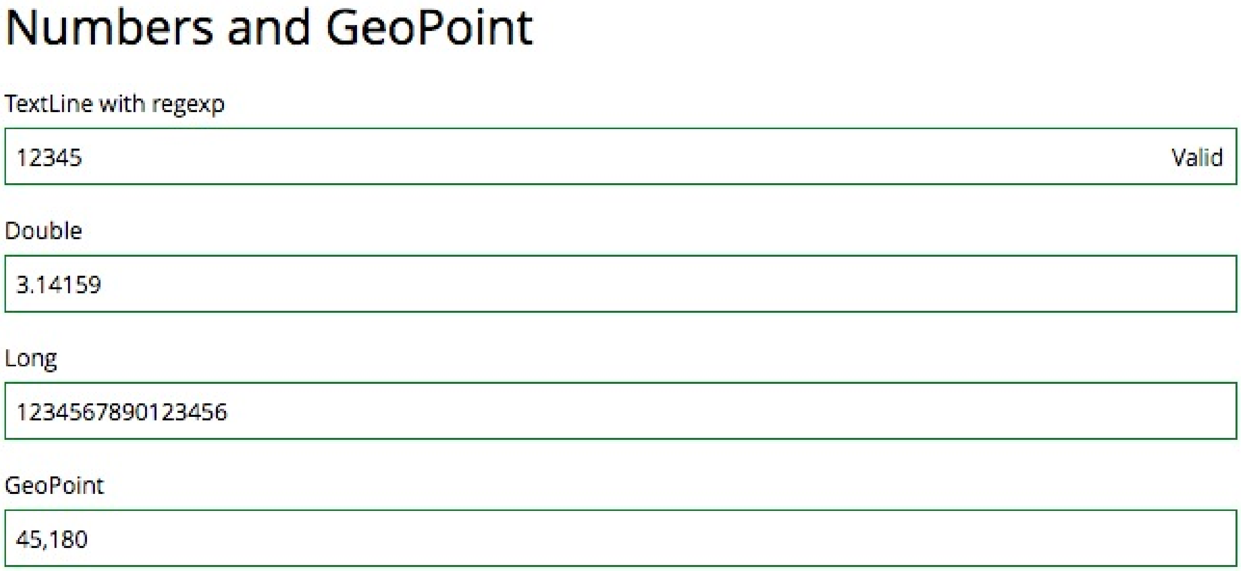
DateTime (with tz)

2016-06-17 16:07

Numbers and GeoPoint input types

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

There are three input types for entering dates and times: **Date**, **Time**, and **DateTime**. Each input has a date/time-picker tool. Values can also be manually typed into the input fields. The input turns red if the value is not a valid format. These inputs can have default values (set in the application code) and the defaults can even be relative to the date and time that the form item was created.



Numbers and GeoPoint

TextLine with regexp

12345 Valid

Double

3.14159

Long

1234567890123456

GeoPoint

45,180

TextLine with RegExp

A TextLine input can be configured with a regular expression in the application code to allow only numbers.

Double

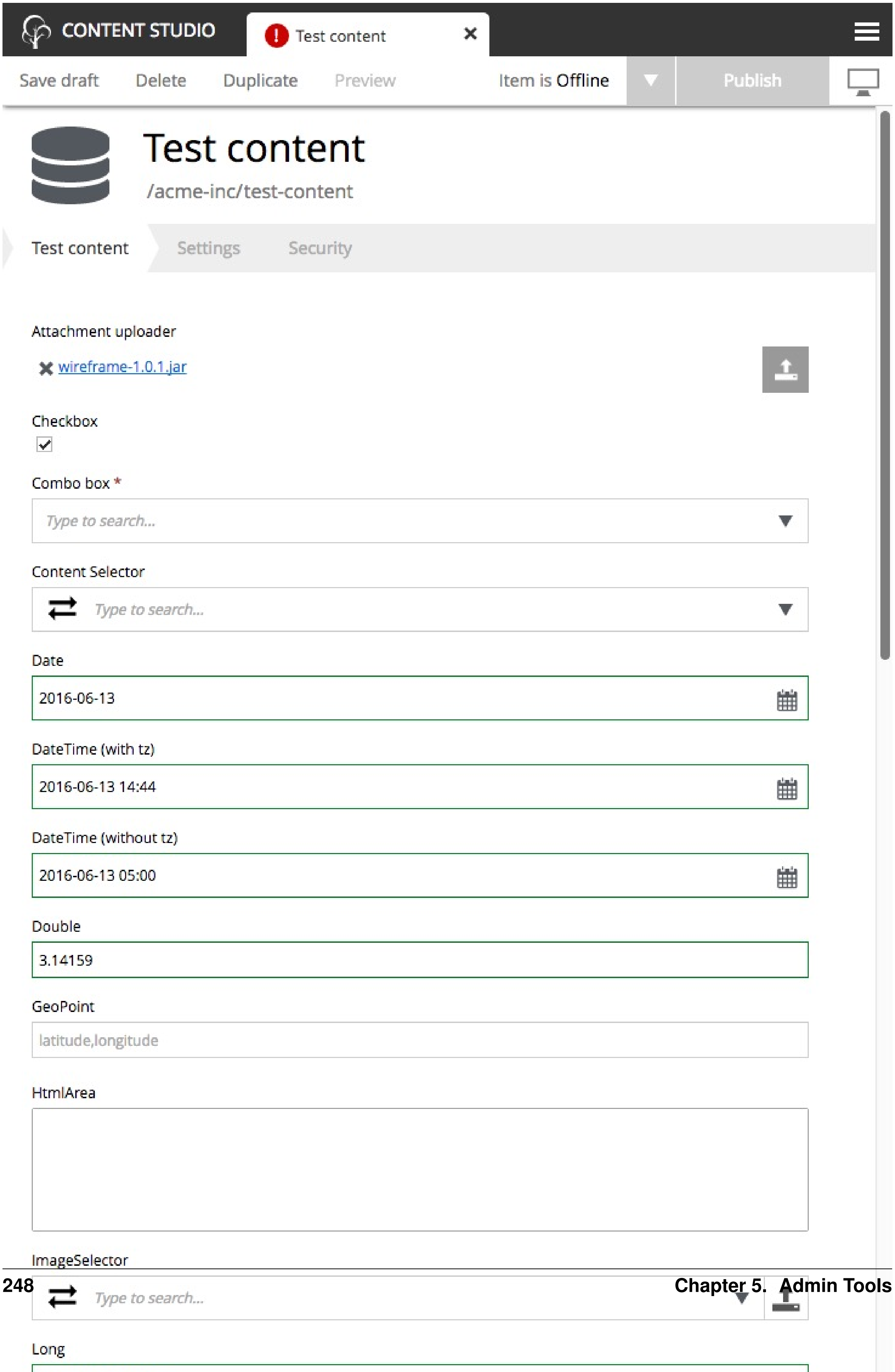
A double is a number with a decimal point.

Long

A long is a large number. The maximum value that can be entered as a long is 8999999999999999.

GeoPoint

A Geo Point stores geographical coordinates with the latitude and longitude separated by a comma.



Test content

/acme-inc/test-content

Attachment uploader

✖ wireframe-1.0.1.jar

Checkbox

☑

Combo box *

Type to search...

Content Selector

Type to search...

Date

2016-06-13

DateTime (with tz)

2016-06-13 14:44

DateTime (without tz)

2016-06-13 05:00

Double

3.14159

GeoPoint

latitude,longitude

HtmlArea

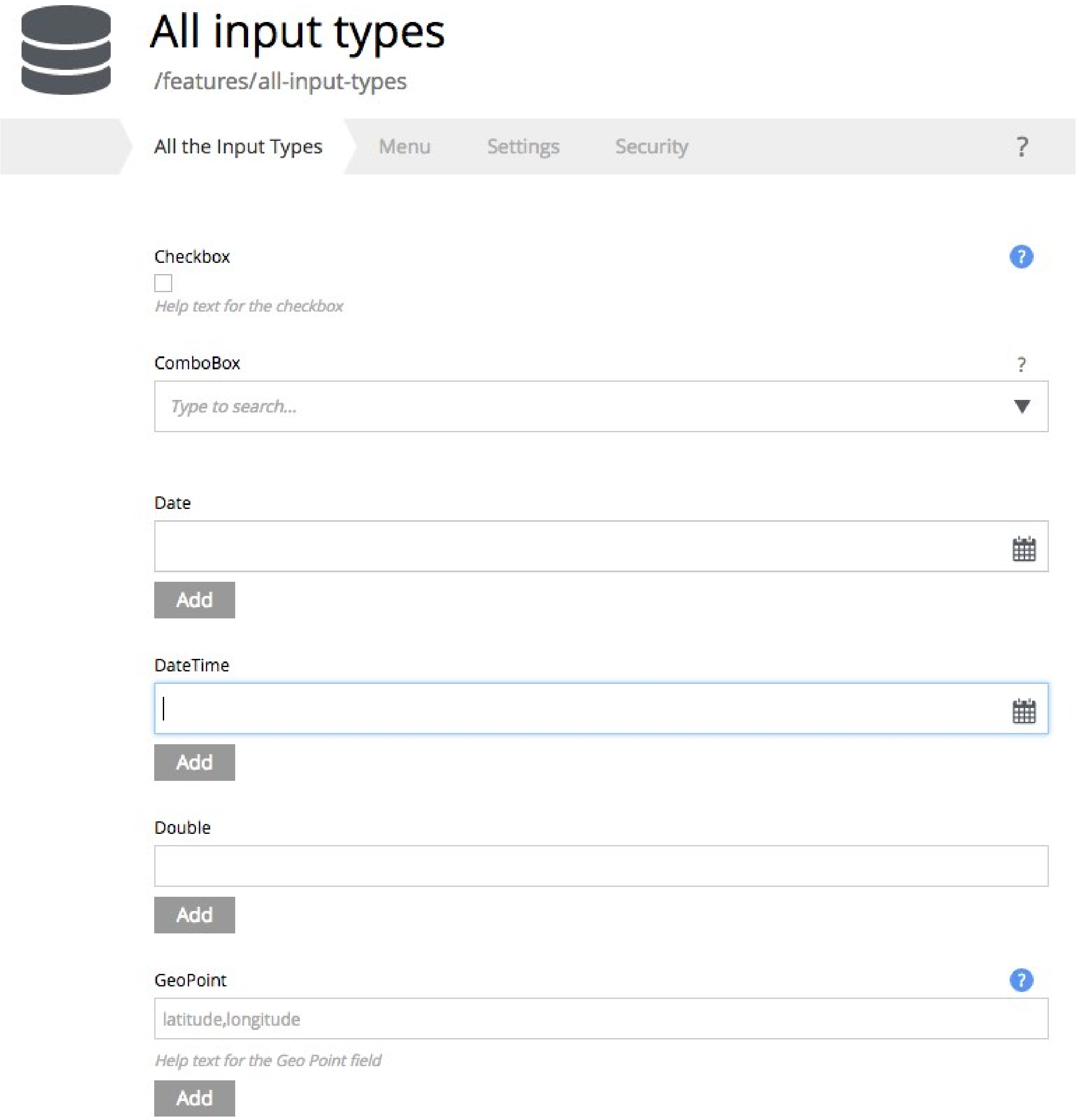
ImageSelector

Type to search...

Long

All inputs can have an optional help text that will be shown next to the input. It's hidden by default but can be turned on by clicking the "?" icon next to the input label.

Note blue "?" icons next to the Checkbox and the GeoPoint fields in the form below - for these two fields the help text is turned on. The ComboBox field also has a help text, but it's hidden - the "?" icon is inactive and must be clicked to show the help text.



All input types

/features/all-input-types

Checkbox ?

□

Help text for the checkbox

ComboBox ?

Type to search... ▼

Date

DateTime

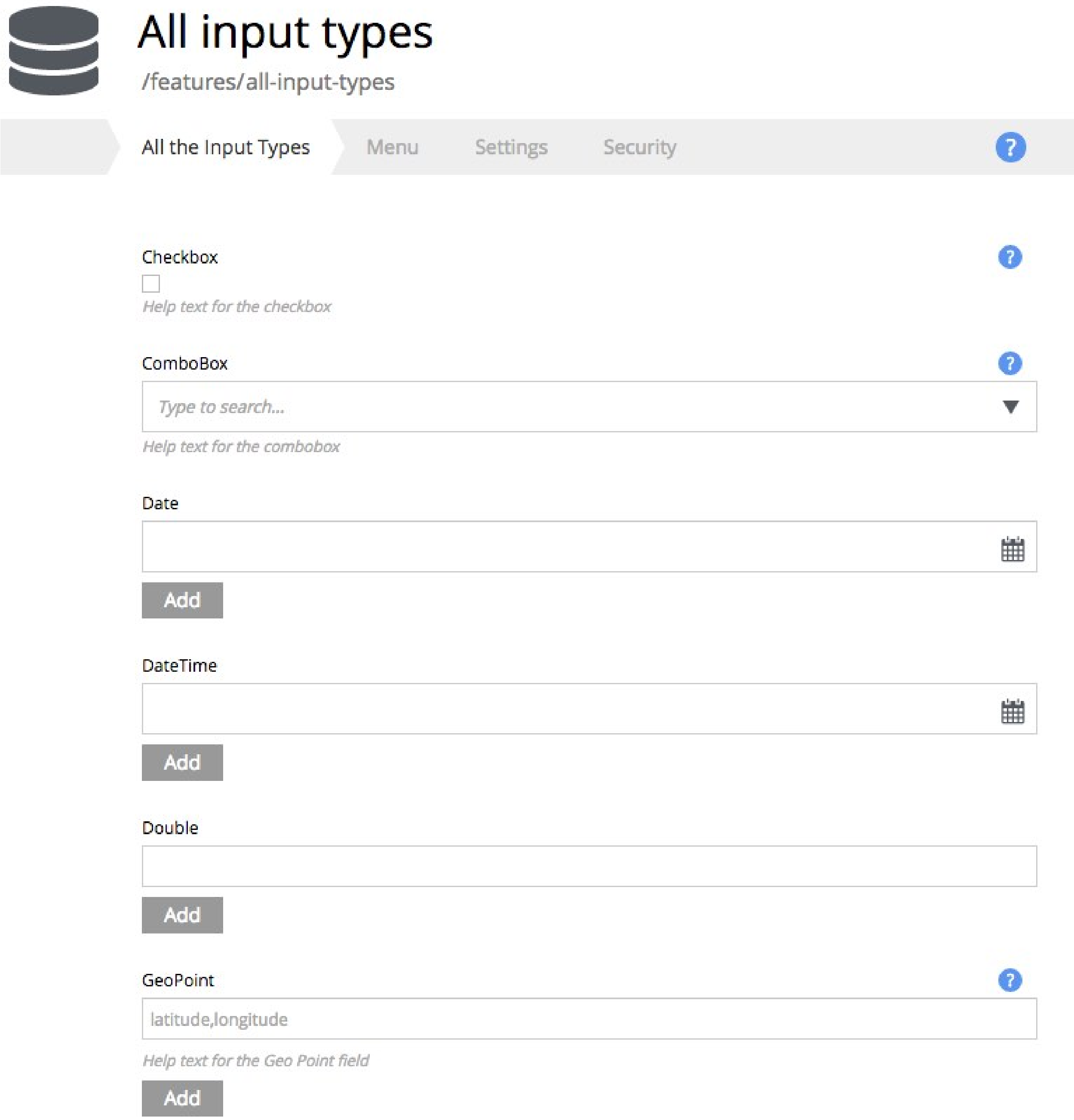
Double

GeoPoint ?

latitude,longitude

Help text for the Geo Point field

It's also possible to turn on help text for all inputs on the form at once by clicking the "?" icon inside the Content Wizard toolbar.



Content Security

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Security of a content is determined by the principals and permissions in the content's security settings. Principals are **Users**, **Groups** and **Roles** which are managed in the *Users* admin tool. Principals are added to each content. Permissions are granted to the principals of the content.

Principals

Individual users could be added to each content. But security is easier to manage by adding groups and roles to each content and then granting the appropriate permissions to these groups and roles. When a group is added to a content then all users who are members of that group will have the group permissions for that content. Similarly, when a role is granted permissions to a content, all users who have that role will also have those permissions.

Permissions

Permissions are granted to principals on a per-content basis. This means that changing a principal's permissions for one content does not affect that principal's permissions for other content. Below is a list of permissions that can be applied to each principal for any content.

- **Read**: The principal can see this content.
- **Create**: The principal can create child items under this content. For example, if a user has this permission on a folder then the user can create new content in the folder.
- **Modify**: The principal can edit this content and save changes in the draft branch.
- **Delete**: The principal can delete this content from the draft branch. If the content is published then its status can be changed to "Pending delete".
- **Publish**: The principal can publish this content to the master branch. Can also remove content that is "Pending delete".
- **Read permissions**: The principal can see this content's permissions.
- **Write permissions**: The principal can change this content's permissions.

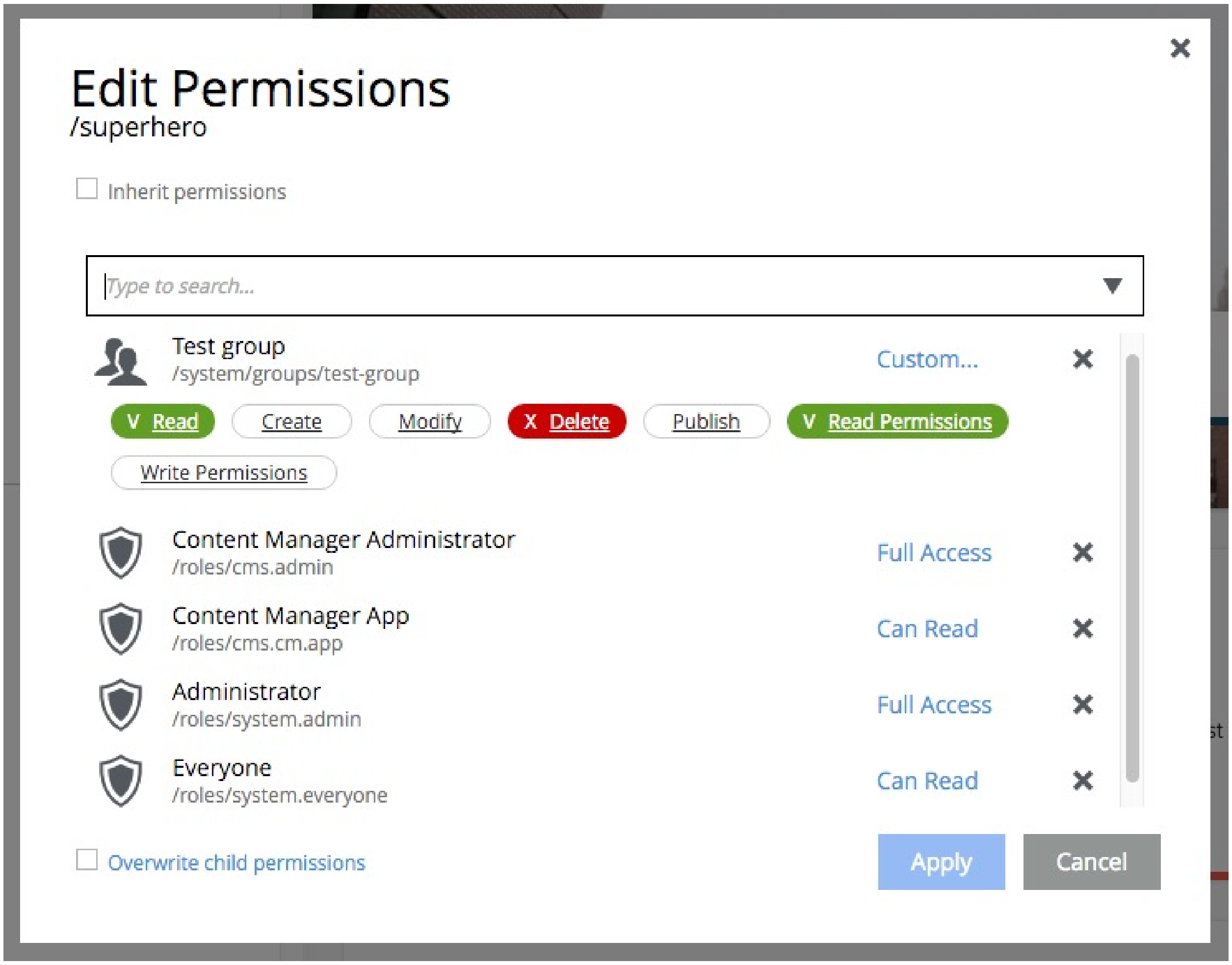
Permissions can also be denied to a principal, even if the permissions would otherwise be granted from another principal. For example, all content editors might be added to a group called "Content editors" which has the **Can Publish** permissions. But new content editors might break stuff, so they would also be added to a group called "Noobs". Users in this group could be prevented from publishing and deleting content by denying those permissions to the group.

Public content

For a content to be accessible to the public, (meaning users who are not logged in), it must have the role **Everyone** with the **Read** permission. The content must also be published. Content without the **Everyone** role can only be seen by users who are logged in and have read access to it through one of the content's principals.

Editing permissions

When a site is first created, it will have the roles **Administrator** and **Content Manager Administrator** with full access. It will also have the role of **Content Manager App** with read access. When content is created it will inherit the security settings from its parent content. The security settings for any content can be changed through the **Edit permissions** dialogue. This can be accessed with the **Edit Permissions** button in the Security section when editing a content, or by the button in the details panel in browse view.



Edit Permissions
/superhero

□ Inherit permissions

Type to search...

Test group
/system/groups/test-group — Custom...

V Read | Create | Modify | X Delete | Publish | V Read Permissions | Write Permissions

Content Manager Administrator
/roles/cms.admin — Full Access

Content Manager App
/roles/cms.cm.app — Can Read

Administrator
/roles/system.admin — Full Access

Everyone
/roles/system.everyone — Can Read

□ Overwrite child permissions

Opening the **Edit Permissions** dialogue shows a list of the current principals and their permissions for the content. By default, all content will inherit permissions from its parent. Permissions cannot be changed until the box for **Inherit permissions** is unchecked. Any changes that are made will also be applied to all child items that directly or indirectly inherit permissions from this content. There is also a checkbox at the bottom of the dialogue labeled **Overwrite child permissions**. Checking this box will force the new permissions on child content that do not inherit permissions.

When the **Inherit permissions** checkbox is unchecked, a dropdown selector will appear for adding new principals. The permissions for each principal in the list can be determined by the settings in blue letters on the right. Clicking a setting will open a menu with the available options that can be applied, which are listed below:

- Can Read: Has only the **Read** permission.
- Can Write: Has permissions for Read, Create, Modify, and Delete.
- Can Publish: Has permissions for Read, Create, Modify, Delete, and Publish.
- Full Access: Has all permissions.
- Custom: Any combination of permissions.

Selecting "Custom" will list all of the permissions. Clicking on a permission will toggle it between white (not granted), green (granted) and red (denied).

Component types

Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Enonic XP has five types of components that can be added to regions on a page. These component types are listed under the "Insert" tab of the page editor. This section covers each type of component in detail. See the *Page Editor panel* and *Inspection Panel* pages for more information about editing pages.

Image component

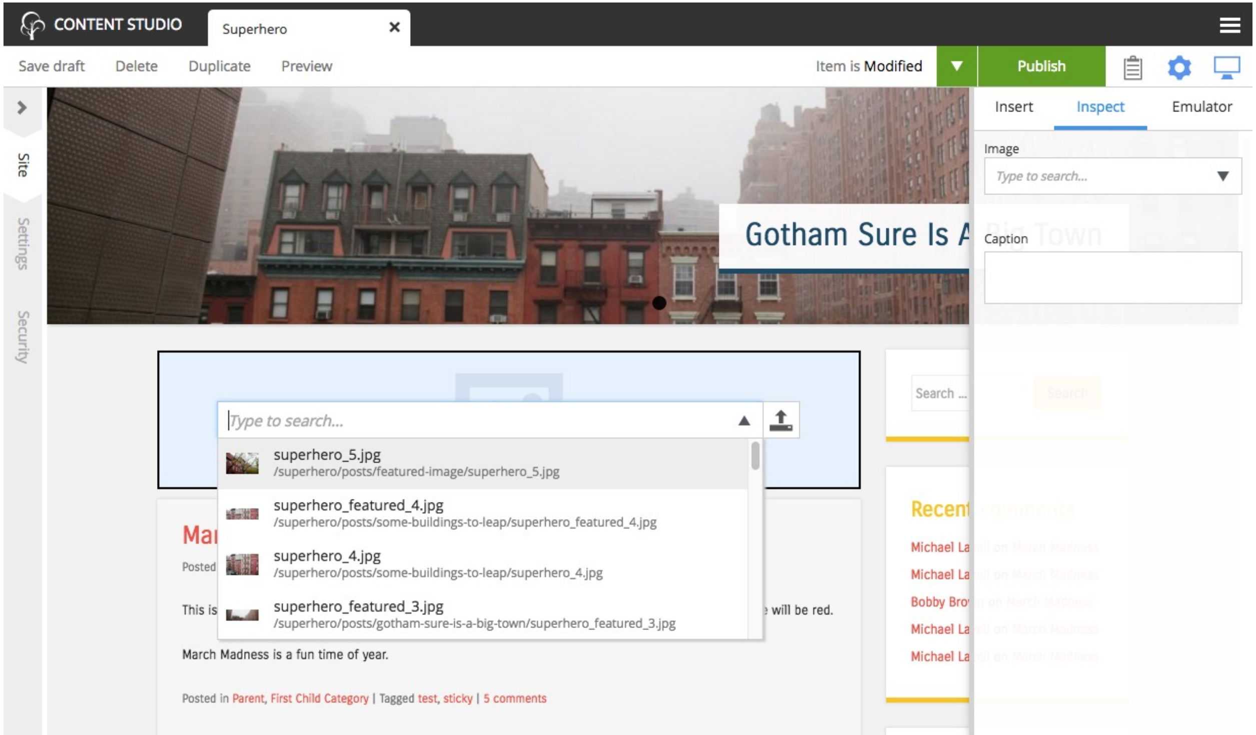
Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The Image component allows content editors to place an image into any region on a page without writing any code.

In the *Page Editor panel*, drag an Image component placeholder from the inspect panel to the desired region on the page. Once placed, the empty image placeholder contains an image selector that can be used to find and select any previously uploaded image content. If the name of the image is known, simply start typing it in the box to filter the list of images. If the name is not known, use the down arrow to open a list of images to choose from. Note that the list will contain all image content items in the XP installation, including images that were created in a different site of a multi-site environment.

If the desired image does not already exist as a content, simply upload it with the button on the right side of the image selector dropdown. The new image content will be created as a child of the page being edited, but it could be moved later if needed.

The inspect panel will also show the dropdown image selector as well as a text area for writing an image caption. If a caption is entered, it will appear below the image.


CONTENT STUDIO
Superhero
Save draft
Delete
Duplicate
Preview
Item is Modified
Publish
Insert
Inspect
Emulator
Image
Type to search...
Caption
Site
Settings
Security
Gotham Sure Is A Big Town
Type to search...
superhero_5.jpg
/superhero/posts/featured-image/superhero_5.jpg
superhero_featured_4.jpg
/superhero/posts/some-buildings-to-leap/superhero_featured_4.jpg
superhero_4.jpg
/superhero/posts/some-buildings-to-leap/superhero_4.jpg
superhero_featured_3.jpg
/superhero/posts/gotham-sure-is-a-big-town/superhero_featured_3.jpg
Search ...
Recent
Michael La
Michael La
Bobby Bro
Michael La
Michael La
Mar
Posted
This is
will be red.
March Madness is a fun time of year.
Posted in Parent, First Child Category | Tagged test, sticky | 5 comments

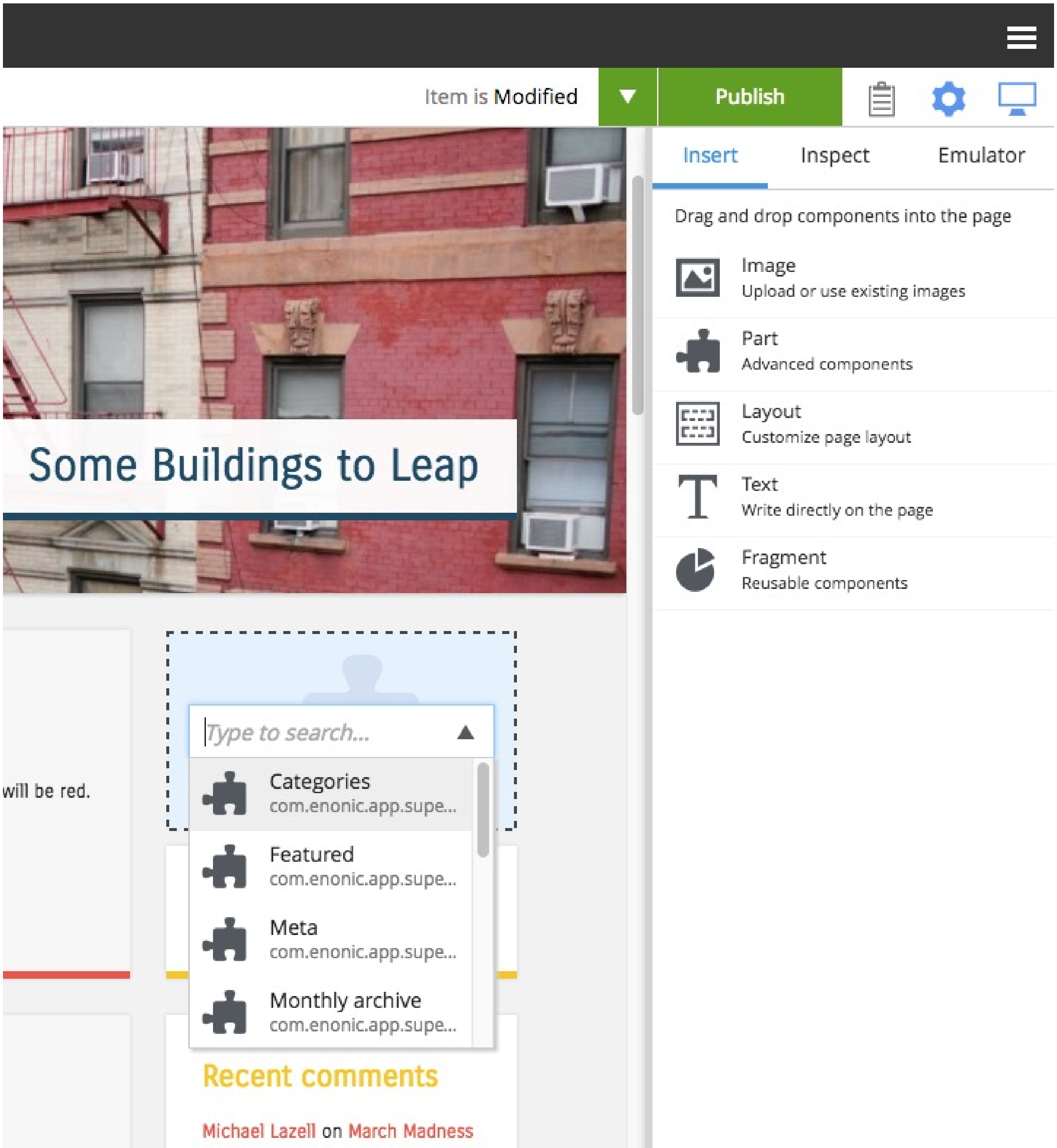
Part component

Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Part components are reusable, configurable components that can be placed into any region of a page with the page editor *Inspection Panel*. This allows content editors to build and customize pages without writing any code. There are no built-in part components. Each one is custom made in the application code. Parts are typically created to render custom content, lists of content, forms, etc.

The first step in adding a part component to a page is to edit the page content and open the inspection panel's "Insert" tab. Drag the part component placeholder (puzzle piece) to the desired location on the page. The part placeholder will now appear as a blue box with a dropdown selector. The same part dropdown selector will appear in the inspect panel. Use one of the selectors to find the desired part component. Once a part component is selected, the placeholder will be replaced with the actual part and the Inspect panel will show the part's configuration options in a form.

Some parts won't have any configuration. Parts with configuration options are independently configured. This means that the same part component can be added to multiple pages, or even multiple times in the same page, and each instance can have different configuration values.



Drag and drop components into the page

- Image — Upload or use existing images
- Part — Advanced components
- Layout — Customize page layout
- Text — Write directly on the page
- Fragment — Reusable components

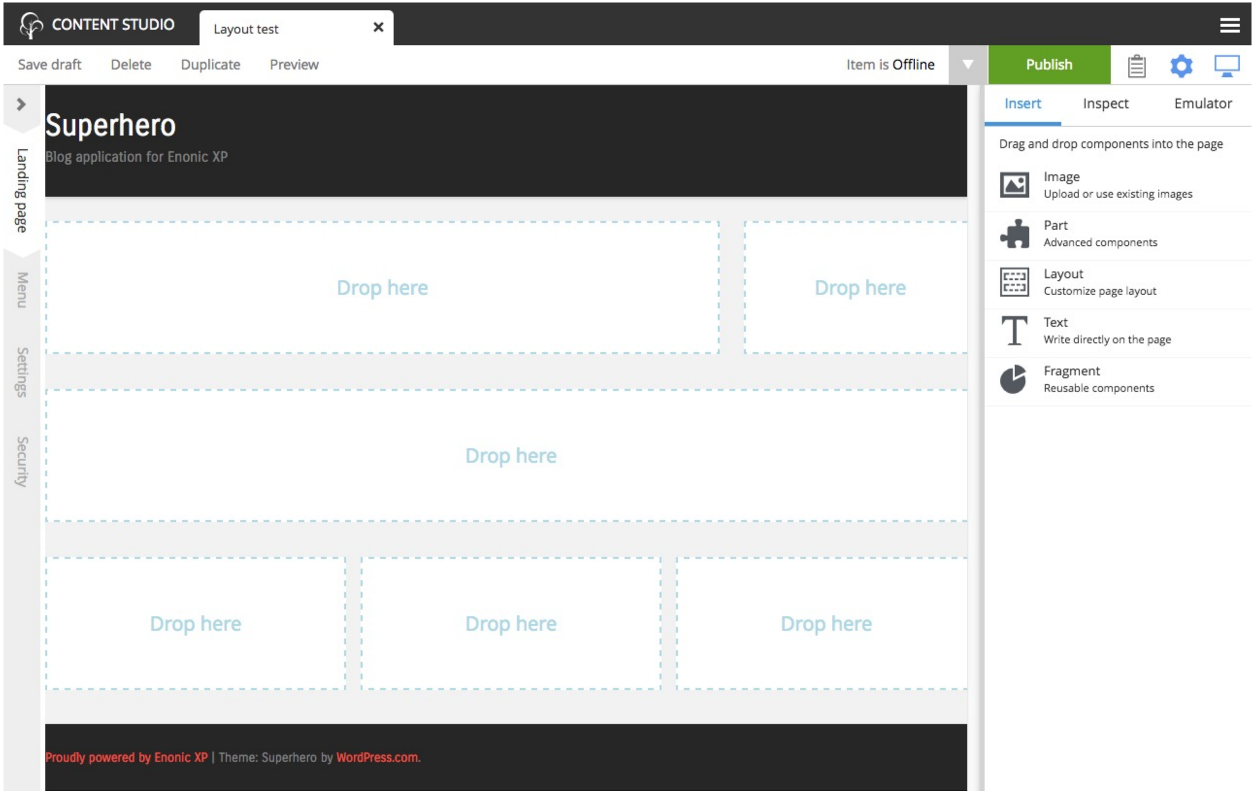
Categories — com.enonic.app.supe...
Featured — com.enonic.app.supe...
Meta — com.enonic.app.supe...
Monthly archive — com.enonic.app.supe...

Layout component

Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Layout components are reusable, configurable components (similar to Part components) that can be placed into any region defined in a page component. Layouts themselves define regions where other components can be placed with the *Page Editor panel*. The primary purpose of a layout is to enable other components to be placed side-by-side. As of version 6.8, a layout cannot be placed inside another layout. There are no built-in layouts. Each one is custom made in the application code. Layouts are typically created for two or three columns and have configuration options for column widths.

In the *Page Editor panel*, drag a Layout component placeholder from the inspect panel to the desired region on the page. The layout placeholder will now appear as a blue box with a dropdown selector. The same dropdown selector will appear in the inspect panel. Use one of the selectors to find the desired layout component. Once a layout is selected, the actual layout rendering will replace the placeholder and its configuration options will appear in the inspect panel. Some layout components may not have any configuration options.



While editing a page, it may be difficult to select a layout to access its configuration. In this case, the *Component View* can help to select the layout. Alternatively, a part within the layout can be selected and then that part's parent can be selected from the right-click context menu. Continue selecting the parent component until the layout is the selected component.

Text component

Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The Text component allows content editors to place and format text into any region on a page without writing any code. Images can also be added inside text components. Macros allow Twitter tweets, YouTube videos, embedded code, and no-format text to be added as well. The formatting and macro options are the same as those for the HtmlArea inputs that can be found in content types and other configuration forms in the Content Studio. The only difference is that the formatting toolbar is at the top of the page for text components.

In the *Page Editor panel*, drag a Text component from the inspect panel to the desired region on the page. A cursor will appear inside the text component and editing can begin. If another component is selected, the text component will need to be double-clicked to resume editing.

Fragment component

Note: This section is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

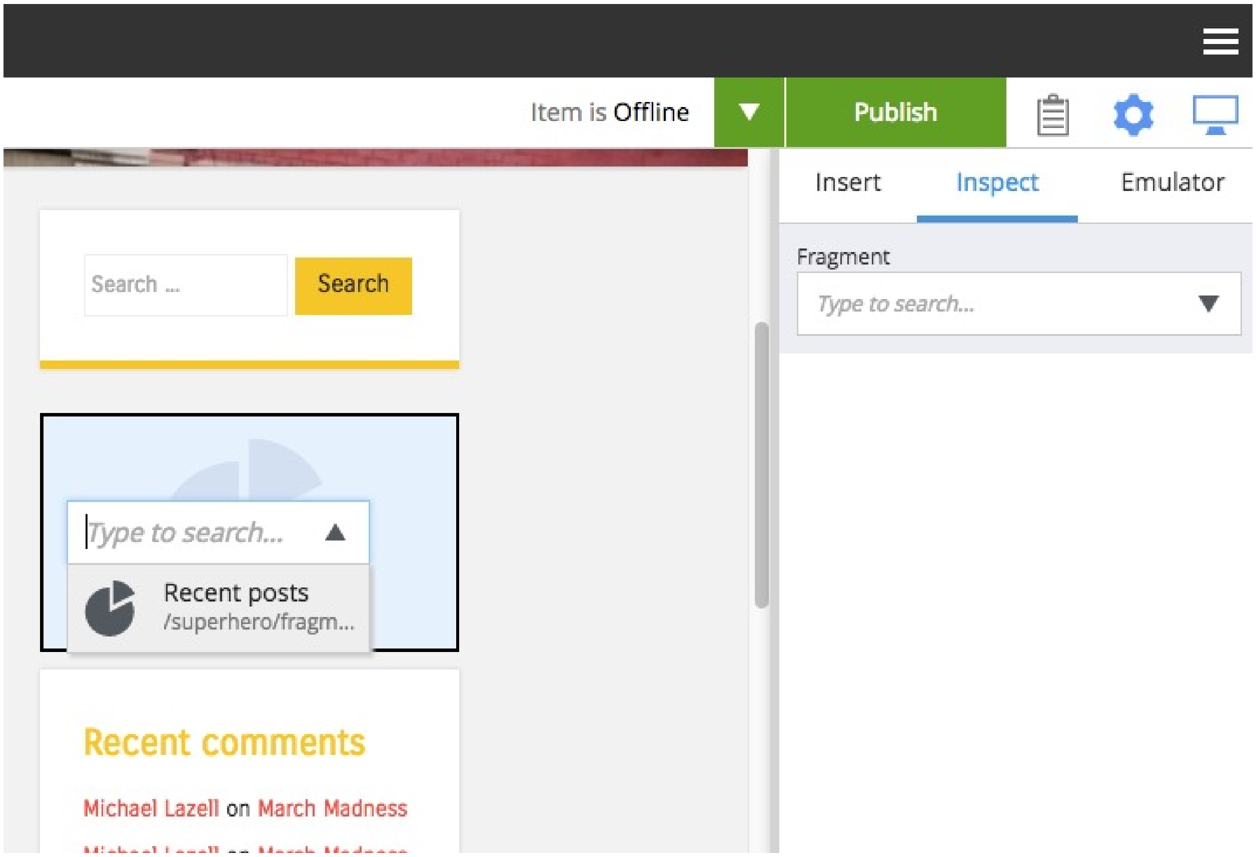
Fragments are created as content from an instance of another component. What makes a fragment special is that it uses the same configuration on every page where it's added. When a fragment content is altered, the change is instantly visible on every page that uses it. All of the other components are independently configured.

Creating fragments

Fragments can be created from any component on a page. When a fragment is created, it makes a content copy of the part, layout, image or text component. In the page editor, right-click the desired component and select "Create fragment" from the context menu. The new fragment content is created as a child of the page being edited. The fragment content will open in a new editor tab where its name and configuration can be changed. At the same time, the component that was copied is replaced with the new fragment.

Using fragments

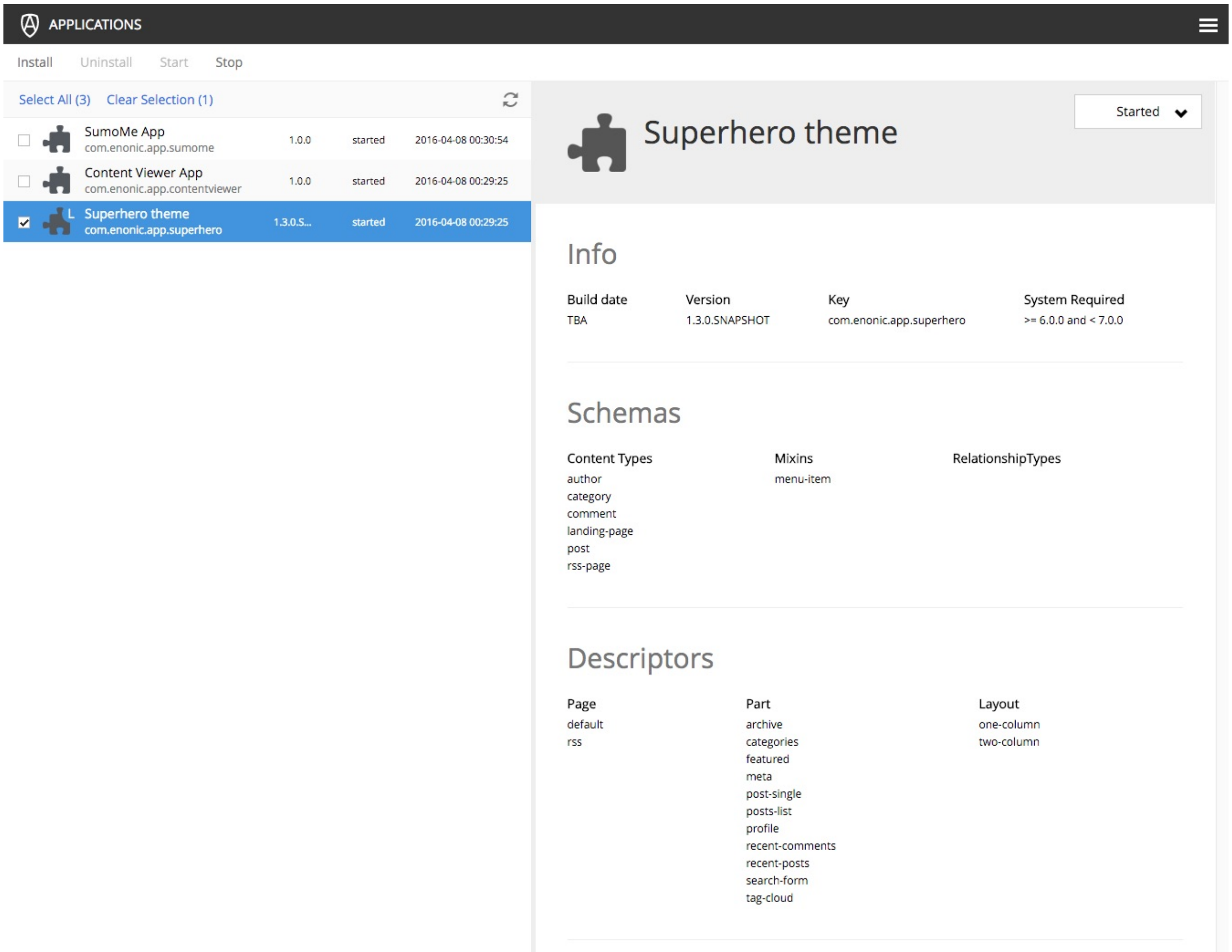
Once a fragment content has been created, it can be added to pages with the page editor. Drag a fragment placeholder from the "Insert" tab of the *Inspection Panel* to the desired location on the page. Use the dropdown selector in the placeholder to find the desired fragment content. Once selected, the fragment will appear.



Applications

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The Applications admin tool provides an interface to install, uninstall, start and stop applications for an Enonic XP installation. A list of the installed applications appears in the left panel. Applications in this list can be selected by clicking on them. Information about a selected app appears in the right panel. This information includes the basics such as version, key and system requirements as well as a list of schemas for content types, mixins and relationship types, and also a list of descriptors, which are components for pages, parts and layouts.



□	SumoMe App com.enonic.app.sumome	1.0.0	started	2016-04-08 00:30:54
□	Content Viewer App com.enonic.app.contentviewer	1.0.0	started	2016-04-08 00:29:25
☑	Superhero theme com.enonic.app.superhero	1.3.0.S...	started	2016-04-08 00:29:25

Superhero theme

Info

Build date	Version	Key	System Required
TBA	1.3.0.SNAPSHOT	com.enonic.app.superhero	>= 6.0.0 and < 7.0.0

Schemas

Content Types	Mixins	RelationshipTypes
author category comment landing-page post rss-page	menu-item	

Descriptors

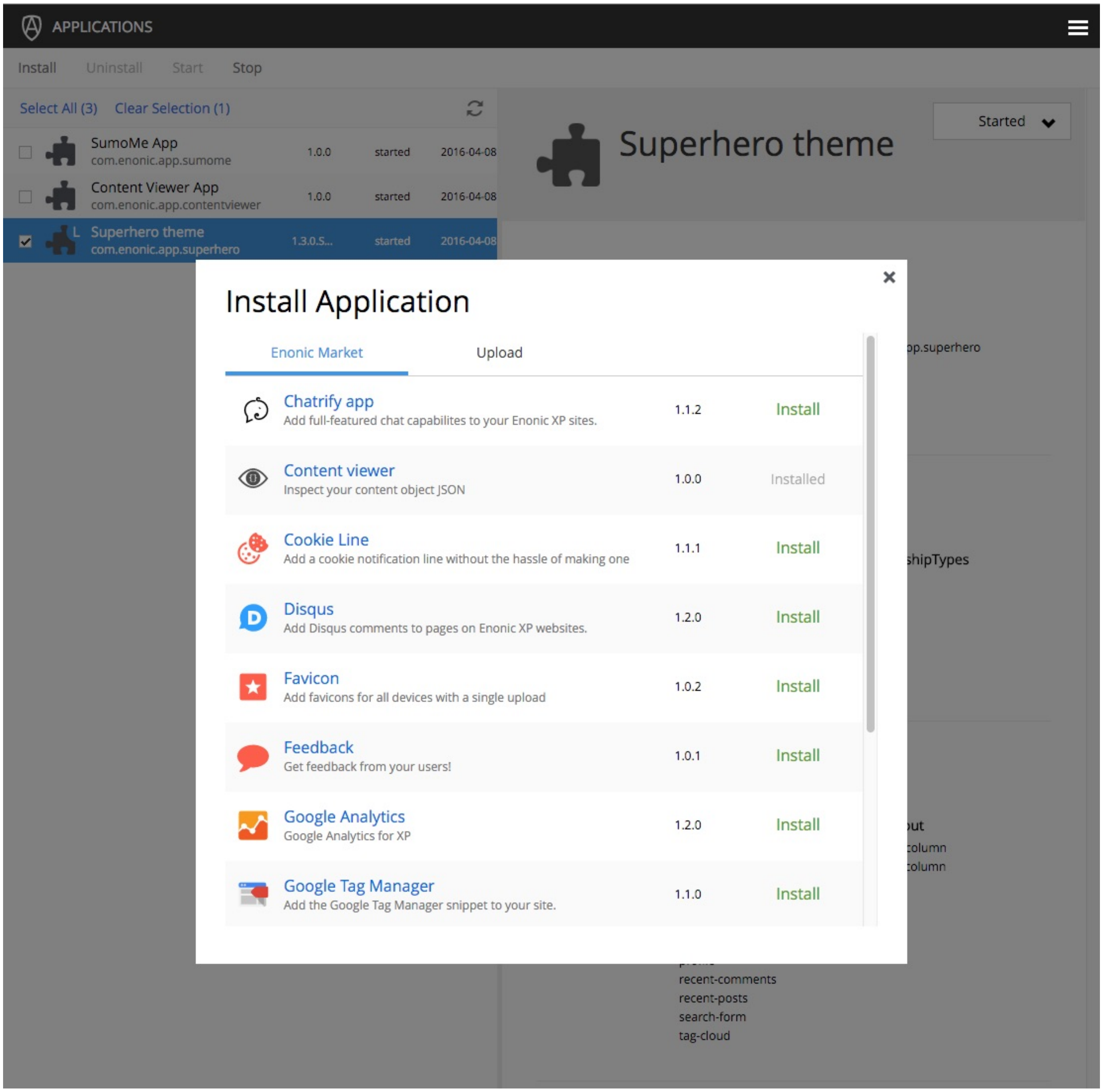
Page	Part	Layout
default rss	archive categories featured meta post-single posts-list profile recent-comments recent-posts search-form tag-cloud	one-column two-column

Installing an app

Two methods of installing apps are available with the Applications tool. Both are initiated by clicking the "Install" button in the toolbar. A modal window appears with tabs labeled "Enonic Market" and "Upload".

Enonic Market install

A list of apps on the Enonic Market will appear under the "Enonic Market" tab. Each app in the list has a name, a brief description, version number and an "Install" link that will download, install and start the app. Installation will occur on all nodes of a clustered environment. Clicking the name of an app will open a page on the Enonic Market website with more information about the app.



SumoMe App com.enonic.app.sumome	1.0.0	started	2016-04-08
Content Viewer App com.enonic.app.contentviewer	1.0.0	started	2016-04-08
☑ Superhero theme com.enonic.app.superhero	1.3.0.S...	started	2016-04-08

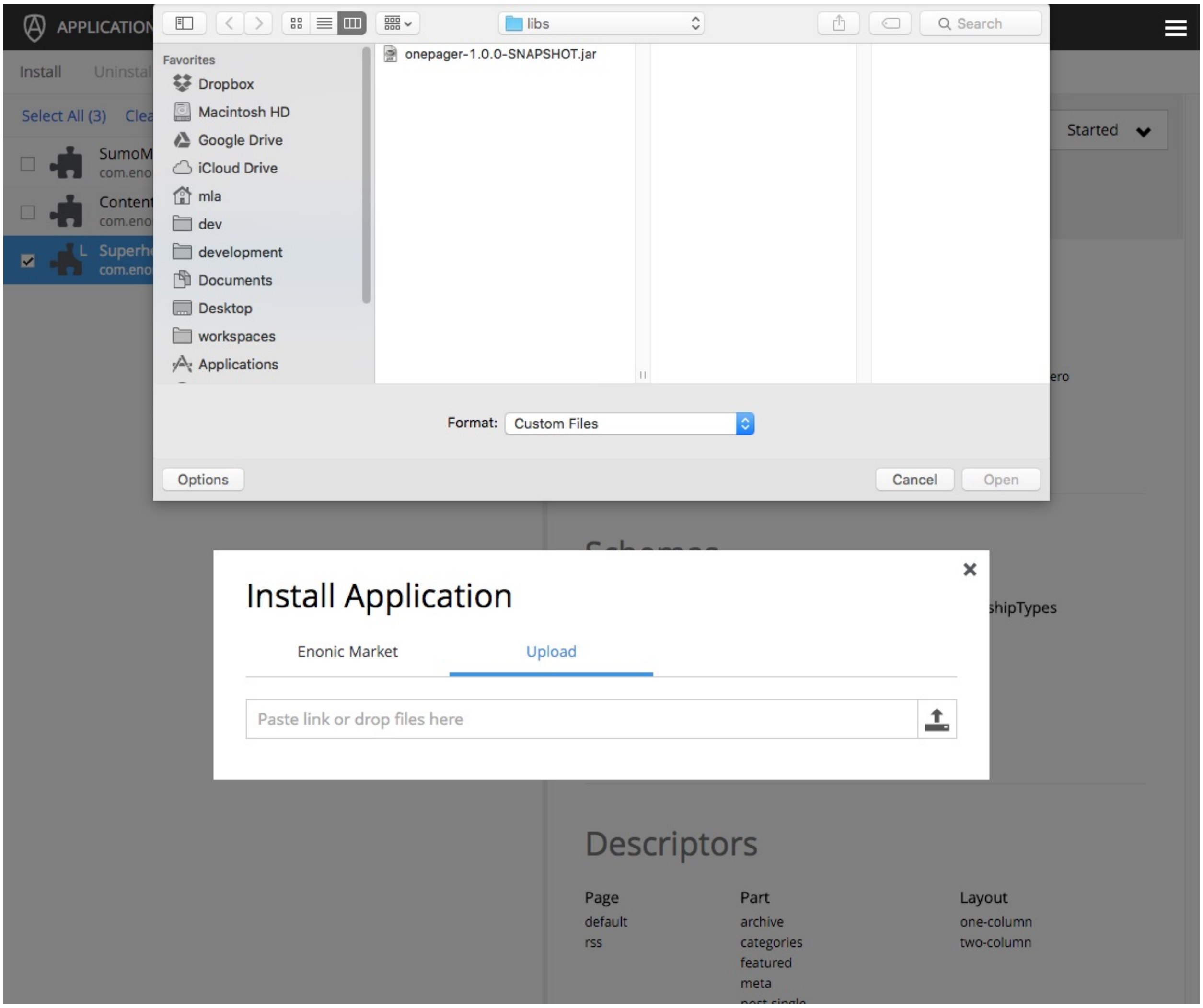
Install Application

Enonic Market | Upload

Chatrify app Add full-featured chat capabilites to your Enonic XP sites.	1.1.2	Install
Content viewer Inspect your content object JSON	1.0.0	Installed
Cookie Line Add a cookie notification line without the hassle of making one	1.1.1	Install
Disqus Add Disqus comments to pages on Enonic XP websites.	1.2.0	Install
Favicon Add favicons for all devices with a single upload	1.0.2	Install
Feedback Get feedback from your users!	1.0.1	Install
Google Analytics Google Analytics for XP	1.2.0	Install
Google Tag Manager Add the Google Tag Manager snippet to your site.	1.1.0	Install

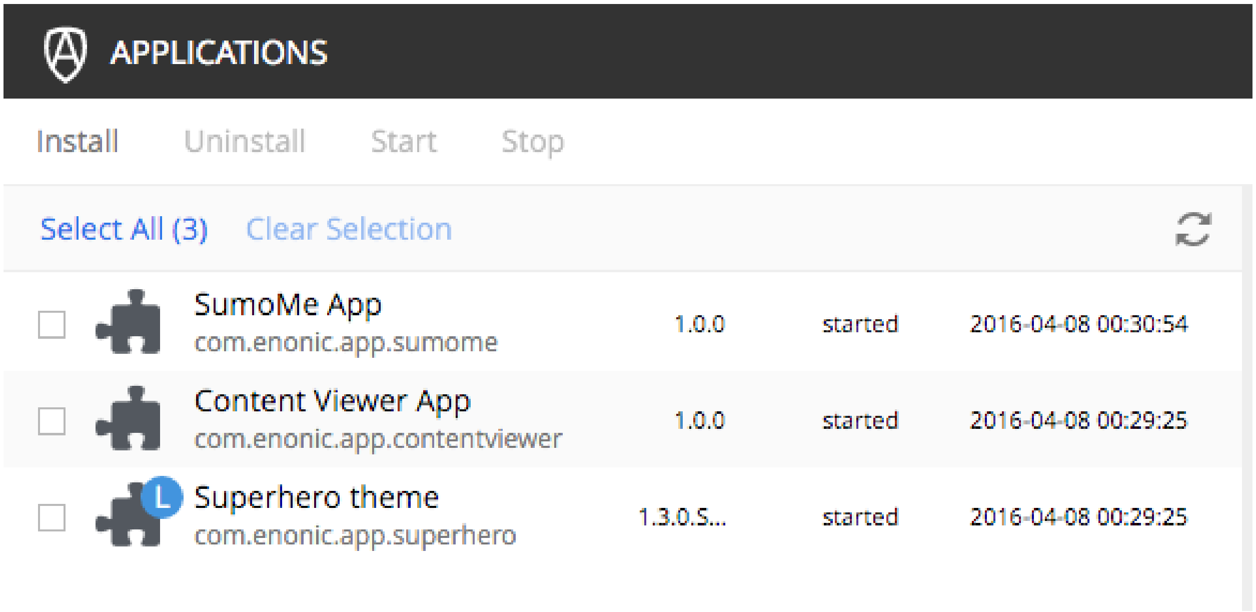
Upload file install

Applications can also be installed by clicking the "Upload" tab in the "Install Application" modal. This shows a file input where you can drag and drop an application JAR file, or click the upload button on the right and select an application JAR file from your local filesystem. The app will be installed and started automatically when the download is complete.



Manual installation

A third method of installing apps involves placing the application JAR file into the `$XP_HOME/deploy` folder. It is not possible to do this with the Applications admin tool. Apps installed this way would require the JAR file to be placed in the `$XP_HOME/deploy` folder on each node in a cluster. Locally installed apps will have a blue circle with an “L” on the app icon in the list of installed apps. The Superhero theme app was installed locally in the image below. Locally installed apps can only be uninstalled by deleting the JAR file in the `$XP_HOME/deploy` folder.



□	SumoMe App com.enonic.app.sumome	1.0.0	started	2016-04-08 00:30:54
□	Content Viewer App com.enonic.app.contentviewer	1.0.0	started	2016-04-08 00:29:25
□	Superhero theme com.enonic.app.superhero	1.3.0.S...	started	2016-04-08 00:29:25

Users

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

The Users tool allows the management of userstores, users, roles, and groups. A basic understanding of Enonic XP identity management is essential for the proper management of your applications.

Userstores

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

All users and groups are created and managed in user stores. Each Enonic XP installation has a System User Store that cannot be deleted. Additional user stores can be created as needed. For example, it might be convenient to use the System User Store for employees who run the website and another user store for customers who log into the public site. Each user store can assign an ID Provider to handle its authentication (see *ID Providers*).

Creating user stores

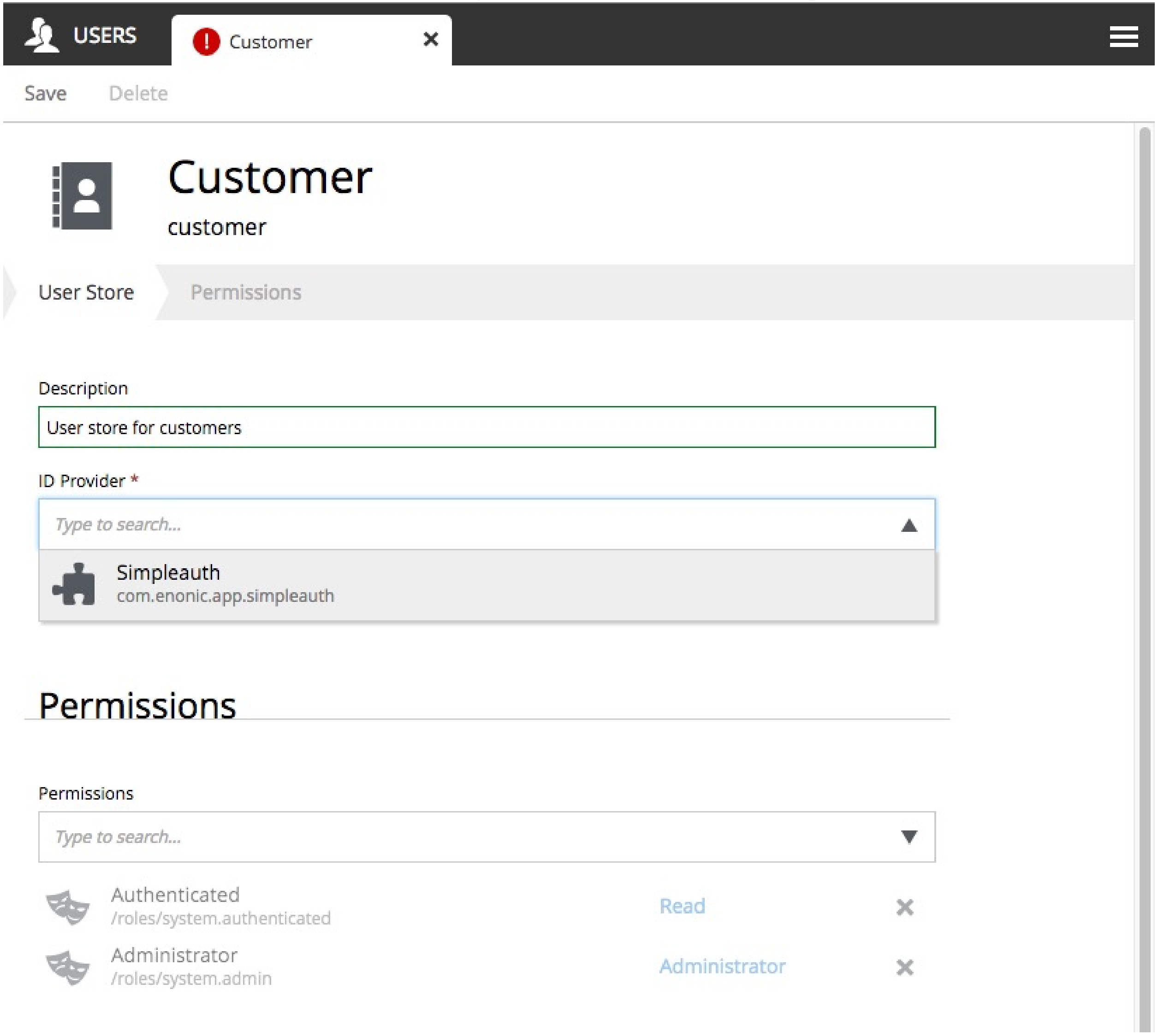
To create a new user store, click "New" in the toolbar while no items are selected in the tree view. This will open an editor tab where the user store details can be entered as described below.

As the `<DisplayName>` is entered, the `<name>` field is automatically filled in with a URL friendly version. The `<name>` field will be part of the user store key and it cannot be changed once the user store is saved.

The ID Provider is for assigning an app that will handle authentication for the user store. ID providers can have their own configuration which can be viewed and modified by clicking the pencil icon.

Finally, the permissions section allows principals (users, roles and groups) to be added to the user store. The Authenticated and Administrator roles are automatically added and they cannot be altered or removed.

When a user store is created, it will automatically have folders for Users and Groups.



Customer

customer

Description: User store for customers

ID Provider *: Type to search...

Simpleauth
com.enonic.app.simpleauth

Permissions

Permissions: Type to search...

Authenticated /roles/system.authenticated	Read	✕
Administrator /roles/system.admin	Administrator	✕

Users

The System User Store has two built-in users. One is the Super User which has full administrative permissions. The other is the Anonymous User which is the principal used by any site visitor that is not logged in.

Creating users

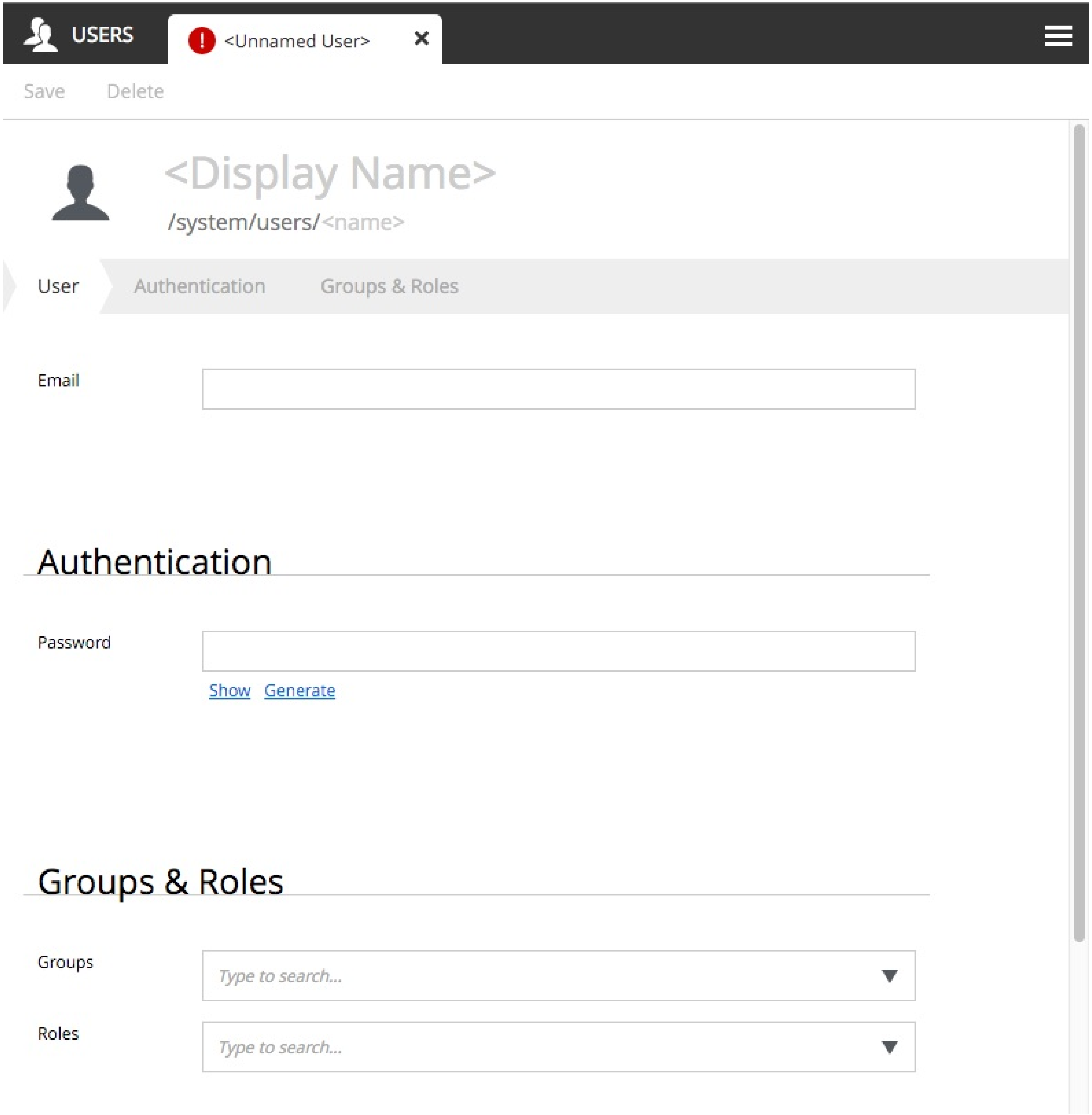
Additional users may be added by right-clicking the "Users" folder and selecting "New" in the context menu. This opens the User editor in a new tab within the page. All the fields are required except for the "Groups & Roles" and the user cannot be "Saved" until the required fields are filled in.

The `<Display Name>` field will typically be the user's first and last name with capital letters and a space. The `<name>` field will be automatically filled in with a URL-friendly version of the `<Display Name>`. The `<name>`

will be used to log in and it can be manually edited at this time. The `<name>` cannot be changed once the entry is saved. The Email field must have a valid format and must be unique per user store. The Password field has buttons to show/hide the characters and the "Generate" button will create a random password for you. The password's strength will be displayed as you type and this ranges from "weak" to "extreme". Passwords cannot be displayed once the entry is saved.

Groups and Roles can be added to the user now or while editing the entry later.

Once the required fields have valid values, the red exclamation mark in the tab goes away and the user entry can be saved by clicking the button in the toolbar.

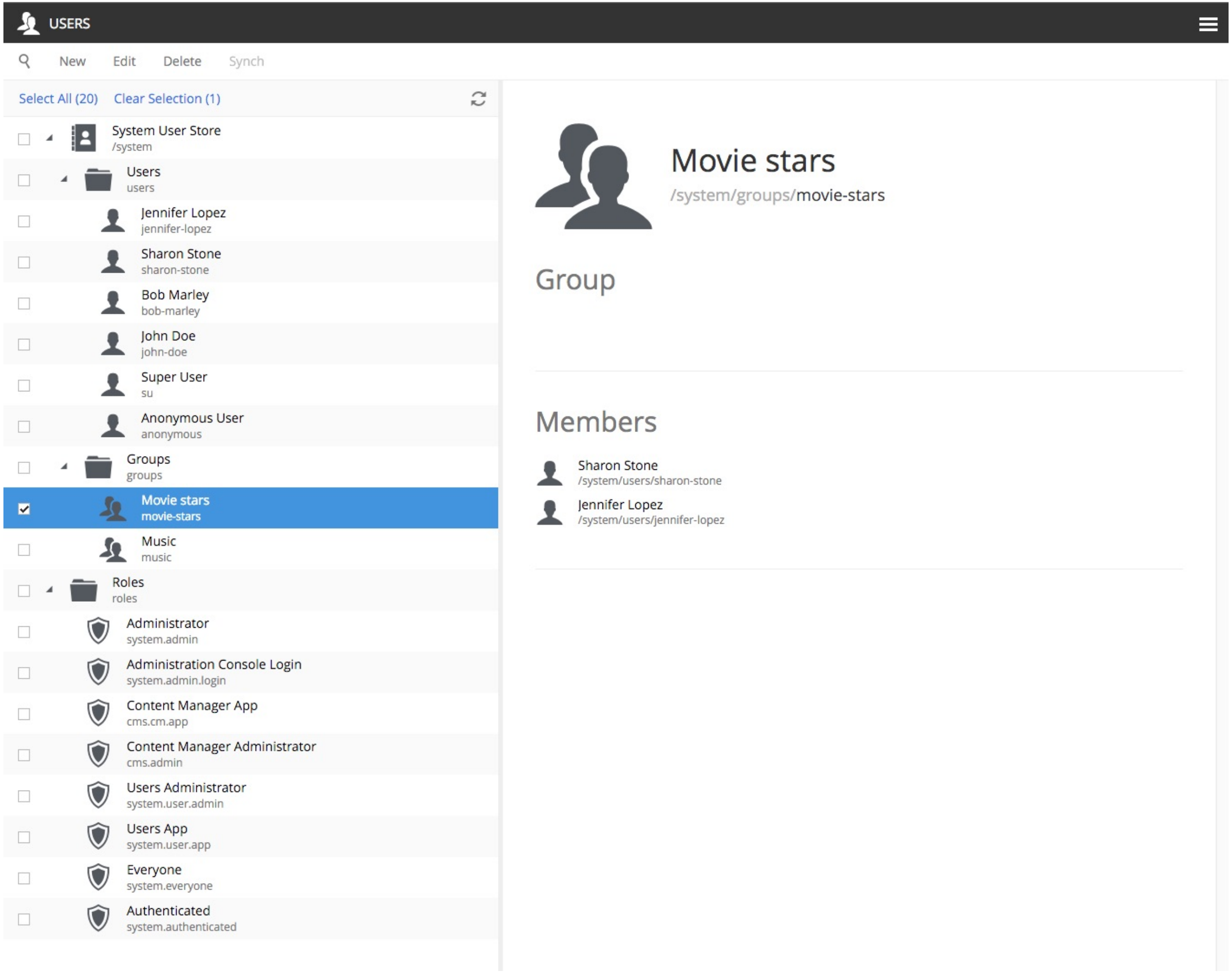


Groups

Groups assist with managing user permissions for content. For example, all content has security permissions which may include roles, groups and users. If a content has only a group named “Customers” (with read access) then only logged in members of that group can see the content. Groups have no function without Members. Users and even other groups may be added as members. Clicking on a group in the Users admin tool will show the group’s display name, principal path, and a list of its members.

Creating groups

Right-click the “Groups” folder in the desired user store and select “New” in the context menu. The `<Display Name>` is what will be listed in the “Groups” folder. The `<name>` is automatically generated as a URL-friendly version of the `<Display Name>` and should not be changed. The Description is optional. Users and other groups can be added to the group as “Members”. Users can also be added to a group by editing the user.



□ System User Store /system
□ Users users
□ Jennifer Lopez jennifer-lopez
□ Sharon Stone sharon-stone
□ Bob Marley bob-marley
□ John Doe john-doe
□ Super User su
□ Anonymous User anonymous
□ Groups groups
☑ Movie stars movie-stars
□ Music music
□ Roles roles
□ Administrator system.admin
□ Administration Console Login system.admin.login
□ Content Manager App cms.cm.app
□ Content Manager Administrator cms.admin
□ Users Administrator system.user.admin
□ Users App system.user.app
□ Everyone system.everyone
□ Authenticated system.authenticated

Movie stars
/system/groups/movie-stars

Group

Members

Sharon Stone /system/users/sharon-stone
Jennifer Lopez /system/users/jennifer-lopez

Roles

Note: This page is under construction. This information is likely incomplete and possibly inaccurate until this notice is removed.

Roles grant certain permissions to the users that have them. New roles can be created by administrators. The built-in roles are listed and described below.

Administrator

Users with the Administrator role have full access to all content and admin tools through the user interface.

Administration Console Login

Users with this role can log in to the administration console. These users will also require a role for each of the admin tools that the users need access to.

Content Manager App

This role allows users to access the Content Studio. Users can see content and sites, but cannot save changes or publish content.

Content Manager Administrator

This role allows full access to the Content Studio admin tool, including saving edits and publishing content.

Users App

This role allows view-only access to the Users admin tool.

Users Administrator

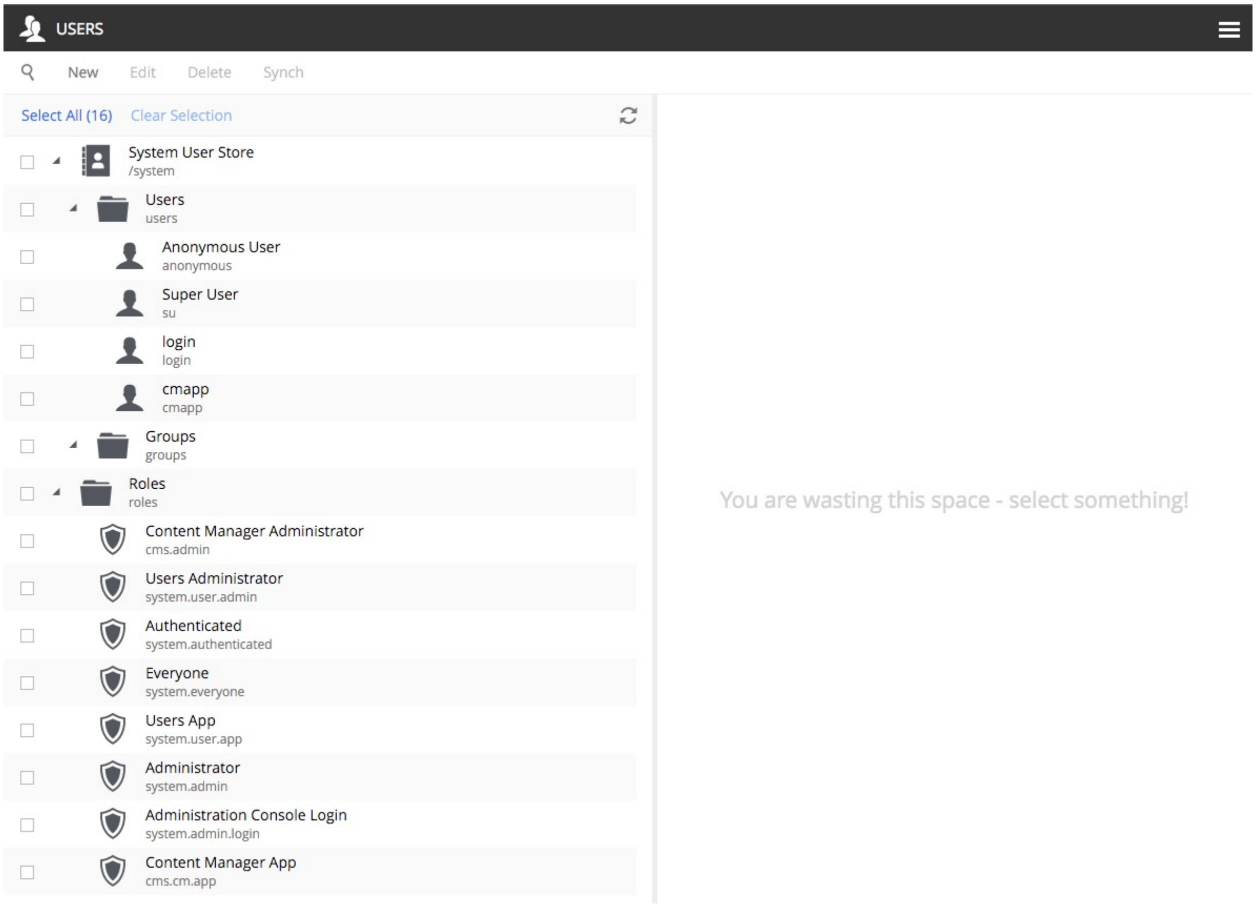
This role allows full access to the Users admin tool, including create/edit/delete for userstores, users, roles, and groups.

Authenticated

Users automatically have this role when they are logged into the system in any way.

Everyone

This is a special role that all users and site visitors have. Content with this role will be publicly available to any non-logged in visitor.



Name	Path
System User Store	/system
Users	users
Anonymous User	anonymous
Super User	su
login	login
cmapp	cmapp
Groups	groups
Roles	roles
Content Manager Administrator	cms.admin
Users Administrator	system.user.admin
Authenticated	system.authenticated
Everyone	system.everyone
Users App	system.user.app
Administrator	system.admin
Administration Console Login	system.admin.login
Content Manager App	cms.cm.app

You are wasting this space - select something!

CHAPTER 6

API and Reference Guide

```
if (idea) {
     enonicExperiencePlatform();
}
else {
     drinkCoffee();
     getIdea();
}
```

```
dependencies {
  include 'com.enonic.xp:<name>:6.8.1'
}
```

Where name is the name of the library. Here's a list of available libraries:

- lib-auth
- lib-cache
- lib-content
- lib-context
- lib-http-client
- lib-i18n
- lib-io
- lib-mail
- lib-mustache
- lib-portal
- lib-task
- lib-thymeleaf
- lib-xslt
- lib-websocket

To include both lib-mail and lib-content you can add both inside the dependency list like this:

```
dependencies {
  include 'com.enonic.xp:lib-mail:6.8.1'
  include 'com.enonic.xp:lib-content:6.8.1'
}
```

Note: The server-side JavaScript reference documentation can be accessed using the following links:

- Read in your browser
- Download as zip

View Functions

Some view technologies support a set of view functions. The list of view functions below are supported out-of-the-box:

assetUrl

This generates a URL pointing to a static file in the site/assets folder, such as CSS, background images, etc.

Parameters:

_path Path to the asset.

_application Use this when the asset referenced is in another application. Defaults to current application. Use the app name, for example, `com.enonic.blog.superhero`.

_type URL type. Either `server` (server-relative URL) or `absolute`. Default is `server`.

everything else Custom parameters to append to the url.

Usage in Thymeleaf:

```
<a data-th-href="${portal.assetUrl({'_path=css/main.css'})}">Link</a>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
    <xsl:value-of select="portal:assetUrl('_path=a')"/>
  </xsl:template>

</xsl:stylesheet>
```

attachmentUrl

This generates a URL pointing to an attachment.

Parameters:

_id Id to the content holding the attachment.

_path Path to the content holding the attachment.

_name Name to the attachment.

_label Label of the attachment. Default is `source`.

_download Set to true if the disposition header should be set to attachment. Default is `false`.

_type URL type. Either `server` (server-relative URL) or `absolute`. Default is `server`.

everything else Custom parameters to append to the url.

Usage in Thymeleaf:

```
<a data-th-href="${portal.attachmentUrl({'_download=true'})}">Link</a>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
    <xsl:value-of select="portal:attachmentUrl('_download=true')"/>
  </xsl:template>

</xsl:stylesheet>
```

componentUrl

This generates a URL pointing to a component.

Parameters:

_id Id to the page.

_path Path to the page.

_component Path to the component. If not set, the current path is set.

_type URL type. Either `server` (server-relative URL) or `absolute`. Default is `server`.

everything else Custom parameters to append to the url.

Usage in Thymeleaf:

```
<a data-th-href="${portal.componentUrl({'_component=main/1'})}">Link</a>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
    <xsl:value-of select="portal:componentUrl('_component=main/1')"/>
  </xsl:template>

</xsl:stylesheet>
```

imageUrl

This generates a URL pointing to an image.

Parameters:

_id Id to the image.

_path Path to the image. If `_id` is specified, this parameter is not used.

_format Format of the image.

_scale Resize and crop the image to fit the available area. See: *Scaling*

_quality Quality for JPEG images, ranges from 0 (max compression) to 100 (min compression). Default is 85.

_background Background color.

_filter Styling filters to use on the image. More than one filter may be combined with a semicolon. See: *Styling*

_type URL type. Either `server` (server-relative URL) or `absolute`. Default is `server`.

everything else Custom parameters to append to the url.

Usage in Thymeleaf:

```
<img data-th-src="${portal.imageUrl({'_id=11', '_scale=width(200)'})}"/>
<img data-th-src="${portal.imageUrl({'_path=test', '_scale=width(200)'})}"/>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
    <xsl:value-of select="portal:imageUrl('_id=11', 'scale=width(200)')"/>
    <xsl:value-of select="portal:imageUrl('_path=test', 'scale=width(200)')"/>
  </xsl:template>

</xsl:stylesheet>
```

pageUrl

This generates a URL pointing to a page.

Parameters:

_id Id to the page. If id is set, then path is not used.

_path Path to the page. Relative paths is resolved using the context page.

_type URL type. Either `server` (server-relative URL) or `absolute`. Default is `server`.

everything else Custom parameters to append to the url.

Usage in Thymeleaf:

```
<a data-th-href="${portal.pageUrl({'_path=/my/page', 'a=3'})}">Link</a>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
    <xsl:value-of select="portal:pageUrl('_path=/my/page', 'a=3')"/>
  </xsl:template>

</xsl:stylesheet>
```

serviceUrl

This generates a URL pointing to a service.

Parameters:

_service Name of the service.

_application Other application to reference to. Default is current application.

_type URL type. Either `server` (server-relative URL) or `absolute`. Default is `server`.

everything else Custom parameters to append to the url.

Usage in Thymeleaf:

```
<a data-th-href="${portal.serviceUrl({'_service=myservice', 'a=3'})}">Link</a>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
    <xsl:value-of select="portal:serviceUrl('_service=myservice', 'a=3')"/>
  </xsl:template>

</xsl:stylesheet>
```

imagePlaceholder

This command generates a URL to an image placeholder.

Parameters:

width Width of image.

height Height of image.

Usage in Thymeleaf:

```
<img data-th-src="${portal.imagePlaceholder({'width=10','height=10'})}"/>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
    <xsl:value-of select="portal:imagePlaceholder('width=10','height=10')"/>
  </xsl:template>

</xsl:stylesheet>
```

localize

This localizes a phrase.

Parameters:

_key The property key.

_locale A string-representation of a locale. If the locale is not set, the site language is used.

_values Optional placeholder values (comma separated).

Usage in Thymeleaf:

```
<div data-th-text="${portal.localize({'_key=mystring','_locale=en'})}">Not translated</div>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
```

```
    <xsl:value-of select="portal:localize('_key=mystring')"/>
  </xsl:template>

</xsl:stylesheet>
```

processHtml

This function replaces abstract internal links contained in an HTML text by generated URLs.

Parameters:

_value Html value string to process.

_type URL type. Either "server" (server-relative URL) or "absolute". Default is "server"

Usage in Thymeleaf:

```
<div data-th-text="${portal.processHtml({'_value=some text'})}">Text</div>
```

Usage in XSLT:

```
<xsl:stylesheet version="2.0" exclude-result-prefixes="#all"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:portal="urn:enonic:xp:portal:1.0">

  <xsl:template match="/">
    <xsl:value-of select="portal:processHtml('_value=some text')"/>
  </xsl:template>

</xsl:stylesheet>
```

Query Language

When finding nodes and content you will be using our query language. It is based on SQL and looks very similar.

queryExr

Grammar:

```
queryExpr = [ constraintExpr ] [ orderExpr ] ;
```

- If no constraint-expression is given, all documents will match.
- If no order-expression is given, results will be ordered by *_score* descending.

Examples:

```
myCategory = 'article'
myCategory = 'article' ORDER BY title DESC
ORDER BY title
```

constraintExpr

Grammar:

```
constraintExpr = compareExpr
               | logicalExpr
               | dynamicConstraint
               | notExpr ;
```

compareExpr

Grammar:

```
compareExpr   = fieldExpr operator valueExpr ;
fieldExpr     = propertyPath ;
operator      = '=', '!=', '>', '>', '<', '<=', 'LIKE', 'NOT LIKE', 'IN', 'NOT IN' ;
valueExpr     = string | number | valueFunc ;
valueFunc     = geoPoint | instant | time | dateTime, localDateTime ;
geoPoint      =  '"' lat ',' lon '"' ;
instant       = 'instant(' string ')' ;
time          = 'time(' string ')' ;
dateTime      = 'dateTime(' string ')' ;
localDateTime = 'localDateTime(' string ')' ;
```

Examples:

```
user.myCategory = "articles"
user.myCategory IN ("articles", "documents")
user.myCategory != "articles"
user.myCategory LIKE "*tic*"
myPriority < 10
myPriority <= 10
myPriority > 10
myPriority < 100
myPriority != 10
myInstant = instant('2014-02-26T14:52:30.00Z')
myInstant <= instant('2014-02-26T14:52:30.00Z')
myInstant <= dateTime('2014-02-26T14:52:30.00+02:00')
myTime = time('09:00')
myLocalDateTime = time('2014-02-26T14:52:30.00')
myLocation = '59.9127300,10.7460900'
myLocation IN ('59.9127300,10.7460900','59.2181000,10.9298000')
```

logicalExpr

Grammar:

```
logicalExpr = constraintExpr operator constraintExpr ;
operator    = 'AND' | 'OR' ;
```

Examples:

```
myCategory = "articles" AND myPriority > 10
myCategory IN ("articles", "documents") OR myPriority <= 10
```

dynamicConstraint

Grammar:

```
dynamicConstraint = functionExpr ;
```

Examples:

```
fulltext('myCategory', 'Searching for fish', 'AND')
ngram('description', 'fish boat', 'AND')
```

notExpr

Grammar:

```
notExpr = 'NOT' constraintExpr ;
```

Examples:

```
NOT myCategory = 'article'
```

orderExpr

Grammar:

```
orderExpr = 'ORDER BY' ( fieldOrderExpr | dynamicOrderExpr )
             ( ',' ( fieldOrderExpr | dynamicOrderExpr ) )* ;
```

fieldOrderExpr

Grammar:

```
fieldOrderExpr = propertyPath [ direction ] ;
direction>     = 'ASC' | 'DESC' ;
```

Examples:

```
_name ASC
_timestamp DESC
title DESC
data.myProperty
```

dynamicOrderExpr

Grammar:

```
dynamicOrderExpr = functionExpr [ direction ] ;
direction        = 'ASC' | 'DESC' ;
```

Examples:

```
geoDistance('59.9127300,10.746090')
```

propertyPath

Grammar:

```
propertyPath = pathElement ( '.' pathElement )* ;
pathElement  = ( [ validJavaIdentifier - '.' ] )* ;
```

Examples:

```
myProperty
data.myProperty
data.myCategory.myProperty
```

Tip: Wildcards in propertyPaths are supported in functions `fulltext` and `ngram` only at the moment. When using these functions, expressions like this are valid:

```
myProp*
*Property
data.*
*.myProperty
data.*.myProperty
```

functionExpr

Grammar:

```
functionExpr = functionName '(' arguments ')' ;
```

Examples

Find all documents where property ‘myCategory’ is populated with a value, and the value does not equal ‘article’.

```
myCategory LIKE '*' AND NOT myCategory = 'article'
```

Find all document where property ‘myCategory’ is either ‘article’ or ‘document’ and title starts with ‘fish’.

```
myCategory IN ('article', 'document') AND ngram('title', 'fish', 'AND')
```

Find all documents where any fulltext-analyzed property contains ‘fish’ and ‘spot’, and order them ascending by distance from Oslo.

```
fulltext('_allText', 'fish spot', 'AND') ORDER BY
geoDistance('data.location', '59.9127300,10.7460900') ASC
```

Find all documents where any property under data-set ‘data’ contains ‘fish’ and ‘spot’, and order them ascending by distance from Oslo.

```
fulltext('data.*', 'fish spot', 'AND') ORDER BY
geoDistance('data.location', '59.9127300,10.7460900') ASC
```

Toolbox CLI

The toolbox is a CLI (command line interface) tool that is used to do administration tasks. Toolbox executables are located in `$XP_INSTALL/toolbox` folder. Use `toolbox.sh` for mac/unix environments and `toolbox.bat` for windows environments.

To get help for the commands, just type the following:

```
$ ./toolbox.sh
usage: toolbox <command> [<args>]

The most commonly used toolbox commands are:
        delete-snapshots   Deletes snapshots, either before a given timestamp or by name.
        dump               Export data from every repository.
        export             Export data for a specified path.
        help               Display help information
        import             Import data from a named export.
        init-project       Initiates an Enonic XP application project.
        install-app        Install an application from URL or file
        list-snapshots     Returns a list of existing snapshots with name and status.
        load               Import data from a dump.
        reindex            Reindex content in search indices for the given repository and branches.
        reprocess          Reprocesses content in the repository.
        restore            Restores a snapshot of a previous state of the repository.
        set-replicas       Set the number of replicas in the cluster.
        snapshot           Stores a snapshot of the current state of the repository.
        upgrade            Upgrade a dump to the current version. The upgraded files will be written

        See 'toolbox help <command>' for more information on a specific command.
```

To get help for a specific command, you can type `toolbox.sh help <command>`, like:

```
$ toolbox.sh help import
```

Here's a list of all the commands that you can do with the toolbox:

snapshot

Create a snapshot of all or a single repository while running. The snapshots will be stored in the directory given in `snapshots.dir` option in the *Storage Configuration* (default `$xp_home/snapshots`). Note that the first snapshot only stores markers in the repository for the current state. Subsequent snapshots stores the changes since the last snapshot. See *Backup and Restore* for more information on snapshots.

Attention: For a clustered installation, the snapshot-location must be on a shared file-system.

Usage:

```
NAME
        toolbox snapshot - Stores a snapshot of the current state of the
        repository.

SYNOPSIS
        toolbox snapshot -a <auth> [-h <host>] [-p <port>] [-r <repository>]

OPTIONS
        -a <auth>
```

```
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).

        -r <repository>
            the name of the repository to snapshot (default is all repositories)
```

Example:

```
$ ./toolbox.sh snapshot -a su:password
```

Related:

- List available snapshots with names: *list-snapshots*
- Restore snapshot: *restore*

restore

Restore a named snapshot. See *Backup and Restore* for more information on snapshots.

Usage:

```
NAME
        toolbox restore - Restores a snapshot of a previous state of the
        repository.

SYNOPSIS
        toolbox restore -a <auth> [-h <host>] [-p <port>] [-r <repository>]
                -s <snapshotName>

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).

        -r <repository>
            The name of the repository to restore. Default is all repositories

        -s <snapshotName>
            The name of the snapshot to restore.
```

Example:

```
$ ./toolbox.sh restore -a su:password -s 2015-07-02t11:53:13.224z
```

Related:

- List available snapshots with names: *list-snapshots*

list-snapshots

List all the snapshots for the installation. See *Backup and Restore* for more information on snapshots.

Usage:

```
NAME
        toolbox list-snapshots - Returns a list of existing snapshots with name
        and status.

SYNOPSIS
        toolbox list-snapshots -a <auth> [-h <host>] [-p <port>]

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).
```

Example:

```
$ ./toolbox.sh list-snapshots -a su:password
```

delete-snapshots

Deletes all snapshots before the given timestamp. See *Backup and Restore* for more information on snapshots.

Usage:

```
NAME
        toolbox delete-snapshots - Deletes snapshots, either before a given
        timestamp or by name.

SYNOPSIS
        toolbox delete-snapshots -a <auth> -b <before> [-h <host>] [-p <port>]

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -b <before>
            Delete snapshots before this timestamp.

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).
```

Example:

```
$ ./toolbox.sh delete-snapshots -a su:password -b 2015-02-14t14:24:20.618z
```

export

Extract data from a given repository, branch and content path. The result will be stored in the `$XP_HOME/data/export` directory. This is useful to move a part of a site from one installation to another. See *Export and Import* for more information on content export/import.

> **Attention:** Exporting content will not include the version history of the content, just the current version.

Usage:

```
NAME
        toolbox export - Export data for a specified path.

SYNOPSIS
        toolbox export -a <auth> [-h <host>] [-p <port>] -s <sourceRepoPath>
                [--skipids] -t <exportName>

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).

        -s <sourceRepoPath>
            Path of data to export. Format:
            <repo-name>:<branch-name>:<node-path>.

        --skipids
            Flag that skips ids in data when exporting.

        -t <exportName>
            Target name to save export.
```

Example:

```
$ ./toolbox.sh export -a su:password -s cms-repo:draft:/ -t myExport
$ ./toolbox.sh export -a su:password -s cms-repo:draft:/content/my-site -t mySiteExport
```

import

Import data from a named export into Enonic XP at the desired content path. The export read has to be stored in the `$XP_HOME/data/export` directory. See *Export and Import* for more information on content export/import.

Usage:

```
OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
```

```
            Port number for server (default is 8080).

        -s <exportName>
            A named export to import.

        --skipids
            Flag that skips ids.

        -t <targetRepoPath>
            Target path for import. Format:
            <repo-name>:<branch-name>:<node-path>. e.g 'cms-repo:draft:/'

        -xslParam <xslParam>
            Parameter to pass to the XSL transformations before importing nodes.
            Format: <parameter-name>=<parameter-value> . e.g. 'applicationId=com.enonic.myapp'

        -xslSource <xslSource>
            Path to xsl file (relative to <XP_HOME>/data/export) for applying
            transformations to node.xml before importing.
```

Example:

```
$ ./toolbox.sh import -a su:password -s myExport -t cms-repo:draft:/
$ ./toolbox.sh import -a su:password -s mySiteExport -t cms-repo:draft:/content
```

Tip: An **XSL** file and a set of *name=value* parameters can be optionally passed for applying transformations to each `node.xml` file, before importing it.

This option could for example be used for renaming types or fields. The *.xsl* file must be located in the `$XP_HOME/data/export` directory.

reindex

Reindex the content in the search indices for the given repository and branches. This is usually required after upgrades, and may be useful in many other situation.

Usage:

```
NAME
        toolbox reindex - Reindex content in search indices for the given
        repository and branches.

SYNOPSIS
        toolbox reindex -a <auth> -b <branches>... [-h <host>] [-i] [-p <port>]
                -r <repository>

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -b <branches>
            A comma-separated list of branches to be reindexed.

        -h <host>
            Host name for server (default is localhost).
        -i
            If flag -i given true, the indices will be deleted before recreated.

        -p <port>
            Port number for server (default is 8080).

        -r <repository>
            The name of the repository to reindex.
```

Example:

```
$ ./toolbox.sh reindex -a su:password -b draft -i -r cms-repo
```

reprocess

Reprocesses content in the repository and **regenerates metadata for the media attachments**. Only content of a media type (super-type = *base:media*) are processed.

Unless the *–skip-children* flag is specified, it processes all descendants of the specified content path.

This command should be used after migrating content from Enonic CMS using the cms2xp tool.

Usage:

```
NAME
        toolbox reprocess - Reprocesses content in the repository.

SYNOPSIS
        toolbox reprocess -a <auth> [-h <host>] [-p <port>]
                -s <sourceBranchPath> [--skip-children]

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).

        -s <sourceBranchPath>
            Target content path to be reprocessed. Format:
          <branch-name>:<content-path>. e.g 'draft:/mySite/media'

        --skip-children
            Flag to skip processing of content children.
```

Example:

```
$ ./toolbox.sh reprocess -a su:password -s draft:/
```

set-replicas

Set the number of replicas in the cluster. For more information on how replicas work and recommended values, see: *Replica setup*.

Usage:

```
NAME
        toolbox set-replicas - Set the number of replicas in the cluster.

SYNOPSIS
        toolbox set-replicas -a <auth> [-h <host>] [-p <port>] [--]
                <numberOfReplicas>

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).

        --
            This option can be used to separate command-line options from the
            list of argument, (useful when arguments might be mistaken for
            command-line options

        <numberOfReplicas>
            Number of replicas
```

Example:

```
$ ./toolbox.sh set-replicas -a su:password 2
```

dump

Make a copy of all the data in an entire system. The result will be stored in the `$XP_HOME/data/dump` directory. See *Export and Import* for more information on system dump/load.

> **Attention:** Performing a dump will delete version history for all data (used for version history and snapshot restoration).

Usage:

```
NAME
        toolbox dump - Export data from every repository.

SYNOPSIS
        toolbox dump -a <auth> [-h <host>] [-p <port>] -t <target>

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).
        -t <target>
            Dump name.
```

Example:

```
$ ./toolbox.sh dump -a su:password -t myDump
```

load

Load data from a named system dump into Enonic XP. The dump read has to be stored in the `$XP_HOME/data/dump` directory. See *Export and Import* for more information on system dump/load.

> **Attention:** A load must never be performed on a repo with existing content. It's meant to be used on empty repos. Make sure the repo is emptied first (delete contents)!

Usage:

```
NAME
        toolbox load - Import data from a dump.

SYNOPSIS
        toolbox load -a <auth> [-h <host>] [-p <port>] -s <source>

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password).

        -h <host>
            Host name for server (default is localhost).

        -p <port>
            Port number for server (default is 8080).

        -s <source>
            Dump name.
```

Example:

```
$ ./toolbox.sh load -a su:password -s myDump
```

upgrade

Upgrade a data dump from a previous version to the current version. The output of the upgrade will be placed alongside the dump that is being upgraded and will have the name <dump-name>_upgraded_<new-version> unless a target location is specified with -t.

The current version XP installation must be running with the upgraded app deployed.

Usage:

```
NAME
        toolbox upgrade - Upgrade a dump.

SYNOPSIS
        toolbox upgrade -d dump-path -t target-location
```

```
OPTIONS
        -d <dump>
           Directory for dump.
```

Example:

```
$ ./toolbox.sh upgrade -d ./data/dump/5.3.1-dump
```

The output would appear as:

```
/data/dump/5.3.1_upgraded_6.0.0/
```

init-project

The init-project tool initializes a new application project structure by retrieving a Git repository, removing all references to the Git repository, and adapting its build file properties (gradle.properties).

Usage:

```
NAME
        toolbox init-project - Initiates an Enonic XP application project

SYNOPSIS
        toolbox init-project [-a <authentication>]
            [(-c <checkout> | --checkout <checkout>)]
            [(-d <destination> | --destination <destination>)]
            (-n <name> | --name <name>)
            (-r <repository> | --repository <repository>)
            [(-v <version> | --version <version>)]

OPTIONS
        -a <authentication>
            Optional authentication token for basic authentication (user:password)

        -c <checkout>, --checkout <checkout>
            Branch or commit to checkout.

        -d <destination>, --destination <destination>
            Optional destination path to create your project, if not specified current
            directory will be used

        -n <name>, --name <name>
            Unique qualifying name that will be given to your application i.e.
            com.company.myapp. NOTE: Choose the name carefully as changing it at a later
            point in time will require updating your content too.

        -v <version>, --version <version>
            Optional version number that will be set in your application project, if not
            used 1.0.0-SNAPSHOT will be set

        -r <repository>, --repository <repository>
            Git repository you wish to use as starting point. Supports both full urls to
            any standard xp git-hosted project, or optionally a GitHub repository path
            (account/repo) - account defaults to "enonic" if not specified
```

Examples:

```
$ toolbox.sh init-project -d ~/Dev/xp/apps/myApp -n com.company.myapp -v 0.9.0 -r https://github.com/
```

```
$ toolbox.sh init-project -n com.company.myapp -v 1.0.0-SNAPSHOT -r enonic/starter-base
```

```
$ toolbox.sh init-project -n com.company.myapp -r starter-base
```

install-app

Installs an application on all nodes.

Usage:

```
NAME
        toolbox install-app - Installs an application on all nodes

SYNOPSIS
        toolbox install-app -a <auth> (-u <url> | -f <file>) [-h <host>] [-p <port>]

OPTIONS
        -a <auth>
            Authentication token for basic authentication (user:password)

        -u <url>
            Url to application

        -f <file>
            Path to application jar-file

        -h <host>
            Host name for server (default is localhost)

        -p <port>
            Port number for server (default is 8080)
```

Example:

```
$  ./toolbox.sh install-app -a su:password -u http://repo.enonic.com/public/com/enonic/app/superhero/

$ ./toolbox.sh install-app -a su:password -f /Users/rmy/Dev/apps/superhero/build/libs/superhero-1.2.0
```

Image Processor

Enonic XP includes a number of image processing commands that may be used to set the size or add style to the images. The commands are appended to the image URLs. To automatically create the URLs, use the *imageUrl* view function.

Scaling

Scale Max

Scales the image proportionally, so the longest edge has the given number of pixels.

Arguments:

size The length of the longest edge. Required

Example:

```
max(600)
```

Scale Wide

Scales the image to fit the given width of the picture. If the image is taller than the given height, it is cropped on top and bottom, based on the focal point position.

Arguments:

width Width in pixels

height maximum height in pixels

Example:

```
wide(600,200)
```

Scale Block

Scales the image, while keeping the aspect ratio, so it fills the rectangle specified by width and height. Then crops the overflowing axis based on the focal point position. The result of a call to this method will be an image that always has the exact size of the specified input.

Arguments:

width Width in pixels

height Height in pixels

Example:

```
block(600,200)
```

Scale Square

Scales the image proprtionally to match the shortest edge. The longest edge will be cropped based on the focal point position.

Arguments:

size The length of both sides in pixels

Example:

```
square(600)
```

Scale Height

Scales the image proportionally to match the given height.

Arguments:

height Height in pixels

Example:

```
height(600)
```

Scale Width

Scales the image proportionally to match the given width.

Arguments:

width Width in pixels

Example:

```
width(600)
```

Styling

Block

Pixelates the image creating a mosaic like effect.

Arguments:

size The number of pixels squared, that should be combined to one block. Default: 2

Example:

```
block(5)
```

Blur

Applies a blur effect.

Arguments:

radius How much blur to apply. Default: 2

Example:

```
blur(8)
```

Border

Applies a rectangular border around the image.

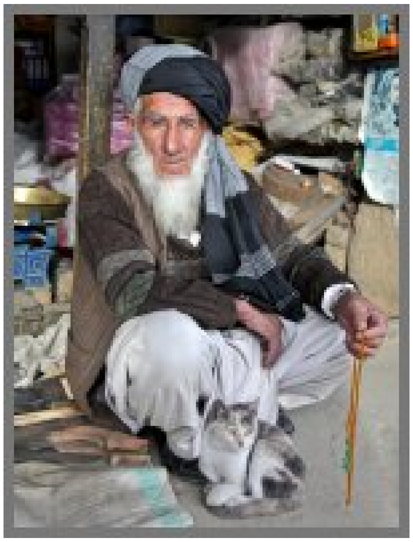
Arguments:

width The width of the border in pixels. Default: 2

color The color of the border as a decimal or hexadecimal number. Default: 0 / 0x000000 (black)

Example:

```
border(5)
border(4, 0x777777)
```



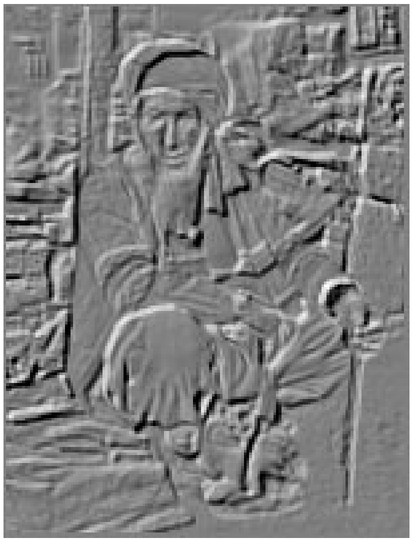
Emboss

Applies an embossing effect on the image.

No arguments

Example:

```
emboss()
```



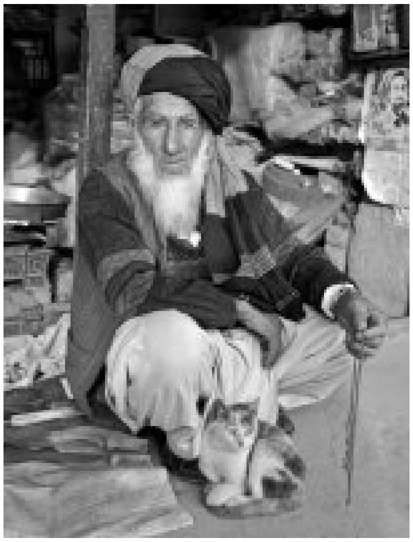
Grayscale

Creates a grayscale variant of the image.

No arguments

Example:

```
grayscale()
```



Invert

Inverts the colors in the image.

No arguments

Example:

```
invert()
```

Rounded

Rounds the corners of the image, with an option of adding a border around the rounded image.

Arguments:

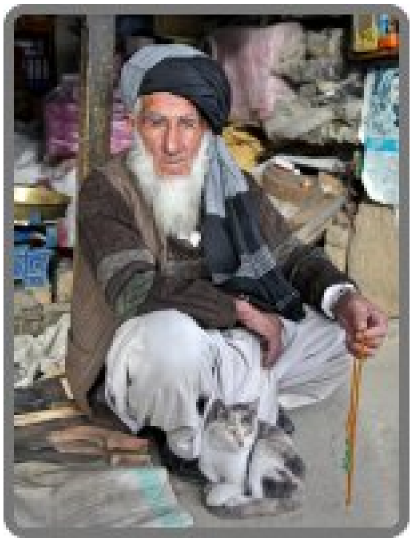
radius The number of pixels from each corner where the rounding starts. Default: 10

borderSize The width of the border in pixels. Default: 0

borderColor The color of the border as a decimal or hexadecimal number. Default: 0 / 0x000000 (black)

Example:

```
rounded()
rounded(15)
rounded(10,1)
rounded(8,4,0x777777)
```



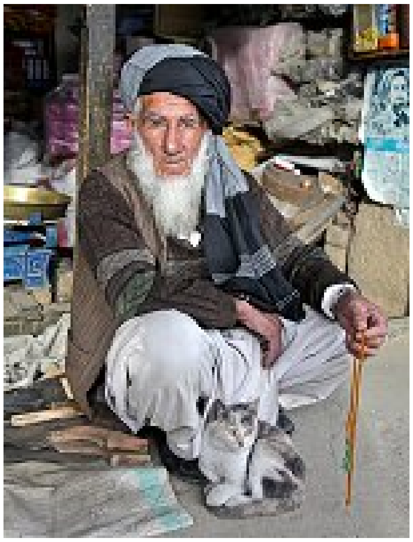
Sharpen

Applies a sharpening filter to the image.

No arguments

Example:

```
sharpen()
```



RGB Adjust

Adjust the red, green and blue levels in the image.

Arguments:

red The adjusted red level for the image. Default: 0

green The adjusted green level for the image. Default: 0

blue The adjusted blue level for the image. Default: 0

Example:

```
rgbadjust(2.0,0.25,-1.75)
```

HSB Adjust

Adjust the hue, saturation and brightness levels in the image.

Arguments:

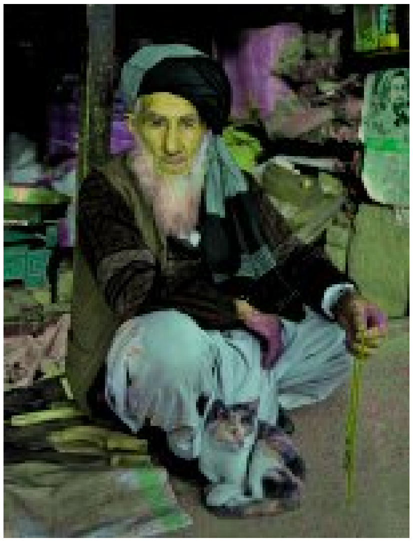
hue Value from -1 to 1, of how far around the color wheel to move the hue of the image. Default: 0

saturation Value from -1 to 1 to adjust the intesity of the colors in the image. Default: 0

brightness Value from -1 to 1 to adjust the brightness of the image. Default: 0

Example:

```
hsbadjust(0.5,-0.1)
hsbadjust(-0.15,0.2,-0.2)
```



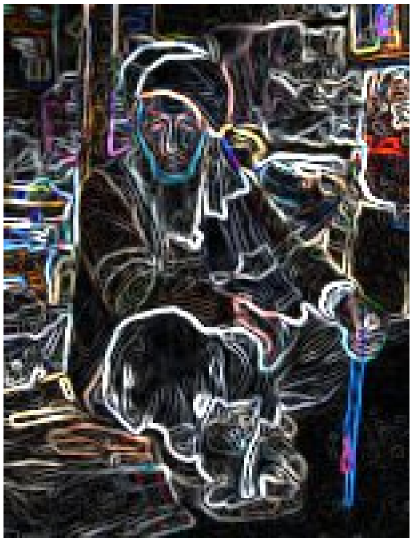
Edge

Creates an abstract image by brightening every edge and darkening every even surface of the image.

No arguments

Example:

```
edge()
```



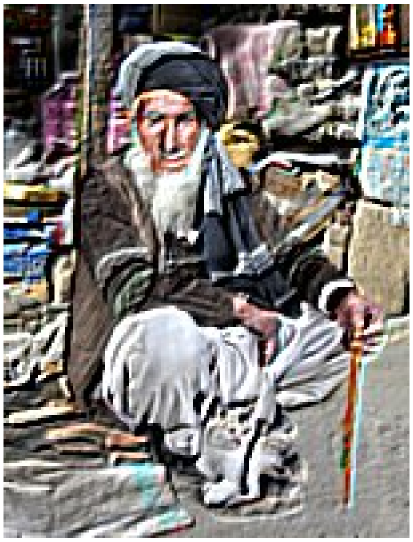
Bump

Creates a 3D looking texture, based on darkening and lighting each side of edges in the image.

No arguments

Example:

```
bump()
```



Sepia

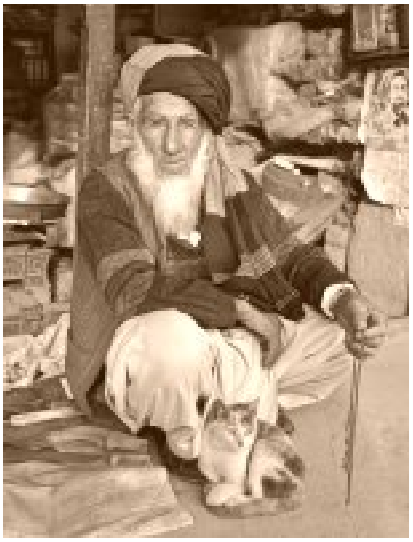
Creates a grayscale image with a yellow-reddish tint to make it look like an old photograph.

Arguments:

depth The brightness of the tint. Default: 20

Example:

```
sepia()
sepia(25)
```



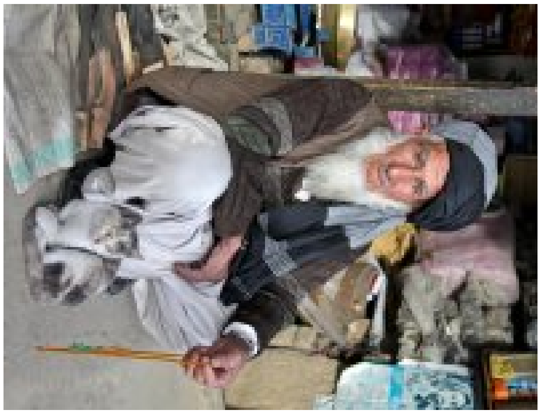
Rotate 90

Rotates an image 90 degrees

No arguments

Example:

```
rotate90()
```



Rotate 180

Rotates an image 180 degrees

No arguments

Example:

```
rotate180()
```

Rotate 270

Rotates an image 270 degrees

No arguments

Example:

```
rotate270()
```

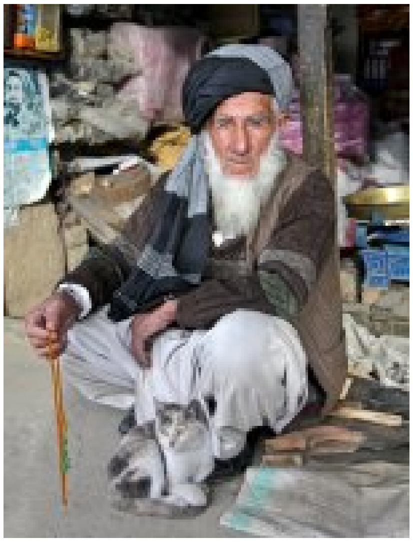
Flip horizontal

Flips an image horizontally

No arguments

Example:

```
fliph()
```



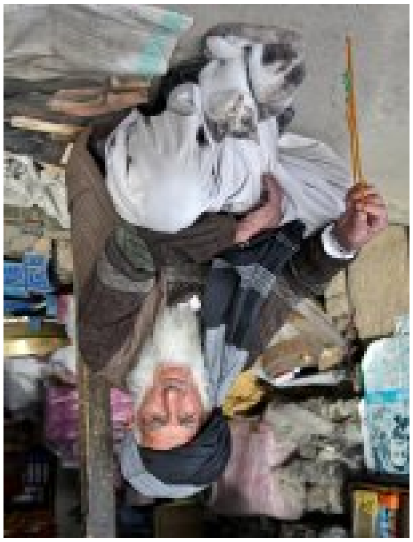
Flip vertically

Flips an image vertically

No arguments

Example:

```
flipv()
```



Colorize

Makes a grayscale image, then applies a tint, based on the specified color.

Arguments:

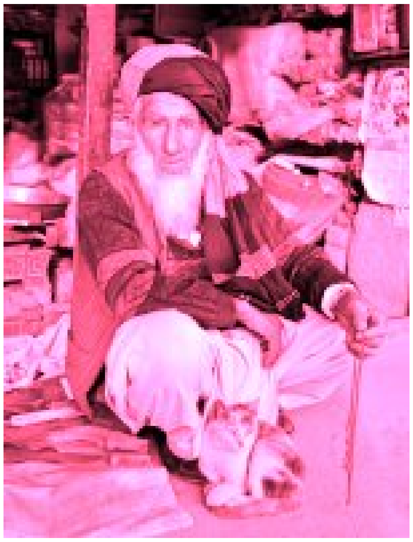
red Red boost value. Default: 1

green Green boost value. Default: 1

blue Blue boost value. Default: 1

Example:

```
colorize(3,1,1.5)
```



HSB Colorize

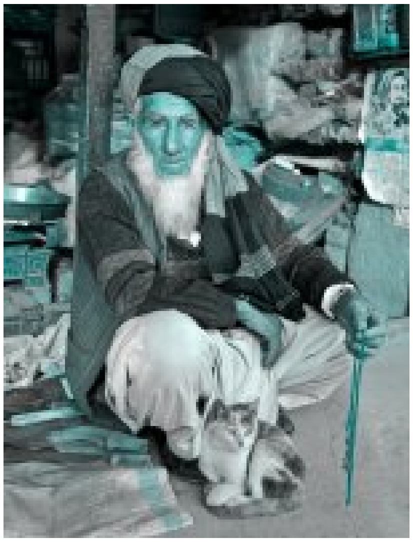
Makes a grayscale image, then applies a tint, based on the specified color.

Arguments:

color The tint color as a decimal or hexadecimal number. Default: 0xFFFFFF

Example:

```
hsbcolorize(0x00AAAA)
```



JavaDoc API

You can either download the JavaDoc as a zip or view it directly in your browser.

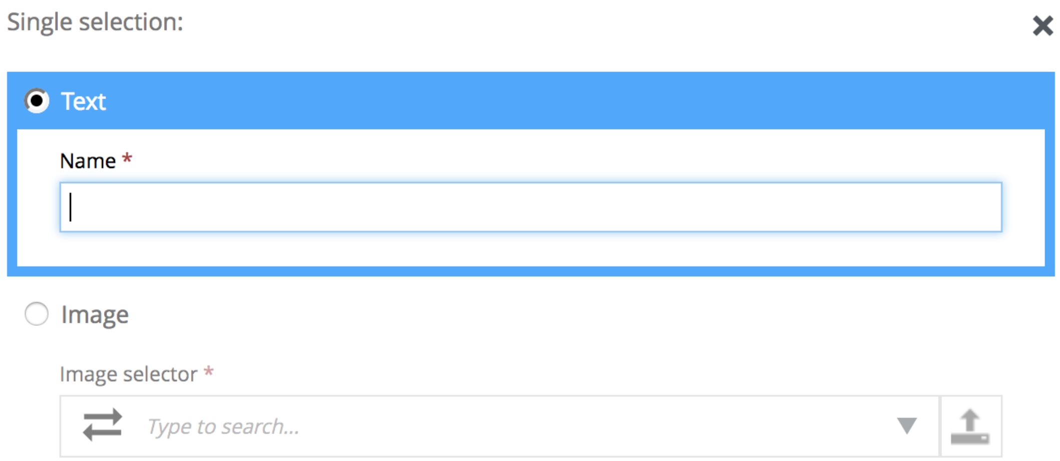
CHAPTER 7

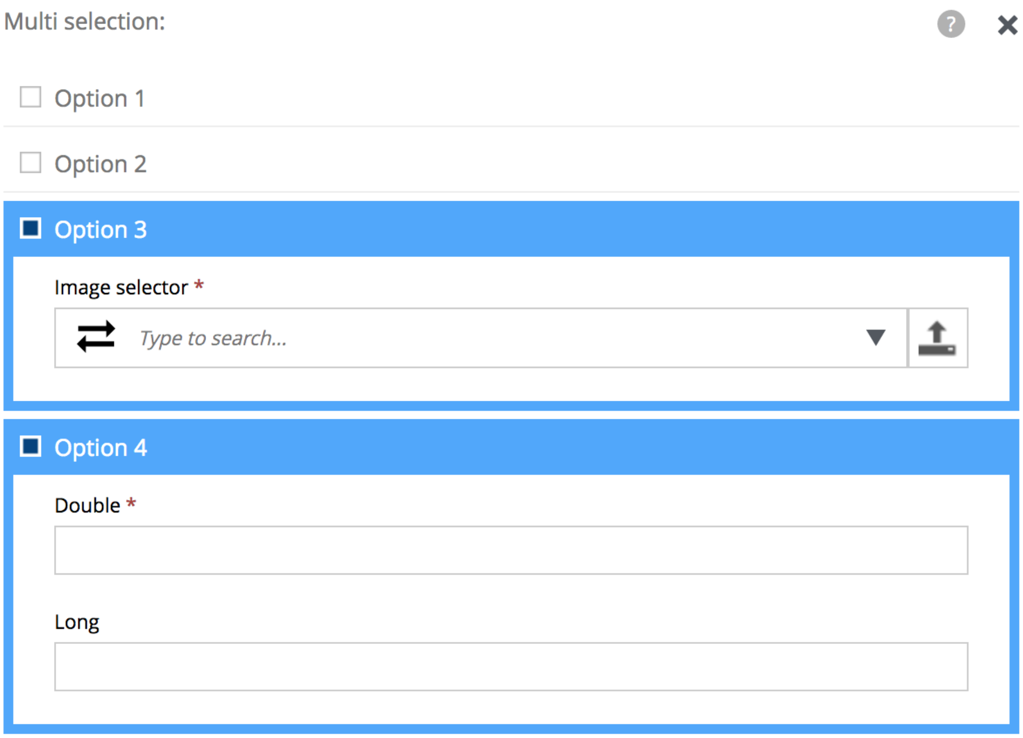
Release Notes

Enonic XP 6.8 is a minor release with new features and improvements.

OptionSet Form Item

This new form item enables fast creation of conditional fields in a schema. Use optionSet to create more advanced forms by allowing the editor to select between the different options - for instance “big article” or “small article” or choose between different types of links such as “content link”, “download link” or “external link”. OptionSet also allow selecting two or more options at once.

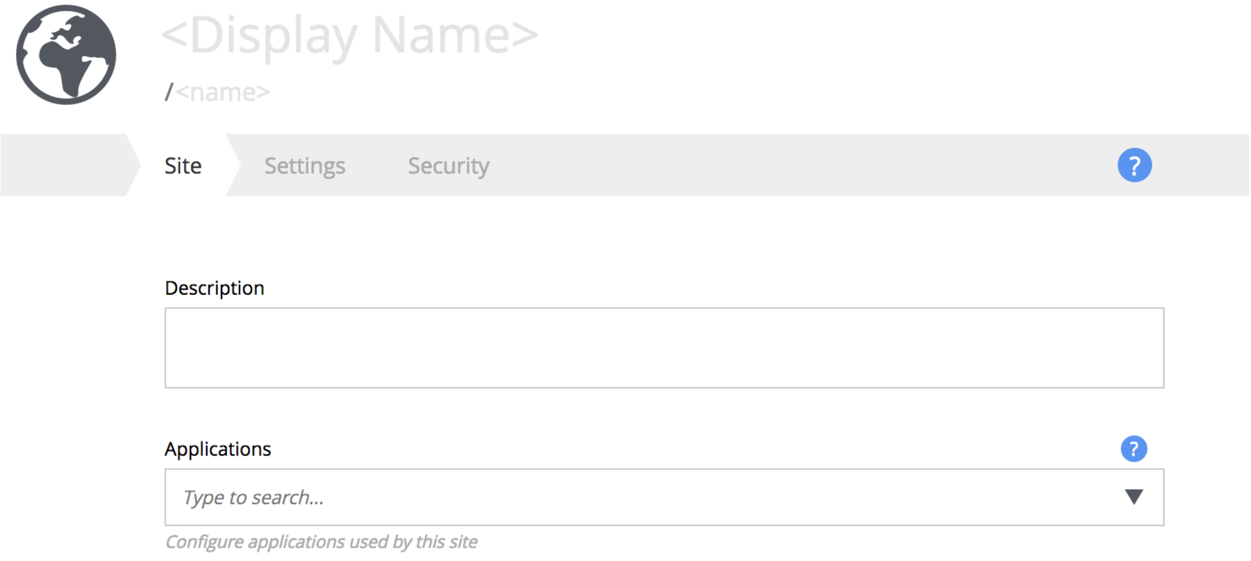




Check out the *Option Sets* documentation.

Expandable Help texts

Help texts can now be shown/hidden for an entire form, or for a single input



Metrics Endpoint

System monitoring is now easier than ever as users can access detailed information about the system through the /status endpoint. Some of the things that are exposed are:

- cluster
- index
- jvm (GC, threads, memory, etc)
- metrics
- osgi
- server

Asynchronous Task API

The Task API enables developers to initialize background tasks, get progress of the task and all other tasks running in the cluster. A task will run both Javascript and Java code. Each task will be started as a separate thread.

Check out the Javascript library *Javascript Libraries* documentation.

Global App configuration

All applications can now easily be configured by adding a file named <application-name>.cfg in the config directory. The configuration is automatically accessible to both Javascript and Java code. Updating a configuration file will automatically restart the corresponding app, injecting the new configuration. The configuration files must be in standard properties format.

Libraries

- http-lib - added support for basic auth
- Content Lib
  - refresh() function to force refresh of index
  - refresh() parameter added to contentLib.create(), default is true

Minor improvements

- Macros - HTML document is now added to macro context, allowing you to create macros such as Table of Contents.
- Schemas - Checkbox input type can now be aligned - for instance left to create nicer form layouts
- Schemas - Custom selector now supports passing of parameters
- Application lifecycle handling - Disposer will be executed when applications are stopped
- Javascript error handlers now support post processing

- Content Studio - preview improved user experience for mobile users
- Page editor - highlighting page component when mouse over for better usability
- Publishing wizard - Shows progress for long running tasks (using the task API)

Changelog

For a complete list of changes and bugfixes see http://github.com/enonic/xp/releases/tag/v6.8.0

CHAPTER 8

Upgrade notes - 6.8

Warning: This documentation describes upgrading from 6.7.x to 6.8.

Note: Enonic XP 6.8 now requires minimum Java 1.8.92

Upgrade Steps

1. Backup the installation

Backup you current installation. This is described in *Backup and Restore*.

You could also do a *dump* of the system, but then you will loose versions if you have to reload it.

2. Install new version

Download Enonic XP http://repo.enonic.com/public/com/enonic/xp/distro/6.8.1/distro-6.8.1.zip and install according to your setup.

Tip: Remember to update any startup scripts you might have to launch your new installation given a server restart

3. Configure XP_HOME

The next step depends on your setup. Do you have your **$XP_HOME** folder outside or inside the **$XP_INSTALL** folder?

Outside the $XP_INSTALL - folder:

Make sure the new installation points to the correct $XP_HOME folder.

Inside the $XP_INSTALL - folder:

Copy your $OLD_XP_INSTALL/home folder to the the new $NEW_XP_INSTALL/ (on all nodes).

4. Stop the old installation

5. Start the new installation

Stricter Site config validation

Validation of application configuration forms in a site has been improved. Where the application configuration is updated while editing a site in *Content Studio*, an extra validation has been added to the server. Therefore there are potentially some existing invalid values that were accepted before, but will cause the Site content to be marked as invalid after the upgrade.

If updating a site makes it invalid after the upgrade, double-check the config in `site.xml` for the applications in the site.

CHAPTER 9

Frequently Asked Questions

What's the latest release?

The latest release can be found here latest release changelog.

Where can I get the source code?

All source code for Enonic XP is published on our GitHub project page.

Do you publish changelogs?

Yes. You can go to the releases tab on GitHub to read the changelog for all versions. If you want to see what's coming, you can go to our GitHub wiki page.

How is Enonic XP Licensed

Enonic XP is available under the AGPL 3 license, with a linking exception. This basically means that you are free to use and re-distribute Enonic XP according to the AGPL license. The linking exception ensures that you can build custom applications and libraries on top of Enonic XP and license these however you see fit. Any changes made to the Enonic XP core platform however must be licensed as APGL.

We encourage the use of FOSS licensing for 3rd party apps and libraries. Libraries in particular should be licensed with a non-intrusive license such as Apache, BSD or MIT.

Note: This is a major difference from products such as Drupal and Wordpress where your themes, plugins and modules must be licensed as GPL too.

Enonic XP also consists of many 3rd party software components. The complete list can be found in our Notice.txt file

What is $XP_INSTALL?

$XP_INSTALL and **$XP_HOME** are referenced frequently in the documentation and it is important to understand the difference. $XP_INSTALL is the top level directory of the XP installation and it contains the directories *bin*, *home*, *lib*, *toolbox* and others.

What is $XP_HOME?

$XP_HOME, by default, is the location of the $XP_INSTALL/home folder which contains the *config*, *deploy*, *repo* and other directories specific to a single XP instance. The home folder can be copied to multiple locations for developers working on more than one project.

There are two situations where the **$XP_HOME environment variable** must be set:

1. When developers are working on an application and intend to use *./gradlew deploy*.
2. When a *home* folder other than $XP_INSTALL/home is to be used.

Where can I get help?

The community forum would be a good place to start. We also offer formal training courses.

Enonic also offers software support subscriptions for business critical installations: https://enonic.com/pricing

CHAPTER 10

Glossary

$XP_HOME By default, this is the location of the $XP_INSTALL/home folder and it contains directories specific to a single XP instance. The home folder can be copied to multiple locations for developers working on multiple isolated projects. The $XP_HOME environment variable should be set to the home folder of the project to be run.

$XP_INSTALL The the location of the unzipped XP download

Tip: The source code for this documentation is available on GitHub .

Index