How to use .content

`.content` is a shorthand for commonly used Content XML operations such as creating content with components and references. In Nitro content defined in this format can be converted in package-time to regular Content XML and thus imported using regular channels.

How do I write .content?

Create a new file that ends with `.content` in the source directory. The format is line oriented, one property per line with name and arguments separated by colon (:). Each content starts with an **id** property with the external id of the content, and each following line until the next **id** defines properties of this content.

example.article.content

```
id:example.article
major:Article
inputtemplate:example.StandardArticle
name:Hello .content
securityparent:GreenfieldTimes.d
component:lead:value:This is an article defined in .content
component:text:value:It is converted from .content to .xml by the p-maven-plugin.
```

How is the conversion performed?

To convert `.content` files in `src/main/content` and include the `.xml` version in the generated `${artifactId}-contentdata.jar`, configuration of the Polopoly Maven plugin in the artifact POM is required. The following example pom will (1) convert `.content` files from the `src/main/content/dotcontent` folder, (2) put the resulting generated content XML in `target/generated-content` and (3) import content XML from `src/main/content` as well as the converted content XML from `target/generated-content`.

${CONTENT_ARTIFACT}/pom.xml

```
<project>
  <build>
    <plugins>
      <plugin>
        <groupId>com.polopoly.extensions</groupId>
        <artifactId>p-maven-plugin</artifactId>
        <version>${polopoly.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>dot-content-to-xml</goal>
              <goal>pack</goal>
            </goals>
            <configuration>

              <!-- list of folders with .content files to be consumed by the dot-content-to-xml g
              <sources>
                <source>
                  <directory>${project.basedir}/src/main/content/dotcontent</directory>
                  <includes>
                    <include>*.content</include>
                    <include>**/*.content</include>
                  </includes>
                </source>
                <source>
                  <directory>${project.basedir}/src/main/content/more-dotcontent</directory>
                  <includes>
                    <include>*.content</include>
                    <include>**/*.content</include>
                  </includes>
                </source>
              </sources>
              <contentDataFileResources>
                <contentDataFileResource>
                  <directory>${project.basedir}/src/main/content</directory>
                  <includes>
                    <include>*</include>
                    <include>**/*</include>
                  </includes>
                  <excludes>
                    <exclude>*.content</exclude>
                    <exclude>**/*.content</exclude>
                    <exclude>*.xml</exclude>
                    <exclude>**/*.xml</exclude>
                  </excludes>
                </contentDataFileResource>
              </contentDataFileResources>

              <!-- list of folder from where to get content xml (default values)-->
              <contentDataXmlResources>
                <contentDataXmlResource>
                  <directory>${project.basedir}/src/main/content</directory>
                  <includes>
                    <include>*.xml</include>
                    <include>**/*.xml</include>
                  </includes>
                </contentDataXmlResource>

                <contentDataXmlResource>
                  <!-- the dotcontent maven goal converts .content files into content XML
                       that ends up in this directory -->
                  <directory>${project.build.directory}/generated-content</directory>
                  <includes>
                    <include>*.xml</include>
                    <include>**/*.xml</include>
                  </includes>
                </contentDataXmlResource>
              </contentDataXmlResources>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

Greenfield Times

In the Greenfield Online project, `.content` is supported by default.
See `${PROJECT}/content/demo-content/src/main/content/dot-content` for `.content` examples.

`.content` files will automatically be converted to `.xml` during installation (ie. during `p:run`) and changes will automatically be imported by The content import scanner.

What properties can I set on a content using `.content`?

Common for all properties

Each line contains a property name followed by a number of arguments, all separated by `:`, to include `:` in a argument escape it with \ (`\n` can be used for new line). Eg.

common.content

```
component:text:value:I have a \:\n new line
```

common.xml

```
<component group="text" name="value"><![CDATA[I have a :
new line]]></component>
```

Any property that takes a external id argument, ie `id`, `securityparent`, `inputtemplate` and `ref` will expand a leading dot into the base name of the file. Eg.

reference.content

```
id:.name
id:.ref
ref:group:name:.name
```

reference.xml

```
<content>
  <metadata>
    <contentid>
      <major>1</major>
      <externalid>reference.name</externalid>
    </contentid>
  </metadata>
</content>
<content>
  <metadata>
    <contentid>
      <major>1</major>
      <externalid>reference.ref</externalid>
    </contentid>
  </metadata>
  <contentref group="group" name="name">
    <contentid>
      <externalid>reference.name</externalid>
    </contentid>
  </contentref>
</content>
```

Property "`id`"

The `id` parameter sets the external id of the content, and also starts a new content definition.

Field	Value
0	**id**
1	The external id of the content

Property "`major`"

The `major` parameter sets the Major of the content, if not present the default is `Article`.

Field	Value
0	major
1	The Major of the content

Valid Majors
majorconfig
article
department
content
layoutelement
workflowtype
workflow
referencemetadata
inputtemplate
outputtemplate
appconfig
userdata
community

Property "`inputtemplate`"

The `intputemplate` parameter sets the input template of the content.

Field	Value
0	inputtemplate
1	The external id of the input template of the content

Property "`name`"

The `name` parmater sets the content name.

Field	Value
0	name
1	The name of the content

Property "`securityparent`"

The `securityparent` parameter sets the security parent of the content.

Field	Value
0	securityparent
1	The external id of the security parent of the content

Property "`component`"

The `component` parameter sets a component of the content.

Field	Value
0	component

1	The component group to set
2	The component name to set
3	The component string value to set

Property "`ref`"

The `ref` parameter sets a content reference of the content.

Field	Value
0	ref
1	The content reference group to set
2	The content reference name to set
3	The content reference external id value to set

Property "`list`"

The `list` parameter adds an external id to a content list, the list will be *reset* to the value of all entries with the same name in the order they appear in the `.content` file.

The external id to insert is mandatory, but there are also optional arguments for the group and metadata external id.

Using a single parameter will insert the provided external id into the list `polopoly.Department`.

Field	Value
0	list
1	The external id to insert into the content list

Using two parameters will insert the provided external id into the named content list.

Field	Value
0	list
1	The content list to insert into
2	The external id to insert into the content list

Using three parameters will insert the provided external id into the named list using a reference metadata link content.
Note that the metadata has to be a reference metadata content, ie use `major:referencemetadata` when defining it.

Field	Value
0	list
1	The content list to insert into
2	The external id to insert into the content list
3	The external id of the reference meta data to insert into the content list

Property "`publish`"

The `public` property adds the content to a content list in another content.

The external id to insert into is mandatory, but there are also optional arguments for the group and metadata external id.

Using a single parameter will insert this content id into the provided external id's content list named `polopoly.Department`.

Field	Value
0	publish
1	The external id to insert into

Using two parameters will insert this content id into the provided external id's content list with the given name.

Field	Value

0	publish
1	The content list to insert into
2	The external id to insert into

Using three parameters will insert this content id into the provided external id's content list with the given name using a reference metadata link content.
Note that the metadata has to be a reference metadata content, ie use `major:referencemetadata` when defining it.

Field	Value
0	publish
1	The content list to insert into
2	The external id to insert into
3	The external id of the reference metadata to insert into the content list

Property "`file`"

The `file` property adds an inline file to a named path on the current content.

Field	Value
0	file
1	The path of the file on the content
2	The path of the file to be included in the .xml, relative to the directory of the *.content* file

Example

file.txt

```
I am a file
```

file.content

```
id:.
file:path/file.txt:file.txt
```

file.xml

```
<content>
  <metadata>
    <contentid>
      <major>1</major>
      <externalid>file</externalid>
    </contentid>
  </metadata>
  <file encoding="base64" name="path/file.txt">SSBhbSBhIGZpbGUK
  </file>
</content>
```

Property "`template`"

The `template` property points to another content defined in the same file and copies all properties of that content into this one.

Field	Value
0	template
1	The external id of the content to copy properties from

Example

template.content

```
id:.parent
name:This is a content template
component:group:name:value:important
id:.copy
name:This overrides the template
template:.parent
```

template.xml

```
<content>
  <metadata>
    <contentid>
      <major>1</major>
      <externalid>template.copy</externalid>
    </contentid>
  </metadata>
  <component group="polopoly.Content" name="name"><![CDATA[This overrides the template]]></compon
  <component group="group" name="name"><![CDATA[important]]></component>
</content>
<content>
  <metadata>
    <contentid>
      <major>1</major>
      <externalid>template.parent</externalid>
    </contentid>
  </metadata>
  <component group="polopoly.Content" name="name"><![CDATA[This is a content template]]></compone
  <component group="group" name="name"><![CDATA[important]]></component>
</content>
```

Property "`action`"

The `action` property defines a workflow action to perform on the imported content.

Field	Value
0	action
1	The workflow action

Example

workflow.content

```
id:.article
major:article
inputtemplate:example.StandardArticle
name:This article will be approved
securityparent:GreenfieldTimes.d
action:approve
```