Escenic Content Engine

Resource Reference

5.6.13.183224

escenic

A CCI COMPANY

Table of Contents

1 Introduction

This manual contains reference material describing the Escenic **publication resources**. The publication resources are text files containing important system configuration information that determines the structure and other characteristics of an Escenic publication. Publication resources are by convention stored in the following location in a publication JAR file:

`META-INF/escenic/publication-resources/escenic`

There are four publication resources:

`content-types`
: An XML file that defines the types of content items allowed in a publication.

`image-version`
: An XML file that defines the image versions available for use in a publication.

`layout-group`
: An XML that defines the article, section and element templates available to the editors/writers of a publication and the logical structure of the sections in a publication.

`feature`
: A plain text file containing property settings that define miscellaneous aspects of the Content Engine's behavior.

In order for changes to a publication resource to take effect, the- resource file must be uploaded to the Content Engine using the web administration interface. For a description of how to do this, see Creating/Updating Publication Resources (**Escenic Content Engine Template Developer Guide**, **section 1.4.2.2**) .

In addition to the resources listed above, this manual also contains a description of the XML elements in the **`interface-hints`** namespace. These elements can optionally be inserted at various points in the XML resource files in order to:

- Provide a richer, more user-friendly interface in Content Studio.
- Add extra (mostly presentation-related) information that can be accessed and used by your publication's template code.

1.1 Syntax Diagram Conventions

The descriptions of the XML resource files include diagrams describing the syntax of the elements in the files. The diagrams look something like this:

```
<content-types
    version="4"
>
  <field-group>...<field-group>*
  <relation-type-group>...<relation-type-group>*
  <content-type>...<content-type>+
  ANY-FOREIGN-ELEMENT*
</content-types>
```

In these diagrams, anything appearing in plain black characters is **literal** content: that is, it should be entered in the file exactly as shown. Anything appearing in black, italic characters is a placeholder that must be replaced by something else. The only such placeholders currently in use are:

- *ANY-FOREIGN-ELEMENT* - This placeholder can be replaced by any **foreign** element. A foreign element is an element from a different namespace. This is mainly intended to allow you to enter elements from the `interface-hints` namespace (see chapter 5). You can, however, insert elements from any other namespace if you have the need.
- *ANY-ELEMENT* - This placeholder can be replaced by any element from this namespace or any other.

Anything appearing in **this color** is neither literal content nor a placeholder, but has one of the following special meanings:

()
Encloses a set of alternatives, only one of which may be selected.

|
Separates the alternatives in a **()** set.

?
Indicates that the preceding element is optional.

*****
Indicates that the preceding element may appear 0 or more times.

+
Indicates that the preceding element may appear 1 or more times.

...
Represents possible content in an element or attribute.

> Element order is **not significant** in any of the XML publication resource files. The syntax diagram shown above seems to suggest that a `field-group` element must appear before a `relation-type-group` element, but this is **not** the case: the elements can in fact appear in any order.

2 content-type

The **content-type** schema defines the content of the Escenic **content-type** publication resource. The purpose of the **content-type** resource is to specify:

- All the content types and relations allowed in a particular Escenic publication.
- How those content types are organized and presented in Content Studio.

Namespace URI

The namespace URI of the **content-type** schema is **http://xmlns.escenic.com/2008/content-type**.

Root Element

The root of a **content-type** file must be a **content-types** element.

2.1 array

Specifies that this is an array **field**. An array **field** may contain more than one value. Any field type may be an array.

Syntax

```
<array
    default="integer"
    max="integer"?
  />
```

Attributes

default="*integer*"
The default number of elements in this array field. This determines how many data entry controls are initially displayed in Content Studio.

max="*integer*" (optional)
The maximum number of elements allowed in this array field.

2.2 complex

Provides a wrapper for the members of a complex field.

Syntax

```
<complex>
  (<ref-field-group/>|<field>...</field>)+
</complex>
```

Child Elements

`ref-field-group`: section 2.17, **`field`**: section 2.7.

The following forms of the **`field`** element may be used: **Basic field (Simplified)** (section 2.7.2), **Boolean field (Simplified)** (section 2.7.4), **URI field (Simplified)** (section 2.7.20), **Schedule field (Simplified)** (section 2.7.18), **Number field (Simplified)** (section 2.7.16), **Link field (Simplified)** (section 2.7.14), **Enumeration field (Simplified)** (section 2.7.11), **Date field (Simplified)** (section 2.7.9), **Collection field (Simplified)** (section 2.7.6).

2.3 constraints

This element can appear in a number of different forms, described in the following sections.

2.3.1 Boolean constraints

Sets constraints on the values that can be entered in a boolean, enumeration, schedule, date or collection **`field`**.

Syntax

```
<constraints>
  <required>...</required>?
</constraints>
```

2.3.2 Link constraints

Sets constraints on the values that can be entered in a link **`field`**. Since link fields are used to hold references to binary data, the constraints actually apply to the referenced binary data rather than the reference itself.

Syntax

```
<constraints>
  <required>...</required>?
    <mime-type>...</mime-type>*
</constraints>
```

Examples

- This example shows typical constraints for a link field that is to be used for image objects.

```
<constraints>
  <mime-type>image/jpeg</mime-type>
  <mime-type>image/png</mime-type>
</constraints>
```

2.3.3 Number constraints

Sets constraints on the values that can be entered in a numeric **`field`**.

Syntax

```
<constraints>
  <required>...</required>?
```

```
    <minimum>...</minimum>?
    <maximum>...</maximum>?
</constraints>
```

Examples

- ```
  <constraints>
    <minimum>1</minimum>
    <maximum>6</maximum>
  </constraints>
  ```

2.3.4 Text constraints

Sets constraints on the values that can be entered in a basic **`field`**.

Syntax

```
<constraints>
  <required>...</required>?
    <maxchars>...</maxchars>?
    <well-formed>...</well-formed>?
</constraints>
```

2.4 content-type

Defines an Escenic **content type**. A content type defines a particular type of content item. It defines:

- The **`field`**s a content item is composed of.
- The **`relation-type`**s a content item may have.
- How the content item is presented in Content Studio.

The **`field`**s in the content type are defined indirectly: a **`content-type`** element contains **`panel`** elements, which in turn contain **`field`** elements. This allows the fields to be grouped together in panels, which correspond to tabs in Content Studio content item editors.

Syntax

```
<content-type
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT*
    <ref-relation-type-group/>*
    <panel>...</panel>+
    <parameter/>*
    <summary>...</summary>?
    <url>...</url>?
</content-type>
```

Examples

- This example defines a simple content type containing only one **`panel`**, a **`summary`** and a reference to a **`relation-type-group`**.

  ```
  <content-type name="news">
  ```
  ```

```
  <ui:label>Story</ui:label>
  <ui:description>A news story</ui:description>
  <ui:title-field>title</ui:title-field>
  <panel name="main">
    <ui:label>Main Content</ui:label>
    <ui:description>The main content fields</ui:description>
    <ref-field-group name="title"/>
    <ref-field-group name="summary"/>
    <ref-field-group name="body"/>
  </panel>
  <ref-relation-type-group name="attachments"/>
  <summary>
    <ui:label>Content Summary</ui:label>
    <field name="title" type="basic" mime-type="text/plain"/>
    <field name="summary" type="basic" mime-type="text/plain"/>
  </summary>
  <url>{yyyy}/{MM}/{dd}/article{id}.ece/{field.title}</url>
</content-type>
```

- This example defines a content type to be used for images. Note the use of a link field to store the image reference. Note also the use of **`ui:icon`** to select an icon for this type of content item in Content Studio.

```
<content-type name="image">
  <ui:label>Picture</ui:label>
  <ui:description>An image</ui:description>
  <ui:title-field>name</ui:title-field>
  <ui:icon>image</ui:icon>
  <panel name="main">
    <ui:label>Image content</ui:label>
    <field mime-type="text/plain" type="basic" name="name">
      <ui:label>Name</ui:label>
      <ui:description>The name of the image</ui:description>
      <constraints>
        <required>true</required>
      </constraints>
    </field>
    <field mime-type="text/plain" type="basic" name="description">
      <ui:label>Description</ui:label>
    </field>
    <field mime-type="text/plain" type="basic" name="alttext">
      <ui:label>Alternative text</ui:label>
    </field>
    <field name="binary" type="link">
      <relation>com.escenic.edit-media</relation>
      <constraints>
        <mime-type>image/jpeg</mime-type>
        <mime-type>image/png</mime-type>
      </constraints>
    </field>
  </panel>
  <panel name="crop">
    <ui:label>Cropped Versions</ui:label>
    <field mime-type="application/json" type="basic" name="representations">
      <ui:label>Image Versions</ui:label>
      <rep:representations type="image-versions">
        <rep:representation name="thumbnail">
          <rep:output width="100" height="100"/>
          <rep:crop/>
          <rep:resize/>
```

```
        </rep:representation>
        <rep:representation name="narrow">
          <rep:output width="500" height="400"/>
          <rep:crop/>
          <rep:resize/>
        </rep:representation>
        <rep:representation name="wide">
          <rep:output width="1000" height="800"/>
          <rep:crop/>
          <rep:resize/>
        </rep:representation>
      </rep:representations>
    </field>
  </panel>
  <summary>
    <ui:label>Content Summary</ui:label>
    <field name="caption" type="basic" mime-type="text/plain"/>
    <field name="alttext" type="basic" mime-type="text/plain"/>
  </summary>
</content-type>
```

Attributes

name="*NCName*"
The name of the **content-type** element.

2.5 content-types

The root element of a **content-type** publication resource. It contains a sequence of **field-group**, **relation-type-group** and **content-type** elements that together define all the field and relation types that are to be available in a publication.

Syntax

```
<content-types
    version="4"
  >
  <field-group>...</field-group>*
    <relation-type-group>...</relation-type-group>*
    <content-type>...</content-type>+
    ANY-FOREIGN-ELEMENT*
</content-types>
```

Attributes

version="4"
The version of the **content-types** schema. It must be set to **4**.

2.6 enumeration

Defines an enumeration **field** option.

An enumeration element can have a child **`label`** element from the **`http://xmlns.escenic.com/2008/interface-hints`** namespace. This label is then displayed instead of the **`value`** attribute in the Content Studio user interface.

The following **`field`** definition, for example:

```
<field type="enumeration">
 <enumeration value="1">
  <label xmlns="http://xmlns.escenic.com/2008/interface-hints">One</label>
 </enumeration>
 <enumeration value="2">
  <label xmlns="http://xmlns.escenic.com/2008/interface-hints">Two</label>
 </enumeration>
</field>
```

will result in a field that can hold the value "1" or "2". It will be displayed in Content Studio, however, as a combo box with the options "One" and "Two".

Syntax

```
<enumeration
    value="string"
  >
  ANY-FOREIGN-ELEMENT*
</enumeration>
```

Attributes

`value="`*`string`*`"`
The value of an enumeration field option.

2.7 field

This element can appear in a number of different forms, described in the following sections.

2.7.1 Basic field

Defines a **basic** field. A basic field can contain text of any kind. You can, however, use the mime-type attribute to specify the allowed content more narrowly.

Syntax

```
<field
    name="NCName"
    type="basic"
    mime-type="text"
  >
  <array/>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <rep:representations>...</rep:representations>?
    <options>...</options>?
</field>
```

Child Elements

`array`: section 2.1, **`constraints`**: section 2.3, **`parameter`**: section 2.16, **`rep:representations`**: section 2.25, **`options`**: section 2.14.

Only one form of the **`constraints`** element may be used: **Text constraints** (section 2.3.4).

Examples

- This example defines a plain text field used to hold the title of an article.

```
<field mime-type="text/plain" type="basic" name="title">
  <ui:label>Title</ui:label>
  <ui:description>The title of the article</ui:description>
  <constraints>
    <required>true</required>
  </constraints>
</field>
```

- This example defines an HTML text field used to hold the main body of an article. Note the use of the **`ui:style`** element to control the appearance of **`h1`** elements in the field in Content Studio. You can specify a complete stylesheet inside this element.

```
<field mime-type="application/xhtml+xml" type="basic" name="body">
  <ui:label>Body</ui:label>
  <ui:description>The body text of the article.</ui:description>
  <ui:style>h1 {color:red;}</ui:style>
</field>
```

- This example shows how a basic **`field`** is used to define image crop masks.

```
<field mime-type="application/json" type="basic" name="representations">
  <ui:label>Image Versions</ui:label>
  <rep:representations type="image-versions">
    <rep:representation name="thumbnail">
      <rep:output width="100" height="100"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
    <rep:representation name="narrow">
      <rep:output width="500" height="400"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
    <rep:representation name="wide">
      <rep:output width="1000" height="800"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
  </rep:representations>
</field>
```

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="basic"`
Specifies that this field is a basic field.

mime-type="*text*"
The MIME type of the field. Two values are supported by default:

text/plain (default)
A simple text editing field is displayed in Content Studio for this MIME type.

application/xhmtl+xml
A rich text editing field is displayed in Content Studio for this MIME type.

application/json
If the field also has a child **rep:representations** element, then an image cropping field is displayed in Content Studio for this MIME type.

The field will not display any value if it contains any invalid data.

You can, however, define MIME types of your own and write corresponding field editor plug-ins.

2.7.2 Basic field (Simplified)

Defines a **basic** field. A basic field can contain text of any kind. You can, however, use the mime-type attribute to specify the allowed content more narrowly.

Syntax

```
<field
    name="NCName"
    type="basic"
    mime-type="text"
  >
  <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <rep:representations>...</rep:representations>?
</field>
```

Child Elements

constraints: section 2.3, **parameter**: section 2.16, **rep:representations**: section 2.25.

Only one form of the **constraints** element may be used: **Text constraints** (section 2.3.4).

Examples

- This example defines a plain text field used to hold the title of an article.

```
<field mime-type="text/plain" type="basic" name="title">
  <ui:label>Title</ui:label>
  <ui:description>The title of the article</ui:description>
  <constraints>
    <required>true</required>
  </constraints>
</field>
```

- This example defines an HTML text field used to hold the main body of an article. Note the use of the **ui:style** element to control the appearance of **h1** elements in the field in Content Studio. You can specify a complete stylesheet inside this element.

```
<field mime-type="application/xhtml+xml" type="basic" name="body">
  <ui:label>Body</ui:label>
  <ui:description>The body text of the article.</ui:description>
```

```
    <ui:style>h1 {color:red;}</ui:style>
  </field>
```

- This example shows how a basic **`field`** is used to define image crop masks.

```
<field mime-type="application/json" type="basic" name="representations">
  <ui:label>Image Versions</ui:label>
  <rep:representations type="image-versions">
    <rep:representation name="thumbnail">
      <rep:output width="100" height="100"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
    <rep:representation name="narrow">
      <rep:output width="500" height="400"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
    <rep:representation name="wide">
      <rep:output width="1000" height="800"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
  </rep:representations>
</field>
```

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="basic"`
Specifies that this field is a basic field.

`mime-type="`*text*`"`
The MIME type of the field. Two values are supported by default:

`text/plain` **(default)**
A simple text editing field is displayed in Content Studio for this MIME type.

`application/xhmtl+xml`
A rich text editing field is displayed in Content Studio for this MIME type.

`application/json`
If the field also has a child **`rep:representations`** element, then an image cropping field is displayed in Content Studio for this MIME type.

The field will not display any value if it contains any invalid data.

You can, however, define MIME types of your own and write corresponding field editor plug-ins.

2.7.3 Boolean field

Defines a boolean field that can hold only one of two values (**`true`** or **`false`**) but may also be left unspecified. A boolean field is displayed as a check box in Content Studio.

Syntax

```
<field
    name="NCName"
```

```
    type="boolean"
  >
  <array/>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <options>...</options>?
</field>
```

Child Elements

`array`: section 2.1, **`constraints`**: section 2.3, **`parameter`**: section 2.16, **`options`**: section 2.14.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="boolean"`
Defines the type of the field.

2.7.4 Boolean field (Simplified)

Defines a boolean field that can hold only one of two values (**`true`** or **`false`**) but may also be left unspecified. A boolean field is displayed as a check box in Content Studio.

Syntax

```
<field
    name="NCName"
    type="boolean"
  >
  <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
</field>
```

Child Elements

`constraints`: section 2.3, **`parameter`**: section 2.16.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="boolean"`
Defines the type of the field.

2.7.5 Collection field

Defines a collection field, which can be used to hold a reference to an Atom feed entry.

Collection field values are represented in Content Studio by graphic components called **tokens**.

Syntax

```
<field
    name="NCName"
    type="collection"
    src="text"
    mime-type="text"
    ( select="(content|title|locator)" | select="link" linkrel="text" )
  >
  <array/>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <options>...</options>?
</field>
```

Child Elements

`array`: section 2.1, **`constraints`**: section 2.3, **`parameter`**: section 2.16, **`options`**: section 2.14.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="collection"`
Specifies that this field is a collection field.

`src="`*text*`"`
Contains a URI that references a resource with an **`application/atom+xml;type=feed`** representation. The entries in the referenced feed are presented as a set of alternatives (a drop-down field in Content Studio) from which the user can choose one value.

`mime-type="`*text*`"`
Defines the MIME type of the field.

`select="(content|title|locator)"`
Specifies which part of the selected Atom entry's content is to be stored as the field value.

Allowed values are:

`content`
The content of the selected Atom entry's **`atom:content`** element is to be stored as the field value.

`title`
The content of the selected Atom entry's **`atom:title`** element is to be stored as the field value.

locator
The **href** attribute of one of the selected Atom entry's **viz:locator** elements is to be stored as the field value. There are two points to note in this case:

1. **viz:locator** is a proprietary Vizrt Atom extension element belonging to the **http://www.vizrt.com/types** namespace.
2. An Atom entry may contain several **viz:locator** elements. The field value is taken from the one with the correct MIME type: that is, the one with a **type** attribute that matches this element's **mime-type** attribute.

select="link"
Specifies that the **href** of one the selected Atom entry's **atom:link** elements is to be stored as the field value. The specific **atom:link** to be used is determined by the **linkrel** attribute.

linkrel="*text*"
Specifies an Atom link relation name that determines which **atom:link** element the field value will be taken from. This attribute is only used when the **select** attribute is set to **link**.

2.7.6 Collection field (Simplified)

Defines a collection field, which can be used to hold a reference to an Atom feed entry.

Collection field values are represented in Content Studio by graphic components called **tokens**.

Syntax

```
<field
    name="NCName"
    type="collection"
    src="text"
    mime-type="text"
    ( select="(content|title|locator)" | select="link" linkrel="text" )
  >
  <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
</field>
```

Child Elements

constraints: section 2.3, **parameter**: section 2.16.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="collection"
Specifies that this field is a collection field.

src="*text*"
Contains a URI that references a resource with an **application/atom+xml;type=feed** representation. The entries in the referenced feed are presented as a set of alternatives (a drop-down field in Content Studio) from which the user can choose one value.

mime-type="*text*"
Defines the MIME type of the field.

select="(content|title|locator)"
Specifies which part of the selected Atom entry's content is to be stored as the field value.

Allowed values are:

content
The content of the selected Atom entry's **atom:content** element is to be stored as the field value.

title
The content of the selected Atom entry's **atom:title** element is to be stored as the field value.

locator
The **href** attribute of one of the selected Atom entry's **viz:locator** elements is to be stored as the field value. There are two points to note in this case:

1. **viz:locator** is a proprietary Vizrt Atom extension element belonging to the **http://www.vizrt.com/types** namespace.
2. An Atom entry may contain several **viz:locator** elements. The field value is taken from the one with the correct MIME type: that is, the one with a **type** attribute that matches this element's **mime-type** attribute.

select="link"
Specifies that the **href** of one the selected Atom entry's **atom:link** elements is to be stored as the field value. The specific **atom:link** to be used is determined by the **linkrel** attribute.

linkrel="*text*"
Specifies an Atom link relation name that determines which **atom:link** element the field value will be taken from. This attribute is only used when the **select** attribute is set to **link**.

2.7.7 Complex field

Defines a **complex** field. A complex field is composed of one or more simple fields.

Syntax

```
<field
    name="NCName"
    type="complex"
  >
  <array/>?
    <complex>...</complex>

    <required>...</required>?
    ANY-FOREIGN-ELEMENT*
</field>
```

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="complex"
Specifies that this field is a complex field.

2.7.8 Date field

Defines a date field, which may contain a date/time value. The date/time value is stored as a UTC time in ISO-8601 format - that is, *YYYY-MM-DD*`'T'` *HH:mm:ss*`'Z'`.

Syntax

```
<field
    name="NCName"
    type="date"
  >
  <array/>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <options>...</options>?
</field>
```

Child Elements

array: section 2.1, **constraints**: section 2.3, **parameter**: section 2.16, **options**: section 2.14.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="date"
Specifies that this field is a date field.

2.7.9 Date field (Simplified)

Defines a date field, which may contain a date/time value. The date/time value is stored as a UTC time in ISO-8601 format - that is, *YYYY-MM-DD*`'T'` *HH:mm:ss*`'Z'`.

Syntax

```
<field
    name="NCName"
    type="date"
  >
  <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
</field>
```

Child Elements

constraints: section 2.3, **parameter**: section 2.16.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="date"
Specifies that this field is a date field.

2.7.10 Enumeration field

Defines an enumeration field. An enumeration field has a number of predefined values from which the user can choose. It can be configured to accept either a single choice or multiple choices using the **multiple** attribute.

Syntax

```
<field
    name="NCName"
    type="enumeration"
    multiple="(true|false)"?
  >
  <array/>?
    <enumeration>...</enumeration>+
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <options>...</options>?
</field>
```

Child Elements

array: section 2.1, **enumeration**: section 2.6, **constraints**: section 2.3, **parameter**: section 2.16, **options**: section 2.14.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.3.1).

Examples

- This example defines an enumeration field that allows the Content Studio user to select an option from a list.

```
<field type="enumeration" name="review-type">
  <ui:label>Review Type</ui:label>
  <ui:description>Select the required type</ui:description>
  <enumeration value="film"/>
  <enumeration value="play"/>
  <enumeration value="book"/>
  <enumeration value="game"/>
</field>
```

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="enumeration"`
Specifies that this field is an enumeration field.

`multiple="(true|false)"` (optional)
If set to **`true`** then the field will accept multiple user choices; it is displayed as a multiple-choice list in Content Studio. If set to **`false`** or unspecified then the field will only accept a single choice; it is displayed as a combo box in Content Studio.

2.7.11 Enumeration field (Simplified)

Defines an enumeration field. An enumeration field has a number of predefined values from which the user can choose. It can be configured to accept either a single choice or multiple choices using the **`multiple`** attribute.

Syntax

```
<field
    name="NCName"
    type="enumeration"
    multiple="(true|false)"?
  >
  <enumeration>...</enumeration>+
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
</field>
```

Child Elements

`enumeration`: section 2.6, **`constraints`**: section 2.3, **`parameter`**: section 2.16.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.3.1).

Examples

- This example defines an enumeration field that allows the Content Studio user to select an option from a list.

```
<field type="enumeration" name="review-type">
  <ui:label>Review Type</ui:label>
  <ui:description>Select the required type</ui:description>
  <enumeration value="film"/>
  <enumeration value="play"/>
  <enumeration value="book"/>
  <enumeration value="game"/>
</field>
```

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="enumeration"
Specifies that this field is an enumeration field.

multiple="(true|false)" (optional)
If set to **true** then the field will accept multiple user choices; it is displayed as a multiple-choice list in Content Studio. If set to **false** or unspecified then the field will only accept a single choice; it is displayed as a combo box in Content Studio.

2.7.12 Inherited field

Defines an **inherited field**. An inherited field inherits all its characteristics (type, constraints, default value and so on) from another named **field**. These characteristics cannot be overridden: in other words, if you specify the **inherits-from** attribute, you cannot specify **type** or **mime-type** attributes.

If an inherited field is left empty, the Content Engine will try to retrieve a value from the **field** it inherits from.

Syntax

```
<field
    name="NCName"
    inherits-from="text"
  >
  ANY-FOREIGN-ELEMENT*
</field>
```

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

inherits-from="*text*"
Specifies the **field** element from which this field is to inherit its characteristics. Enter the name of another field in the same **content-type**. The field you specify may not itself be an inherited field.

> If the field you specify contains any elements from foreign namespaces (such as a **label** element from the **interface-hints** namespace), then these will be inherited along with the field's other characteristics. However, you can override these inherited foreign elements by adding the same elements to this **field**.

2.7.13 Link field

Defines a link field. A link field is intended to contain the URI of some resource you want to make use of in some way. Link fields are most commonly used to contain references to binary objects such as images and media files; all binary objects in content items are referenced in this way.

> Note that a **content-type** element may only contain **one** link field.

Syntax

```
<field
    name="NCName"
    type="link"
  >
  <array/>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <relation>...</relation>
    <options>...</options>?
</field>
```

Child Elements

array: section 2.1, **constraints**: section 2.3, **parameter**: section 2.16, **relation**: section 2.19, **options**: section 2.14.

Only one form of the **constraints** element may be used: **Link constraints** (section 2.3.2).

Examples

- This example defines a link field used to contain references to image objects.

```
<field name="binary" type="link">
  <relation>com.escenic.edit-media</relation>
  <constraints>
    <mime-type>image/jpeg</mime-type>
    <mime-type>image/png</mime-type>
  </constraints>
</field>
```

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="link"
Specifies that this field is a link field.

2.7.14 Link field (Simplified)

Defines a link field. A link field is intended to contain the URI of some resource you want to make use of in some way. Link fields are most commonly used to contain references to binary objects such as images and media files; all binary objects in content items are referenced in this way.

> Note that a **content-type** element may only contain **one** link field.

Syntax

```
<field
    name="NCName"
    type="link"
  >
  <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <relation>...</relation>
```

```
</field>
```

Child Elements

constraints: section 2.3, **parameter**: section 2.16, **relation**: section 2.19.

Only one form of the **constraints** element may be used: **Link constraints** (section 2.3.2).

Examples

- This example defines a link field used to contain references to image objects.

```
<field name="binary" type="link">
  <relation>com.escenic.edit-media</relation>
  <constraints>
    <mime-type>image/jpeg</mime-type>
    <mime-type>image/png</mime-type>
  </constraints>
</field>
```

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="link"
Specifies that this field is a link field.

2.7.15 Number field

Defines a number field, which may contain any numeric value (including decimals).

Syntax

```
<field
    name="NCName"
    type="number"
  >
  <array/>?
    <format>...</format>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <options>...</options>?
</field>
```

Child Elements

array: section 2.1, **format**: section 2.9, **constraints**: section 2.3, **parameter**: section 2.16, **options**: section 2.14.

Only one form of the **constraints** element may be used: **Number constraints** (section 2.3.3).

Examples

- This example defines a constrained numeric field in which only numbers between 1 and 6 are allowed.

```
<field type="number" name="score">
  <ui:label>Score</ui:label>
  <ui:description>Enter your rating</ui:description>
  <constraints>
    <minimum>1</minimum>
    <maximum>6</maximum>
  </constraints>
</field>
```

Attributes

`name="`*`NCName`*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="number"`
Specifies that this field is a number field.

2.7.16 Number field (Simplified)

Defines a number field, which may contain any numeric value (including decimals).

Syntax

```
<field
    name="NCName"
    type="number"
  >
  <format>...</format>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
</field>
```

Child Elements

`format`: section 2.9, **`constraints`**: section 2.3, **`parameter`**: section 2.16.

Only one form of the **`constraints`** element may be used: **Number constraints** (section 2.3.3).

Examples

- This example defines a constrained numeric field in which only numbers between 1 and 6 are allowed.

```
<field type="number" name="score">
  <ui:label>Score</ui:label>
  <ui:description>Enter your rating</ui:description>
  <constraints>
    <minimum>1</minimum>
    <maximum>6</maximum>
  </constraints>
</field>
```

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="number"
Specifies that this field is a number field.

2.7.17 Schedule field

Defines a schedule field. A schedule field contains a schedule start and end date, an event start and end time and an optional set of recurrence rules.

Syntax

```
<field
    name="NCName"
    type="schedule"
  >
  <array/>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <options>...</options>?
</field>
```

Child Elements

array: section 2.1, **constraints**: section 2.3, **parameter**: section 2.16, **options**: section 2.14.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

name="*NCName*"
The name of the **field** element. Note that no two **field**s belonging to the same **panel** may have the same name.

type="schedule"
Defines the type of the field.

2.7.18 Schedule field (Simplified)

Defines a schedule field. A schedule field contains a schedule start and end date, an event start and end time and an optional set of recurrence rules.

Syntax

```
<field
    name="NCName"
    type="schedule"
  >
  <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
</field>
```

Child Elements

`constraints`: section 2.3, **`parameter`**: section 2.16.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="schedule"`
Defines the type of the field.

2.7.19 URI field

Defines a URI field that may contain any valid URI. URI syntax is defined by RFC 2396 and RFC 2732.

Syntax

```
<field
    name="NCName"
    type="uri"
  >
  <array/>?
    <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
    <options>...</options>?
</field>
```

Child Elements

`array`: section 2.1, **`constraints`**: section 2.3, **`parameter`**: section 2.16, **`options`**: section 2.14.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="uri"`
Defines the type of the field.

2.7.20 URI field (Simplified)

Defines a URI field that may contain any valid URI. URI syntax is defined by RFC 2396 and RFC 2732.

Syntax

```
<field
    name="NCName"
    type="uri"
  >
```

```
  <constraints>...</constraints>?
    ANY-FOREIGN-ELEMENT*
    <parameter/>*
</field>
```

Child Elements

`constraints`: section 2.3, **`parameter`**: section 2.16.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.3.1).

Attributes

`name="`*NCName*`"`
The name of the **`field`** element. Note that no two **`field`**s belonging to the same **`panel`** may have the same name.

`type="uri"`
Defines the type of the field.

2.8 field-group

Defines a field group. A field group is a convenience element that enables you to:

Re-use field definitions
Instead of adding many identical field definitions (**`field`** elements) to different panels you can create a field group containing the field definition, and then just add references (**`ref-field-group`** elements) to panels.

Group related field definitions
You can then add whole sets of related field definitions to a panel with a single **`ref-field-group`** element.

Syntax

```
<field-group
    name="NCName"
  >
  <field>...</field>*
</field-group>
```

Child Elements

`field`: section 2.7.

The following forms of the **`field`** element may be used: **Complex field** (section 2.7.7), **Basic field** (section 2.7.1), **Boolean field** (section 2.7.3), **URI field** (section 2.7.19), **Schedule field** (section 2.7.17), **Number field** (section 2.7.15), **Link field** (section 2.7.13), **Enumeration field** (section 2.7.10), **Date field** (section 2.7.8), **Collection field** (section 2.7.5), **Inherited field** (section 2.7.12).

Examples

- This example defines a group consisting of two fields.

```
<field-group name="review-fields">
```

```
  <field type="enumeration" name="review-type">
    <ui:label>Review Type</ui:label>
    <ui:description>Select the required type</ui:description>
    <enumeration value="film"/>
    <enumeration value="play"/>
    <enumeration value="book"/>
    <enumeration value="game"/>
  </field>
  <field type="number" name="score">
    <ui:label>Score</ui:label>
    <ui:description>Enter your rating</ui:description>
    <constraints>
      <minimum>1</minimum>
      <maximum>6</maximum>
    </constraints>
  </field>
</field-group>
```

Attributes

name="*NCName*"
The name of the **field-group** element.

2.9 format

Specifies a number format defining how the field contents are to be formated. You may enter any valid java.text.DecimalFormat format specification.

Syntax

```
<format>
  text
</format>
```

2.10 maxchars

Specifies the maximum number of characters a basic **field** is allowed to contain. A negative or zero value represents no maxchars constraint for the **field**

Syntax

```
<maxchars>
  integer
</maxchars>
```

2.11 maximum

The maximum value that may be entered in a numeric **field**.

Syntax

```
<maximum>
  integer
</maximum>
```

2.12 mime-type

An allowed MIME type for the binary data referenced in a link **`field`**.

Syntax

```
<mime-type>
  string
</mime-type>
```

2.13 minimum

The minimum value that may be entered in a numeric **`field`**.

Syntax

```
<minimum>
  integer
</minimum>
```

2.14 options

Contains **`field`** elements that can be used to set options governing attributes of the owning **`field`** element. An option might, for example, be used to allow Content Studio users to specify a color to be associated with a particular field. The template developer can then use this color in rendering the content of the field. Option fields defined in this way are displayed on the **Options** tab in Content Studio's content item editor.

This element can also appear as a child of **`group`** and **`area`** elements in a **`layout-group`** resource file and has the same function, allowing you to associate formatting options with section page groups and areas.

Syntax

```
<options
    scope="(current|items)"?
  >
  <field>...</field>*
</options>
```

Child Elements

`field`: section 2.7.

The following forms of the **`field`** element may be used: **Basic field (Simplified)** (section 2.7.2), **Boolean field (Simplified)** (section 2.7.4), **URI field (Simplified)** (section 2.7.20), **Schedule**

field (Simplified) (section 2.7.18), **Number field (Simplified)** (section 2.7.16), **Link field (Simplified)** (section 2.7.14), **Enumeration field (Simplified)** (section 2.7.11), **Date field (Simplified)** (section 2.7.9), **Collection field (Simplified)** (section 2.7.6).

Attributes

`scope="(current|items)" (optional)`
If the **`options`** element's owner is an **`area`** element in a **`layout-group`** resource file, then this attribute can be set to determine whether the options it defines apply to the area itself or to the items it contains.

Allowed values are:

`current`
The options apply to the **`options`** element's owning **`area`**.

`items`
The options apply to the items contained in the **`options`** element's owning **`area`**.

2.15 panel

eDefines a panel. A panel is a set of fields that are grouped together and displayed on a single tab in a Content Studio content editor.

Syntax

```
<panel
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT*
    <field>...</field>*
    <ref-field-group/>*
</panel>
```

Child Elements

`field`: section 2.7, **`ref-field-group`**: section 2.17.

The following forms of the **`field`** element may be used: **Complex field** (section 2.7.7), **Basic field** (section 2.7.1), **Boolean field** (section 2.7.3), **URI field** (section 2.7.19), **Schedule field** (section 2.7.17), **Number field** (section 2.7.15), **Link field** (section 2.7.13), **Enumeration field** (section 2.7.10), **Date field** (section 2.7.8), **Collection field** (section 2.7.5), **Inherited field** (section 2.7.12).

Examples

- This example defines a simple panel containing three **`field-group`** references.

```
<panel name="main">
  <ui:label>Main Content</ui:label>
  <ui:description>The main content fields</ui:description>
  <ref-field-group name="title"/>
  <ref-field-group name="summary"/>
  <ref-field-group name="body"/>
</panel>
```

- This example defines a panel containing **`field-group`** definitions rather than references.

```
<panel name="main">
  <ui:label>Image content</ui:label>
  <field mime-type="text/plain" type="basic" name="name">
    <ui:label>Name</ui:label>
    <ui:description>The name of the image</ui:description>
    <constraints>
      <required>true</required>
    </constraints>
  </field>
  <field mime-type="text/plain" type="basic" name="description">
    <ui:label>Description</ui:label>
  </field>
  <field mime-type="text/plain" type="basic" name="alttext">
    <ui:label>Alternative text</ui:label>
  </field>
  <field name="binary" type="link">
    <relation>com.escenic.edit-media</relation>
    <constraints>
      <mime-type>image/jpeg</mime-type>
      <mime-type>image/png</mime-type>
    </constraints>
  </field>
</panel>
```

Attributes

name="*NCName*"
The name of the **panel** element.

2.16 parameter

Defines a parameter to be associated with a **content-type**, **panel** or **field**. The parameter has a fixed value defined in the resource file. It is not displayed in Content Studio or used by Content Studio in any way. It can, however be retrieved by template developers and used for a variety of purposes.

Syntax

```
<parameter
    name="NCName"
    value="text"
  />
```

Attributes

name="*NCName*"
The name of the **parameter** element.

value="*text*"
The value of this **parameter**.

2.17 ref-field-group

Defines a reference to a **field-group**. **ref-field-group** lets you re-use **field-group** definitions.

Syntax

```
<ref-field-group
    name="NCName"
  />
```

Examples

- ```
  <ref-field-group name="title"/>
  ```

Attributes

name="*NCName*"
The name of the **ref-field-group** element.

2.18 ref-relation-type-group

Defines a reference to a **relation-type-group**. **ref-relation-type-group** lets you re-use **relation-type-group** definitions.

Syntax

```
<ref-relation-type-group
    name="NCName"
  />
```

Examples

- ```
  <ref-relation-type-group name="attachments"/>
  ```

Attributes

name="*NCName*"
The name of the **ref-relation-type-group** element.

2.19 relation

Defines the relationship between the resource referenced by a link **field** and the content item it contains. The only **relation** value supported by the Content Engine core is **com.escenic.edit-media**. This indicates that the resource referenced in the link field is a binary object of some kind (an image, movie, sound file, PDF or Word document, etc.).

Content Engine plug-ins may define other values for this element.

Syntax

```
<relation>
  text
</relation>
```

Examples

- ```
  <relation>com.escenic.edit-media</relation>
  ```

2.20 relation-type

Defines a named relation type. Relation types allow you to classify relations between content items. You might, for example, define a **`content-type`** with the **`relation`**s "article-image" (intended for an image to be displayed with a content item's body) and "teaser-image" (intended for an image to be displayed in a content item's teaser). In Content Studio content item editors these relation types will appear as two **drop zones** labelled "article-image" and "teaser-image". A drop zone is an area in which the Content Studio user can drop content items (in this case the images he wants to appear in these locations).

Syntax

```
<relation-type
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT*
</relation-type>
```

Examples

- ```
  <relation-type name="images">
    <ui:label>Pictures</ui:label>
  </relation-type>
  ```

Attributes

`name="`*`NCName`*`"`
The name of the **`relation-type`** element.

2.21 relation-type-group

Defines a relation type group. A relation type group is a convenience element that enables you to:

Re-use relation type definitions
Instead of adding many identical relation type definitions (**`relation-type`** elements) to different panels you can create a relation type group containing the relation type definition, and then just add references (**`relation-type-group`** elements) to panels.

Group related relation type definitions
You can then add whole sets of relevant relation type definitions to a panel with a single **`ref-relation-type-group`** element.

Syntax

```
<relation-type-group
    name="NCName"
  >
  <relation-type>...</relation-type>+
</relation-type-group>
```

Examples

- ```
  <relation-type-group name="attachments">
    <relation-type name="images">
      <ui:label>Pictures</ui:label>
    </relation-type>
    <relation-type name="stories">
      <ui:label>Stories</ui:label>
    </relation-type>
  </relation-type-group>
  ```

Attributes

`name="`*`NCName`*`"`
The name of the **`relation-type-group`** element.

2.22 rep:crop

Indicates that the source image will be cropped if necessary to meet the image representation's output requirements (that is, the width:height ratio implied by the output element's width and height attributes).

Note that this element is currently required (meaning that image representations will always be cropped if necessary).

This element belongs to the **`http://xmlns.escenic.com/2009/representations`** namespace. The conventional prefix for this namespace is **`rep`**.

Syntax

```
<rep:crop/>
```

2.23 rep:output

This element can appear in a number of different forms, described in the following sections.

2.23.1 Derived Image Version rep:output

Defines the required characteristics of a "derived image version"-style image representation. One of the **`width`**/**`height`** attributes must be specified, but not both.

This element belongs to the **`http://xmlns.escenic.com/2009/representations`** namespace. The conventional prefix for this namespace is **`rep`**.

Syntax
```

```
<rep:output
    ( width="positiveInteger" | height="positiveInteger" )
  />
```

Attributes

`width="`*`positiveInteger`*`"`
Specifies the required width of this image representation in pixels.

`height="`*`positiveInteger`*`"`
Specifies the required height of this image representation in pixels.

2.23.2 Image Version rep:output

Defines the required characteristics of an "image-version"-style image representation. One or both of the **`width`**/**`height`** attributes must be specified. If both are specified, then the crop mask displayed in Content Studio for this representation will have a fixed aspect ratio. If only **`width`** or **`height`** is specified, then users will be able to reshape the crop mask displayed in Content Studio as well as resize it.

> This element belongs to the **`http://xmlns.escenic.com/2009/representations`** namespace. The conventional prefix for this namespace is **`rep`**.

Syntax

```
<rep:output
    ( width="positiveInteger" | height="positiveInteger" | width="positiveInteger"
 height="positiveInteger" )
  />
```

Attributes

`width="`*`positiveInteger`*`"`
Specifies the required width of this image representation in pixels.

`height="`*`positiveInteger`*`"`
Specifies the required height of this image representation in pixels.

`width="`*`positiveInteger`*`"`
Specifies the required width of this image representation in pixels.

`height="`*`positiveInteger`*`"`
Specifies the required height of this image representation in pixels.

2.24 rep:representation

This element can appear in a number of different forms, described in the following sections.

2.24.1 Custom rep:representation

Contains the parameters needed to define an image representation using a custom method. The contents of this element are undefined. The attributes specified with this element must include a **`name`** attribute.

This element belongs to the **http://xmlns.escenic.com/2009/representations** namespace. The conventional prefix for this namespace is **rep**.

Syntax

```
<rep:representation
    ANY-ATTRIBUTE*
  >
  (ANYTHING|text)*
</rep:representation>
```

Attributes

ANY-ATTRIBUTE

2.24.2 Derived Image Version rep:representation

Defines a secondary "image-version" representation that is based on another representation. A representation of this kind takes the image defined by the representation referenced in its **based-on** attribute, and generates a new representation by applying the constraints defined in its own child **rep:output** element.

A derived representation may not be based on another derived representation.

Such representations do not show up in Content Studio, as they do not need a crop mask from the user, as the mask of the representation on which it is based is used instead.

This element belongs to the **http://xmlns.escenic.com/2009/representations** namespace. The conventional prefix for this namespace is **rep**.

Syntax

```
<rep:representation
    name="NCName"
    based-on="text"
  >
  ANY-FOREIGN-ELEMENT*?
    <rep:output/>
</rep:representation>
```

Child Elements

rep:output: section 2.23.

Only one form of the **rep:output** element may be used: **Derived Image Version output** (section 2.23.1).

Attributes

name="*NCName*"
The name of the **representation** element.

based-on="*text*"
Specifies the **representation** element on which which this representation is to be based. Enter the name of a sibling **representation** element (that is, one that is a child of the

same **representations** element). The representation you specify must be a top-level representation (that is, not another derived representation).

2.24.3 Image Version rep:representation

Contains the parameters needed to define an "image-version"-style image representation.

> This element belongs to the **http://xmlns.escenic.com/2009/representations** namespace. The conventional prefix for this namespace is **rep**.

Syntax

```
<rep:representation
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT*?
    <rep:output/>

    <rep:crop/>

    <rep:resize/>
</rep:representation>
```

Child Elements

rep:output: section 2.23, **rep:crop**: section 2.22, **rep:resize**: section 2.26.

Only one form of the **rep:output** element may be used: **Image Version output** (section 2.23.2).

Examples

- This example shows how **representation** elements are used in basic **field**s to define image crop masks.

```
<field mime-type="application/json" type="basic" name="representations">
  <ui:label>Image Versions</ui:label>
  <rep:representations type="image-versions">
    <rep:representation name="thumbnail">
      <rep:output width="100" height="100"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
    <rep:representation name="narrow">
      <rep:output width="500" height="400"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
    <rep:representation name="wide">
      <rep:output width="1000" height="800"/>
      <rep:crop/>
      <rep:resize/>
    </rep:representation>
  </rep:representations>
</field>
```

Attributes

name="*NCName*"
The name of the **representation** element.

2.25 rep:representations

This element can appear in a number of different forms, described in the following sections.

2.25.1 Custom rep:representations

Defines a set of different versions of an image that are created and maintained using a custom mechanism.

> This element belongs to the **http://xmlns.escenic.com/2009/representations** namespace. The conventional prefix for this namespace is **rep**.

Syntax

```
<rep:representations
    type="NCName"
  >
  <rep:representation>...</rep:representation>+
</rep:representations>
```

Child Elements

rep:representation: section 2.24.

Only one form of the **rep:representation** element may be used: **Custom representation** (section 2.24.1).

Attributes

type="*NCName*"
An identifier for the custom mechanism used to create image representations.

2.25.2 Image Version rep:representations

Defines a set of different versions of an image that are created and maintained using the standard Content Engine **image-versions** mechanism. This mechanism allows the Content Studio user to define different cropped and resized versions of an image.

> This element's parent **field**'s **mime-type** attribute must be set to **application/json**.

> This element belongs to the **http://xmlns.escenic.com/2009/representations** namespace. The conventional prefix for this namespace is **rep**.

Syntax

```
<rep:representations
    type="image-versions"
  >
  <rep:representation>...</rep:representation>+
</rep:representations>
```

Child Elements

`rep:representation`: section 2.24.

The following forms of the **`rep:representation`** element may be used: **Image Version representation** (section 2.24.3), **Derived Image Version representation** (section 2.24.2).

Attributes

`type="image-versions"`
Identifies the contents of this element as "image-version"-style image representations.

2.26 rep:resize

Indicates that the cropped source image will be resized if necessary to meet the image representation's output requirements (that is, the exact width and height specified in the output element's width and height attributes).

Note that this element is currently required (meaning that image representations will always be resized if necessary).

> This element belongs to the **`http://xmlns.escenic.com/2009/representations`** namespace. The conventional prefix for this namespace is **`rep`**.

Syntax

```
<rep:resize/>
```

2.27 required

If set to **`true`** then the Content Studio user is required to enter a value in this **`field`**.

Syntax

```
<required>
  (true|false)
</required>
```

2.28 summary

Defines a **`content-type`**'s summary. A summary is a set of **`field`**s intended to be used when content items appear as:

- Relations in other content items
- Teasers on section pages

A **`summary`** usually contains a subset of the **`content-type`**'s ordinary fields. It can, however, also contain fields that are specifically intended for use on the summary.

The content of summary fields can be locally overridden in the relations/teasers that use the content item. That is, when a Content Studio user adds a content item to a section page (for example), she can modify the fields in the teaser without affecting the source content item's fields.

Syntax

```
<summary>
  ANY-FOREIGN-ELEMENT*
    <field>...</field>*
    <ref-relation-type-group/>*
</summary>
```

Child Elements

`field`: section 2.7, **`ref-relation-type-group`**: section 2.18.

The following forms of the **`field`** element may be used: **Complex field** (section 2.7.7), **Basic field** (section 2.7.1), **Boolean field** (section 2.7.3), **URI field** (section 2.7.19), **Schedule field** (section 2.7.17), **Number field** (section 2.7.15), **Link field** (section 2.7.13), **Enumeration field** (section 2.7.10), **Date field** (section 2.7.8), **Collection field** (section 2.7.5), **Inherited field** (section 2.7.12).

Examples

- This example defines a simple summary containing two **`field`**s.

```
<summary>
  <ui:label>Content Summary</ui:label>
  <field name="title" type="basic" mime-type="text/plain"/>
  <field name="summary" type="basic" mime-type="text/plain"/>
</summary>
```

2.29 url

Defines a pattern from which relative URLs can be generated. The pattern is used to generate URLs for content items of the type represented by this element's parent **`content-type`**. By default, the final component of a content item's URL is generated from the content item's id, prefixed by the string **`article`** and followed by the suffix **`.ece`** - for example:

`article1234.ece`

You can use this element to specify a pattern from which more meaningful URLs (sometimes called "pretty" URLs) can be generated.

The pattern must be specified as a combination of literal characters and the following placeholders:

`{d}`
: Replaced by the content item's publication day of the month, in the format 1,2,...9,10 etc.

`{dd}`
: Replaced by the content item's publication day of the month, in the format 01,02,...09,10 etc.

`{MM}`
: Replaced by the content item's publication month, in the format 01,02,...12.

{MMM}
Replaced by the content item's publication month, in the format Jan, Feb, etc. The language of the month names used are determined by the **locale** attribute (if specified).

{MMMM}
Replaced by the content item's publication month, in the format January, February, etc. The language of the month names used are determined by the **locale** attribute (if specified).

{YY}
Replaced by the content item's publication year, in the format in the format 01,02, etc.

{YYYY}
Replaced by the content item's publication year, in the format in the format 2001,2002, etc.

{field.*name*}
Replaced by the content of the content item field specified with *name*. The specified field must have a **mime-type** of **text/plain**. Any spaces in the field content are replaced by the character specified with the **white-space-replacement** attribute. If this attribute is not specified, then any white spaces are replaced by _ (underscore) characters. Any multibyte Unicode characters are automatically URL-encoded.

{id}
Replaced by the content item's id.

{counter}
Replaced by an integer counter used to distinguish otherwise identical URLs from one another. Note that a counter will always be appended to otherwise identical URLs even if you do not use this placeholder. Using this placeholder just lets you control the position of the counter if you do not want it to appear at the end of the URL.

If you want to use multibyte Unicode characters or other special characters in the literal parts of your URL pattern, then you must enter them as URL-encoded strings - the will not encode them for you.

> The Content Engine cannot generate URLs with more than 255 characters. If your URL pattern results in a URL that is longer than this limit, then the last **{field.*name*}** in the pattern will be truncated to fit the limit. Using field content in URL patterns can easily result in very long URLs if the fields contain multi-byte Unicode characters, as each byte is converted to a three-character sequence. A single 3-byte character, for example, will result in a 9-character URL sequence.

Syntax

```
<url
    locale="..."?
    white-space-replacement="..."?
  >
  text
</url>
```

Examples

- This example defines the structure of the URL generated for content items of the type defined by the **url** element's parent **content-type**. It will generate the following relative URL for a content item with the title "Pretty URL" and a publication date of 30.05.2012:

  2012/05/30/article1231875438.ece/Pretty_URL

  ```
  <url>{yyyy}/{MM}/{dd}/article{id}.ece/{field.title}</url>
  ```

- This example will generate the following relative URL for a content item with the title "The story title" and a publication date of 30.05.2012:

  2012/05/30/the-story-title-1231875438.ece

  ```
  <url white-space-replacement="-">{yyyy}/{MM}/{dd}/{field.title}-{id}.ece</url>
  ```

Attributes

`locale="..."` (optional)
Specifies the locale to use for formatting the URL. The locale setting only affects **`{MMM}`** and **`{MMMM}`** placeholders. The locale specification is a string of the form:

language or
language_region or
language_region_variant

where:

language
is a valid ISO language code. These codes are lower-case, two-letter codes as defined by ISO-639 - for example: **`en`**, **`no`**, **`zh`**, **`de`**, **`sv`**.

country
is a valid ISO country code. These codes are upper-case, two-letter codes as defined by ISO-639 - for example: **`GB`**, **`US`**, **`DE`**, **`SE`**.

variant
is a vendor or browser-specific code. For example, use **`WIN`** for Windows, **`MAC`** for Macintosh, and **`POSIX`** for POSIX.

For more detailed information, see the documentation of the Java Locale class.

`white-space-replacement="..."` (optional)
Specifies a character or sequence of characters to be used to replace white space sequences in the generated URL, for example **`-`** or **`_`**. If this attribute is not specified then **`_`** is used as a replacement character by default. Note that the specified character(s) are only used to replace white space inside fields specified with the **`{field.`***name***`}`** placeholder.

2.30 well-formed

If set to **`true`** then the content of this **`field`** must be **`well-formed`** XML. This means that:

- There must only be one root node.
- All start tags must be matched by corresponding end tags.
- All elements must be perfectly nested, with no overlapping.

Syntax

```
<well-formed>
  (true|false)
</well-formed>
```

3 image-versions

> Use of the **image-version** resource is deprecated. You should use **representation** elements in the **content-type** resource instead, where possible. (In other words, you should **only** use **image-version** if you require functionality that cannot be provided using **representation** elements.)

The **image-versions** schema defines the content of the Escenic **image-version** publication resource. The purpose of the **image-version** resource is to define the **image versions** that are to be used in a publication. It is often the case that several versions of images are required for use in different contexts: thumbnails in teasers and one or more larger versions in articles, for example.

The image-versions contains an **originalVersion** element that defines the **label**s or names used to identify the original versions of images, plus a number of **version** elements defining the additional versions that are required. Each **version** element defines the **label**s or names used to identify the version plus other attributes such as size and format.

Namespace URI

The namespace URI of the **image-versions** schema is **http://xmlns.escenic.com/2008/image-versions**.

Root Element

The root of an **image-versions** file must be an **imageDef** element.

3.1 fallback

Version generation is based on the assumption that the original version of an image is larger than the image size specified with **maxHeight** and **maxWidth**. **fallback** specifies what to do when this is not the case: if, for example, the original image is 160x200 and the maximum size of this version is defined as 300x200.

Syntax

```
<fallback
    operation="(copy|skip|resize)"
  />
```

Attributes

operation="(copy|skip|resize)"
Specifies the fallback action to be taken.

Allowed values are:

copy (default)
Use the original version with no scaling applied (recommended).

skip
Do not generate this version of the image.

resize
Scale the image up to the required size. This may result in a poor quality image.

3.2 format

Specifies the format to use for this image version.

Syntax

```
<format
    name="(jpeg|jpg|gif|png|wbmp)"?
    ( quality="..." | compression="..." )?
    sharpen="..."?
  />
```

Attributes

name="(jpeg|jpg|gif|png|wbmp)" (optional)
Allowed values are:

jpeg (default)
JPEG is a lossy compressed image format, mainly intended for photographic images.

jpg
A synonym for **jpeg**.

gif
GIF is a compact image format, mainly intended for non-photographic images such as charts and diagrams).

png
PNG is a compact image format, mainly intended for non-photographic images such as charts and diagrams.

wbmp
WBMP is a monochrome image format intended for low-bandwidth mobile applications. WBMP is the standard image format for use in WAP applications.

quality="..." (optional)
The image quality level to be applied when generating this version. It is only used when **format** is set to **JPEG**. The value specified must be a number between `0.0` (lowest quality) and `1.0` (highest quality). The default is `0.7`.

compression="..." (optional)

> Deprecated. This attribute is a synonym for **quality**, but should not be used: use **quality** instead.

sharpen="..." (optional)
The **sharpening** level to be applied when generating this version. Sharpening is an image processing algorithm that can improve blurred or unclear images. The value specified must be a number between `0.0` (no sharpening) and `1.0` (maximum sharpening). The default is `0.0` (no sharpening).

3.3 imageDef

The root element of the **image-versions** file.

Syntax

```
<imageDef>
  <originalVersion>...</originalVersion>

    <version>...</version>*
</imageDef>
```

3.4 label

The label used to identify this image version in applications. **label** may have several labels in different languages. If so, the **lang** attribute must be used to specify the language of the label.

Syntax

```
<label
    lang="..."?
  >
  text
</label>
```

Attributes

lang="..." (optional)
The language of the **label**. You should use ISO-639 format language IDs. For a complete list of these IDs see http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt.

3.5 maxHeight

The maximum allowed height for this image version. If the height of the original version of an image is greater than this value, then the image will be scaled down to this height. If **maxWidth** is also specified, then the height may be set to less than **maxHeight** in order to preserve the image's width-height ratio.

The **fallback** element specifies what should happen if the height of the original version of an image is **smaller** than this value.

Syntax

```
<maxHeight
    pix="..."?
  />
```

Attributes

pix="..." (optional)
The **maxHeight** value, specified in pixels. The value you specify must be a whole number.

3.6 maxWidth

The maximum allowed width for this image version. If the width of the original version of an image is greater than this value, then the image will be scaled down to this width. If **`maxHeight`** is also specified, then the width may be set to less than **`maxWidth`** in order to preserve the image's width-height ratio.

The **`fallback`** element specifies what should happen if the width of the original version of an image is **smaller** than this value.

Syntax

```
<maxWidth
    pix="..."?
  />
```

Attributes

`pix="..."` (optional)
The **`maxWidth`** value, specified in pixels. The value you specify must be a whole number.

3.7 originalVersion

Holds the **`id`** and **`label`**s used to identify the original unscaled, uncompressed version of images.

Syntax

```
<originalVersion
    id="..."
  >
  <label>...</label>+
</originalVersion>
```

Attributes

`id="..."`
The internal identifier of the **`originalVersion`**. This is the identifier used in JSP and Java code. It must be unique and must not contain any spaces or special characters.

3.8 parameter

> Not currently used.

Syntax

```
<parameter
    name="..."
    value="..."?
  />
```

Attributes

`name="..."`
The name of the **`parameter`**.

`value="..."` (optional)
The value of the **`parameter`**.

3.9 pluginGenerator

Not currently used.

Syntax

```
<pluginGenerator
    class="..."
  >
  <parameter/>*
</pluginGenerator>
```

Attributes

`class="..."`
Not currently used.

3.10 version

Defines one of the image versions required by the publication.

Syntax

```
<version
    id="..."
  >
  <label>...</label>+
    (<maxWidth/>|<maxHeight/>|<maxWidth/> <maxHeight/>)

    <fallback/>?
    <format/>?
    <pluginGenerator>...</pluginGenerator>?
</version>
```

Attributes

`id="..."`
The internal identifier of the **`version`**. This is the identifier used in JSP and Java code. It must be unique and must not contain any spaces or special characters.

4 layout-group

The **layout-group** schema defines the structure of the allowed Escenic **layout-group** publication resource. The purpose of the **layout-group** resource is to define a set of layouts for use on Escenic section pages. These layouts are composed of **group** and **area** elements, and include external references to **teaser-type**s (defined in the **teaser-type** resource) and **content-type**s (defined in the **content-type** resource).

An **area** is a named location on a section page in which a sequence of teasers can be displayed. The actual size and location of an **area** is not specified in the **layout-group** resource. Physical layout is defined in the publication templates and the **layout-group** resource is only responsible for the logical structure of section pages.

Areas can contain **group** elements. A **group** element contains a series of one or more **area**s. These containment rules mean that you can use the **group** and **area** elements to create complex multi-column page structures (although the actual positioning of the columns is carried out by the publication templates).

group elements have a **root** attribute that specifies whether or not they can form the root element of a section page. A section page's layout is determined by assigning one of these "root" **group**s to it.

Namespace URI

The namespace URI of the **layout-group** schema is **http://xmlns.escenic.com/2008/layout-group**.

Root Element

The root of a **layout-group** file must be a **groups** element.

4.1 allow-content-types

Defines the content types that are allowed in this element's owning **area**. Any teaser added to this **area** must belong to a content item of one of these types. The allowed content types are defined by a series of **ref-content-type** and/or **ref-content-type-group** elements.

To allow lists of contents to be desked onto an **area** along with other restricted content types; you should explicitly add **com.escenic.list** as an **allowed content type**

Syntax

```
<allow-content-types>
  (<ref-content-type/>|<ref-content-type-group/>)+
</allow-content-types>
```

4.2 area

Defines an area. An area is a series of **ref-group** elements in any order.

Syntax

```
<area
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT*
    <ct:options>...</ct:options>*
    <ref-group/>*
    <allow-content-types>...</allow-content-types>?
</area>
```

Examples

- This example shows an **area** element. Note the use of the **ct:options** element to associate alternative CSS settings with the area. The options are displayed in Content Studio, enabling editorial staff to select different area layouts.

```
<area name="header">
  <ui:label>Header</ui:label>
  <ui:description>Content added here will appear on top of page</ui:description>
  <ct:options>
    <ct:field type="enumeration" name="border">
      <ui:label>Style</ui:label>
      <ui:description>Changes the style of the header</ui:description>
      <ct:enumeration value="border: 1px solid black;">
        <ui:label>Border</ui:label>
      </ct:enumeration>
      <ct:enumeration value="border: 5px solid black;">
        <ui:label>Fat Border</ui:label>
      </ct:enumeration>
      <ct:enumeration value="background: #F55;">
        <ui:label>Red Background</ui:label>
      </ct:enumeration>
    </ct:field>
  </ct:options>
</area>
```

- This example shows an **area** containing two groups.

```
<area name="center">
  <ui:label>Center Column</ui:label>
  <ui:description>Content placed here will appear in the Center column</
ui:description>
  <ref-group name="two-col"/>
  <ref-group name="three-col"/>
</area>
```

Attributes

name="*NCName*"
The name of the **area** element.

4.3 content-type-group

Defines a content-type group. A **content-type-group** is a series of one or more **ref-content-type** elements.

Syntax

```
<content-type-group
    name="NCName"
  >
  <ref-content-type/>+
</content-type-group>
```

Attributes

`name="`*`NCName`*`"`
The name of the **`content-type-group`** element.

4.4 group

Defines a section page group. A **`group`** is a series of one or more **`area`**s.

Syntax

```
<group
    name="NCName"
    root="(true|false)"?
  >
  ANY-FOREIGN-ELEMENT*
    <ct:options>...</ct:options>?
    <area>...</area>+
</group>
```

Examples

- This example shows a **`group`** that defines a simple two-column layout.

```
<group name="two-col">
  <area name="left"/>
  <area name="right"/>
</group>
```

- This example shows a root **`group`** (a **`group`** that is used to define an entire page). Note the use of the **`ct:options`** element to associate alternative CSS settings with the group. The options are displayed in Content Studio, enabling editorial staff to select different page layouts.

```
<group name="news" root="true">
  <ui:label>News</ui:label>
  <ct:options>
    <ct:field name="news-background-option" type="enumeration">
      <ui:label>Group background</ui:label>
      <ct:enumeration value="white">
        <ui:label>White</ui:label>
      </ct:enumeration>
      <ct:enumeration value="#CCCCCC">
        <ui:label>mourn</ui:label>
      </ct:enumeration>
      <ct:enumeration value="pink">
        <ui:label>Pink</ui:label>
      </ct:enumeration>
    </ct:field>
  </ct:options>
```

```
  <area name="header">
    <ui:label>Header</ui:label>
    <ui:description>Content added here will appear on top of page</ui:description>
    <ct:options>
      <ct:field type="enumeration" name="border">
        <ui:label>Style</ui:label>
        <ui:description>Changes the style of the header</ui:description>
        <ct:enumeration value="border: 1px solid black;">
          <ui:label>Border</ui:label>
        </ct:enumeration>
        <ct:enumeration value="border: 5px solid black;">
          <ui:label>Fat Border</ui:label>
        </ct:enumeration>
        <ct:enumeration value="background: #F55;">
          <ui:label>Red Background</ui:label>
        </ct:enumeration>
      </ct:field>
    </ct:options>
  </area>
  <area name="rightcolumn">
    <ui:label>Right Column</ui:label>
    <ui:description>Content placed here will appear in the right column</
ui:description>
  </area>
  <area name="center">
    <ui:label>Center Column</ui:label>
    <ui:description>Content placed here will appear in the Center column</
ui:description>
    <ref-group name="two-col"/>
    <ref-group name="three-col"/>
  </area>
</group>
```

Attributes

`name="`*NCName*`"`
The name of the **`group`** element.

`root="(true|false)" (optional)`
If set to **`true`**, then this **`group`** can be used as the root **`group`** of a section page.

4.5 groups

The root element of a **`layout-group`** publication resource. It contains a sequence of **`group`** elements defining all the groups that are to be available for a publication's section pages. It may also contain a sequence of **`content-type-group`** elements referring to a list of existing content-type names, which can be used by the **`allow-content-types`** element of an **`area`** of a group.

Syntax

```
<groups>
  <group>...</group>+
    <content-type-group>...</content-type-group>*
</groups>
```

4.6 ref-content-type

A reference to one of the **content-type**s defined in the **content-type** publication resource.

Syntax

```
<ref-content-type
    name="text"
  />
```

Attributes

name="*text*"
The name of the **content-type** referenced by this **ref-content-type** element. The name you enter must exactly match the name of a **content-type** defined in the **content-type** publication resource.

4.7 ref-content-type-group

A reference to one of the **content-type-group**s defined elsewhere in the **layout-group** resource file.

Syntax

```
<ref-content-type-group
    name="text"
  />
```

Attributes

name="*text*"
The name of the **content-type-group** referenced by this **ref-content-type-group** element. The name you enter must exactly match the name of a **content-type-group** defined elsewhere in the **layout-group** resource file.

4.8 ref-group

A reference to a **group**.

Syntax

```
<ref-group
    name="text"
  />
```

Attributes

name="*text*"
The name of the **group** referenced by this **ref-group** element. The name you enter must exactly match the name of a **group** defined elsewhere in the resource file. If this is not the case

then an error will be reported when you upload the **`layout-group`** resource to the Content Engine.

5 interface-hints

The **interface-hints** schema defines additional elements used by Escenic components such as Content Studio and the Content Engine. They can be included at various points in Escenic publication resources such as the **content-type**, **teaser-type** and **layout-group** files.

The purpose of the **interface-hints** elements is to define labels, descriptions, icons and other user-interface related items that can be used in application user interfaces and the Escenic presentation layer.

These elements may be inserted in the publication resource files at any location where the *ANY-FOREIGN-ELEMENT* placeholder indicates that foreign elements are allowed. However, not all elements are meaningful in all locations. The descriptions in this chapter indicate the locations in which each element is intended to be used.

Namespace URI

The namespace URI of the **interface-hints** schema is **http://xmlns.escenic.com/2008/interface-hints**.

5.1 blacklisted-elements

Contains a list of XHTML elements that are to be disallowed in the rich text field defined by this element's parent **field** element. This element only has any effect if specified as the child of a rich text field (a **basic** field where **mime-type** is set to **application/xhtml+xml**).

Specifying XMHTL element names here causes the corresponding user interface buttons to be hidden when the owning field is displayed in a Content Studio content editor. Note, however, that this does not really prevent the blacklisted element from being used: it can still be entered by hand in the field using the **Edit source** option.

The blacklist may contain one or more of the following elements, separated by spaces:

h1 h2 h3 h4 h5 h6 b i u s p[align] p[align=right] sub sup table ul ol a

Syntax

```
<blacklisted-elements>
  text
</blacklisted-elements>
```

5.2 boolean-label

Defines a label that can be used instead of the value 'true' or 'false' for this element's parent boolean **field**. This element should normally appear in pairs, one for each boolean value. The parent element may contain several such pairs in different languages. In this case, the **xml:lang** attribute must be used to specify the language of each **boolean-label**.

Syntax

```
<boolean-label
    xml:lang="text"?
    value="(true|false)"
  >
  text
</boolean-label>
```

Examples

- `<ui:boolean-label value="true">On</ui:boolean-label>`
- `<ui:boolean-label lang="no" value="false">Av</ui:boolean-label>`

Attributes

`xml:lang="`*`text`*`"` (optional)
The language of the **`boolean-label`**. You should use ISO-639 format language IDs. For a complete list of these IDs see http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt. Applications are responsible for using this attribute to select the most appropriate language.

`value="(true|false)"`
Indicates which of the boolean values this **`boolean-label`** is to represent.

Allowed values are:

`true`
This **`boolean-label`** represents **`true`**.

`false`
This **`boolean-label`** represents **`false`**.

5.3 decorator

Defines a **decorator** for this element's parent **`content-type`** or **`group`** element. Decorators are Java programming constructs that can be used by Escenic plugins and third party code to simplify the development of complex templates. This is done by allowing complex logic to be implemented in Java code in a decorator, instead of in the templates. See the JavaDoc for **`neo.xredsys.presentation`** for more information on decorators.

Syntax

```
<decorator
    class="text" name="text"?
  >
  <parameter/>?
</decorator>
```

Attributes

`class="`*`text`*`"` (optional)
The name of the Java class that implements the decorator. You are recommended to omit this attribute and specify the instance name of the decorator in the **`name`** attribute instead.

`name="`*`text`*`"` (optional)
If the **`class`** attribute is omitted (recommended), then this attribute must hold the instance name of the decorator you want to use. If **`class`** **is** specified, then this attribute can hold any name you choose (for documentation purposes).

5.4 default-content-type

If present, this element indicates that its parent **`content-type`** element represents the default content type of this publication and may be treated accordingly in client applications. This element only has meaning as the child of a **`content-type`** element, and it should not appear more than once in a **`content-type`** resource. The result of specifying the element more than once is undefined.

Syntax

```
<default-content-type/>
```

5.5 default-duplicate-content-type

If present, this element indicates that its parent **`content-type`** element represents the default duplicate content type of this publication and may be treated accordingly in client applications. This element only has meaning as the child of a **`content-type`** element, and it should not appear more than once in a **`content-type`** resource. The result of specifying the element more than once is undefined.

A non-binary content item cannot be "duplicated" into a binary content item.

Syntax

```
<default-duplicate-content-type/>
```

5.6 description

Defines text that can be used to describe this element's parent element in applications. Text entered here may, for example, be displayed as tool-tips where appropriate. The parent element may contain several **`description`**s in different languages. In this case, the **`xml:lang`** attribute must be used to specify the language of each label.

Syntax

```
<description
    xml:lang="text"?
  >
  text
</description>
```

Attributes

`xml:lang="`*text*`"` (optional)
The language of the **`description`**. You should use ISO-639 format language IDs. For a complete list of these IDs see http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt. Applications are responsible for using this attribute to select the most appropriate language.

5.7 editor

Specifies a field editor where user will be able to use HTML,JS,CSS to create a web field editor UI which will be placed in CS content editor.

Syntax

```
<editor
    type="html"
    name="text"
  />
```

Attributes

`type="html"`
Represents html view in user interface

`name="`*text*`"`
A token which is mapped with a uri in **`com/escenic/resolver/NamedServiceResolver`** component

5.8 editor-style

Determines how the item represented by the parent element will be displayed in editing applications. Currently this element may only appear as the child of a **`relation-type`** element and may only have the value **`gallery`**: it is ignored in any other location or if it has any other value. When used as described above, it causes relations of the type defined by the parent element to be treated like ordinary fields in Content Studio. "Gallery" relations are:

- Displayed in the main area of the Content Studio content editor together with fields, rather than on the right hand side where relations are normally displayed.
- Not affected by Content Studio's **View** > **Show relations** option.

Syntax

```
<editor-style>
  text
</editor-style>
```

5.9 expert

If present, this element indicates that its parent is not typically used on a day-to-day basis, and that it should be hidden in application user interfaces, except when the field contains any data. The user should be provided with an option to show these expert items.

Syntax

```
<expert/>
```

5.10 field-set

Groups a set of fields together for display purposes in Content Studio. The **`style`** attribute specifies the method to be used to group the fields. The **`field-set`** may optionally have a child **`label`** element. If present it is displayed as a title for the group.

Syntax

```
<field-set
    style="(box)"
  >
  <label>...</label>?
    (<ctct:field/>|<ctct:field-group/>)+
</field-set>
```

Attributes

`style="(box)"`
The graphical method to be used to group the fields in the **`field-set`**.

Allowed values are:

`box`
The fields in the **`field-set`** are grouped by enclosing them in a box, making them look similar to a complex field.

5.11 group

Defines a content type group. Content type groups are used by Content Studio for two purposes:

- To group the content types displayed in the **File** > **New** menu, making the menu easier to navigate. (Note, however, that this is only done for groups that contain more than one content type.)
- To display buttons in the **Search** panel that can be used to filter search results. When one of the filter group buttons is selected in Content Studio, search results are filtered to show only content items of the types in the group.

The **`group`**'s child **`ref-content-type`** elements reference the content types that belong to the group.

Syntax

```
<group>
  <label>...</label>?
    <ref-content-type/>+
</group>
```

Examples

- ```
  <ui:group name="articles">
  ```
  ```

```
    <ui:label>Stories</ui:label>
    <ui:ref-content-type name="news"/>
  </ui:group>
```

5.12 hidden

If present, this element indicates that its parent should be hidden in application user interfaces.

Syntax

```
<hidden/>
```

5.13 icon

Contains the name of an icon that can be used by application user interfaces when displaying the object represented by this element's parent element. Currently, the icon is only used in Content Studio.

The value may either be the name of one of the following predefined icons:

Name	Icon
article	
attachment	
audio	
generic	
graphic	
image	
image-series	
inbox	
link	
list	
map	
media	
news	
page	

Name	Icon
person	
poll	
publication	
section	
video	
widget	

or the absolute URI of an image you want to use as an icon. The referenced image must be accessible from all the machines on which Content Studio is used. It can be in any image format supported by Java (but PNG is recommended). For best results you should use a small (32*32), square image.

> When an **icon** element appears as the child of a **style-class** element you can also specify a single unicode character as element content. An icon will then be generated from the specified character. You can specify the character either literally or as a numeric entity (**↑** for example).

Syntax

```
<icon>
  text
</icon>
```

Examples

- This example selects one of Content Studio's built-in icons.

  ```
  <ui:icon>audio</ui:icon>
  ```

- This example selects a custom icon stored on a server somewhere in the network.

  ```
  <ui:icon>http://my-company-server/icons/custom-audio.png</ui:icon>
  ```

- This example creates an icon from the ↑ character. You can also specify characters as entities (**↑** in this case). Note that this form of the **icon** element only works when the element is the child of a **style-class** element.

  ```
  <ui:icon>↑</ui:icon>
  ```

5.14 keystroke

Contains the definition of a key combination that can be used by application user interfacesto insert/invoke the content/functionality represented by this element's parent element. Currently, the icon is only used in Content Studio.

The syntax for specifying key combinations is:

```
modifier* key
```

where:

modifier
is one of **shift**, **control**, **ctrl**, **meta**, **alt** or **altGraph** and indicates one of the keyboard modifier key (note that **ctrl** is the **Cmd** key on a Mac).

key
Is a key identifier. These are always upper case. For the standard alphanumeric keys, the character on the key is used. For a complete list of all key identifiers, see http://download.oracle.com/javase/6/docs/api/java/awt/event/KeyEvent.html. Use the key names listed here, without the "**VK_**" prefix.

Here are some example keystroke definitions:

```
alt 8
shift alt PAGE_DOWN
```

Make sure you avoid keystroke combinations that are already in use in Content Studio.

Syntax

```
<keystroke>
  text
</keystroke>
```

5.15 label

Defines a label that can be used to identify this element's parent element in applications. The parent element may contain several **label**s in different languages. In this case, the **xml:lang** attribute must be used to specify the language of each label.

Syntax

```
<label
    xml:lang="text"?
  >
  text
</label>
```

Attributes

xml:lang="*text*" (optional)
The language of the **label**. You should use ISO-639 format language IDs. For a complete list of these IDs see http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt. Applications are responsible for using this attribute to select the most appropriate language.

5.16 macro

Defines a macro to be added to the Content Studio style bar of the rich text field defined by this element's parent **field** element. This element only has any effect if specified as the child of a rich text field (a **basic** field where **mime-type** is set to **application/xhtml+xml**).

Inserting this element as the child of a rich text **field** element causes an additional button to be displayed in the field's Content Studio style bar. The **icon** sub-element defines the appearance of the button, the **step** sub-element defines the action to be performed by the macro and the **description** sub-element determines the button's tool-tip text. An optional **keystroke** sub-element allows a keyboard shortcut to be associated with the macro.

> **macro** buttons and **style-class** buttons appear in the style bar in the same order as they appear in the **interface-hints** resource.

Syntax

```
<macro
    name="..."
  >
  <icon>...</icon>

    <step/>

    <keystroke>...</keystroke>?
    <description>...</description>?
</macro>
```

Attributes

name="..."
The name of the macro. The name must be unique among all **macro** and **style-class** names in the file. The name may only contain English alphanumeric characters: no spaces, punctuation marks or special characters of any kind are allowed.

5.17 parameter

Defines a parameter (name/value pair) to be used by the decorator defined in this element's parent **decorator** element.

Syntax

```
<parameter
    name="text"
    value="text"
  />
```

Attributes

name="*text*"
The name of the **parameter**.

value="*text*"
The value of the **parameter**.

5.18 read-only

If present, this element indicates that its parent should be considered read-only in application user interfaces.

Syntax

```
<read-only/>
```

5.19 ref-content-type

Specifies the name of a **content-type** that is to belong to this element's parent **group**.

Syntax

```
<ref-content-type/>
```

5.20 ref-relation-type

Specifies a relation type to be associated with this element's parent **panel** element. Relations of this type will then be displayed on the corresponding panel by Content Studio. The **ref-relation-type** element must be a child of a **panel** element in a **content-type** resource file.

If a **relation-type** is referred to from two different panels in the same **content-type**, the **relation-type** will be displayed on both panels.

All relation types not referred to from a **ref-relation-type** will be added to the **content-type**'s first panel.

Syntax

```
<ref-relation-type
    ref="text"
  />
```

Attributes

ref="*text*"
The name of the **relation-type** to display on this panel. The **relation-type** must be referred to from the **panel**'s parent **content-type**.

5.21 step

Contains the definition of one step in the macro represented by this element's parent element. Currently, only one such step is allowed, and it always defines a sequence of characters to be inserted. It may also optionally define an in-line XHTML element in which the inserted characters are to be wrapped, and a **class** attribute value to be assigned to the wrapping element.

Syntax

```
<step
    action="insert"
    text="text"
    wrap-element="NCName"
    class="NCName"?
  />
```

Attributes

`action="insert"`
The action to be performed in this macro step. Currently, the only value allowed is **`insert`**.

`text="`*`text`*`"`
The text to be inserted by this step. Note that:

- The text may **not** include any XML or HTML markup: if you attempt to do this, the markup will be escaped.
- You cannot use standard HTML entities such as **`&mdash;`**. If you need to enter special characters you can either enter them directly as UTF-8 characters, or use numeric entities (for example, **`&#x2014;`** instead of **`&mdash;`**).

`wrap-element="`*`NCName`*`"` (optional)
The name of an XHTML in-line element in which the inserted text is to be wrapped.

`class="`*`NCName`*`"` (optional)
A **`class`** attribute value to be added to the XHTML element specified with the **`wrap-element`** attribute: a space-separated list of CSS class names.

5.22 style

Contains a CSS style definition that can be applied to this element's parent **`field`** element by application user interfaces. Currently, it is used to style the content of rich text fields (**`basic`** fields where **`mime-type`** is set to **`application/xhtml+xml`**) in Content Studio. It is not used by any other application, and has no effect if specified as the child of any other kind of element.

Syntax

```
<style>
  text
</style>
```

Examples

- This example sets different colors for the HTML headings h1 and h2.

  ```
  <ui:style>
      h1 {color:red;}
      h2 {color:green;}
      </ui:style>
  ```

- This example sets default fonts and font size for the the whole field, and adds overrides for the HTML headings h1 to h4. Note that **`body`** here refers to the HTML **`body`** element, not the name of the content item field being styled - it will work on any rich text field.

  Note that Content Studio will only actually use fonts that are either available in the operating system or supplied by Java.

```
<ui:style>
    body {
      font-family: "Monaco";
      font-size: 16px;
    }
    h1, h2, h3, h4 {
      font-family: "Georgia", sans-serif;
    }
    </ui:style>
```

- This example adds paragraph numbering to **p** elements.

```
<ui:style>
    p {
        margin: 0; padding: 0;
        line-height: 1.6;
        max-width: 20cm;
        counter-increment: paragraph 1;
      }
    p+p {
        text-indent: 30px
      }
    p:before{
        float: right;
        margin-right: -2.5em;
        content: "#" counter(paragraph);
        color: #444;
        font-size: 0.8em;
      }
    p+p:before{
        margin-right: 0;
      }
    </ui:style>
```

- This example prepends the text "VIDEO:" before video links and wraps the link in [] parentheses. Related elements are marked up in Content Studio as either **img** or **a** elements, so in order to make them easy to target with CSS, they are also assigned **class** attributes. These **class** attributes are formed by appending the content type name with **escenic-** So this example will style related items with the content-type **video**.

```
<ui:style>
    a.escenic-video:before {
      content: "VIDEO: [";
    }
    a.escenic-video:after {
      content: "]";
    }
    </ui:style>
```

5.23 style-class

Defines a style button to be added to the Content Studio style bar of the rich text field defined by this element's parent **field** element. This element only has any effect if specified as the child of a rich text field (a **basic** field where **mime-type** is set to **application/xhtml+xml**).

Inserting this element as the child of a rich text **field** element causes an additional button to be displayed in the field's Content Studio style bar. The **icon** sub-element defines the appearance of the button, and the **description** sub-element determines the button's tool-tip text.

Selecting text and clicking on the button displayed in Content Studio wraps the selected text in an XHTML **span** element with a **class** attribute set to *name*.

You can use one of the **group** or **drop-down** attributes to group **style-class** elements into mutually exclusive sets: this makes it impossible to apply more than one of the grouped styles to the same text selection. If you use the **group** attribute, then the styles are displayed as a set of mutually exclusive buttons. If you use the **drop-down** attribute, then they are presented in a drop-down menu.

> **macro** buttons and **style-class** buttons appear in the style bar in the same order as they appear in the **interface-hints** resource.

Syntax

```
<style-class
    name="text"
    ( group="text" | drop-down="text" )?
  >
  <icon>...</icon>

    <description>...</description>
</style-class>
```

Examples

- This example creates a style button for marking text green, using an icon stored on the server.

```
<field mime-type="application/xhtml+xml" type="basic" name="body">
  <ui:style>
      span.green {
      color: green;
      }
      </ui:style>
  <ui:style-class name="green">
    <ui:icon>http://my.server/myicon.png</ui:icon>
    <ui:description>Mark text green</ui:description>
  </ui:style-class>
</field>
```

- This example creates a set of three mutually exclusive style buttons. Button icons will be generated from the specified arrow characters.

```
<field mime-type="application/xhtml+xml" type="basic" name="body">
  <ui:style>
      h2 { font-weight: normal; text-decoration: underline; }
      span.important { color: red; }
      span.semi-important { color: #ffcc00; }
      span.not-so-important { color: green; }
      </ui:style>
  <ui:style-class name="important" group="importance">
    <ui:icon>↑</ui:icon>
  </ui:style-class>
  <ui:style-class name="semi-important" group="importance">
    <ui:icon>~</ui:icon>
  </ui:style-class>
  <ui:style-class name="not-so-important" group="importance">
```

```
      <ui:icon>↓</ui:icon>
    </ui:style-class>
  </field>
```

- This example creates a drop-down menu containing three options. Icons will be generated from the specified arrow characters and displayed alongside the option labels in the drop-down menu.

```
<field mime-type="application/xhtml+xml" type="basic" name="body">
  <ui:style>
      h2 { font-weight: normal; text-decoration: underline; }
      span.important { color: red; }
      span.semi-important { color: #ffcc00; }
      span.not-so-important { color: green; }
      </ui:style>
  <ui:style-class name="important" drop-down="importance">
    <ui:icon>↑</ui:icon>
  </ui:style-class>
  <ui:style-class name="semi-important" drop-down="importance">
    <ui:icon>~</ui:icon>
  </ui:style-class>
  <ui:style-class name="not-so-important" drop-down="importance">
    <ui:icon>↓</ui:icon>
  </ui:style-class>
</field>
```

Attributes

`name="`*text*`"`
The name of the CSS class to be associated with the style button. For the button to have any visible effect in Content Studio, the parent **`field`** element must also have a **`style`** child element where this CSS class is defined.

`group="`*text*`"` (optional)
The name of a group of mutually-exclusive **`style-class`** elements to which this element belongs. All style buttons belonging to the same group will behave so that only one of the grouped styles can be applied to the same text selection.

`drop-down="`*text*`"` (optional)
The name of a group of mutually-exclusive **`style-class`** elements to which this element belongs. The group is displayed in the Content Studio style bar as a drop-down menu: selecting an entry from the menu applies the corresponding style to the currently selected text.

5.24 tag-scheme

Contains the **scheme** (that is URI) of a tag structure that may be used to tag content items of the type represented by this element's parent **`content-type`** element. client applications such as Content Studio can use the presence of **`tag-scheme`** elements to determine which tags they make available to users editing content items. Content Studio does not display a **Tags** tab for content types that have no **`tag-scheme`** elements.

Syntax

```
<tag-scheme>
  text
</tag-scheme>
```

5.25 title-field

Nominates one of the fields in a **`content-type`** as its **title field**. **`title-field`** must be the child of a **`content-type`** element, and the value it contains must be the name of one of the **`field`**s in that **`content-type`**.

Title field is a legacy concept required for backwards compatibility with earlier versions of the Content Engine. If a title field is not defined in this way then the **`PresentationArticle`** bean's (deprecated) **`title`** property will not return a value.

Please note that the field that is to be used as **`title-field`** needs to be of **`mime-type`**: **`text/plain`**.

Syntax

```
<title-field>
  text
</title-field>
```

5.26 unit

Defines a secondary label that can be used to identify the unit of the values stored in this element's parent **`field`** element. In Content Studio, the contents of this element are displayed **after** the **`field`** it describes. It is typically used to hold a unit name such as "centimetres", "cm." or "seconds".

Syntax

```
<unit>
  text
</unit>
```

5.27 value-if-unset

Specifies a default value to be associated with this element's parent **`field`** element. **`value-if-unset`** must be the child of a **`field`** element in a **`content-type`** resource file. The contents of the element must be a valid value for the parent **`field`** (that is, it must be of the correct type, and must fall within any constraints specified for the **`field`**).

The following field types can have have this value set:

- number
- boolean
- enumeration
- uri
- basic

The different formats will have the same **Java type** as the stored value.

Syntax

```
<value-if-unset>
  text
</value-if-unset>
```

6 feature

The **feature** publication resource, unlike the other publication resources, is a plain text file containing simple property settings in the form:

```
name=value
```

The properties in the file define miscellaneous aspects of the Content Engine's behavior for a particular publication.

The **feature** resource must be uploaded to the Content Engine in the same way as the other XML publication resources.

The following sections describe each of the properties that can be included in the **feature** resource.

6.1 allowFrontPageAsHomeSection

If **allowFrontPageAsHomeSection** is set to **true**, then the publication's front page (that is, its root section page) may be used a home section for content items. If **allowFrontPageAsHomeSection** is not set or is set to any other value, then this is not allowed.

```
allowFrontPageAsHomeSection=true
```

6.2 article.list.age.default

Sets the default value for the **from** attribute of the **article:list** JSP tag. **article:list** retrieves the most recent content items that meet specified criteria. The **from** attribute specifies the maximum age (in hours) of the content items to be retrieved. If **from** is not specified, then the default value specified here is used.

The default value of this property is 720 hours (=30 days). You can set it to "no limit" by specifying a value of -1 This is not recommended for production sites.

For a description of the **article:list** tag, see **Escenic Tag Library Reference**, **section 2.4** .

6.3 article.list.age.max

Sets the maximum allowed value for the **from** attribute of the **article:list** JSP tag. **article:list** retrieves the most recent content items that meet specified criteria. The **from** attribute specifies the maximum age (in hours) of the content items to be retrieved. If the specified **from** value is greater than **article.list.age.max** then it is ignored and **article.list.age.max** is used instead.

The default value of this property is 720 hours (=30 days). You can set it to "no limit" by specifying a value of -1 This is not recommended for production sites, since unlimited queries can cause serious performance problems.

For a description of the **`article:list`** tag, see **Escenic Tag Library Reference**, **section 2.4** .

6.4 article.presentation.gzip

If **`article.presentation.gzip`** is set to **`true`**, then basic (i.e string) field values longer than a certain length are stored in compressed form by the Content Engine. The limit above which fields will be compressed is defined by the **`article.presentation.gzip.threshold property`** (see section 6.5).

If **`article.presentation.gzip`** is not set or set to any other value, then compression is not carried out.

Example:

```
article.presentation.gzip=true
```

6.5 article.presentation.gzip.threshold

This property specifies the maximum number of characters that can be stored as uncompressed string if **`article.presentation.gzip`** (see section 6.4) is set to **`true`**. It is ignored if **`article.presentation.gzip`** is set to **`false`**.

If **`article.presentation.gzip`** is set to **`true`** and this property is not set, then a default value of 100 is used.

Example:

```
article.presentation.gzip.threshold=200
```

6.6 bootstrapOnStartup

The Content Engine incorporates a publication bootstrapper called **`InitialBootstrapper`** that automatically accesses the sections in a publication immediately after server startup. First-time access of a section takes a long time; subsequent accesses are much faster because various components have been compiled and cached for re-use. While this bootstrap process is running, any user accesses to the sections being bootstrapped are refused: the server returns a HTTP 503 response (Service Unavailable).

This bootstrap process therefore gives a better user experience during server startup: the user gets an immediate response (even if it is negative) rather than a "hanging" browser window. It also ensures that the Content Engine does actually start up, instead of being crippled by a flood of time-consuming requests that it cannot respond to.

`bootstrapOnStartup` lets you specify which sections of your publication are to be bootstrapped in this way. If you do not specify **`bootstrapOnStartup`**, then no sections of this publication are bootstrapped. You can specify the sections to be bootstrapped in the following ways:

- Enter a comma-separated list of section unique names. The specified sections will be bootstrapped, in the order specified. For example:

  ```
  bootstrapOnStartup=ece_frontpage,main,news,sports,football
  ```

- Enter a comma-separated list of section unique names. The specified sections will be bootstrapped, in the order specified. For example:

  ```
  bootstrapOnStartup=1,47,30
  ```

- Specify a mixture of section IDs and unique names, separated by commas. Exactly those sections listed will be bootstrapped, in the order specified. It is legal to mix section unique names and section Ids.
- Enter the keyword **`true`**. The sections are bootstrapped in their natural order, starting from the root section. By default only the first two levels of the section hierarchy will be bootstrapped (that is, the root section plus the root section's immediate children). The number of levels bootstrapped can be modified (for all publications) by setting the **`InitialBootstrapper`** component's **`depth`** property. For more information about this, see **Escenic Content Engine Server Administration Guide**, **chapter 7** .
- Enter a single number representing a level in the section hierarchy. The sections are bootstrapped in their natural order, starting from the root section, down to the level specified. So you specify **`3`** then the root section, its children and grandchildren are bootstrapped.
- Enter the keyword **`ece_all`**. All sections in the document are bootstrapped in their natural order, starting from the root section. The **`InitialBootstrapper`** component's **`depth`** and **`timeout`** properties are ignored. This setting is **not** recommended in a production environment, unless it is critical that **all** sections are primed.

6.7 catalog.orderBy

If **`catalog.orderBy`** is set to **`date`**, then objects in Escenic catalogs are sorted by date. Specifically, they are sorted by their "last modified" date in descending order - the most recently updated first.

If **`catalog.orderBy`** is set not set or is set to any other value, then objects in Escenic catalogs are sorted by name in ascending alphabetic order.

Example:

```
catalog.orderBy=date
```

6.8 com.escenic.article.staging

Switches content item staging on or off for this publication. Content item staging is enabled by default, and you can disable it for this publication by setting **`com.escenic.article.staging`** to **`false`**:

```
com.escenic.article.staging=false
```

If content item staging has been globally disabled, then you can enable it for this for this publication by setting **`com.escenic.article.staging`** to **`true`**:

```
com.escenic.article.staging=true
```

6.9 default.crop

If **`default.crop`** is set to **`rule-of-thirds`**, then the default vertical positioning of crops is based on the "rule-of-thirds". This means that instead of being centered on the true vertical center of the original image, they are centered on a line drawn one-third of the distance from the top of the image. This is a better choice for many kinds of image. It is more likely, for example, to include people's faces.

For example:

```
default.crop=rule-of-thirds
```

6.10 htaccess.user

Sets the HTTP username that any outgoing HTTP requests from the server must use when accessing the local publication. Must be used in combination with **`htaccess.password`**.

For example:

```
htaccess.user=test003
htaccess.password=secret003
```

6.11 htaccess.password

Sets the HTTP password that any outgoing HTTP requests from the server must use when accessing the local publication. Must be used in combination with **`htaccess.user`**.

For example:

```
htaccess.user=test003
htaccess.password=secret003
```

6.12 initialInlineImageVersion

Specifies image version that is to be used as default when inserting inline images. The following example specifies that an image version called **`inline`** is to be used as default:

```
initialInlineImageVersion=inline
```

The name you specify must exactly match the **`id`** attribute of a **`version`** element in the **`image-versions`** publication resource.

6.13 com.escenic.image.quality

Specifies the quality setting to be used when converting images to PNG (the format used internally by the Content Engine). You may specify a value between 0 (maximum compression, minimum quality) and 100 (no compression, maximum quality). If this property is not set, then a default value of 70 is used.

Example:

```
com.escenic.image.quality=80
```

6.14 local.url

Specifies the URL that should be used instead of the publication's URL when this server attempts to access itself via HTTP. If the publication URL is set to a load-balancing front end, then attempts to bootstrap the local publication will fail, as the requests may be serviced by other servers than the one being primed.

Example:

```
local.url=http://localhost:8080/mypublication/
```

6.15 multimedia.archive.orderBy

This property is a deprecated synonym for **`catalog.orderBy`** (see section 6.7). If both **`catalog.orderBy`** and **`multimedia.archive.orderBy`** are specified, then **`catalog.orderBy`** is used.

6.16 multimedia.image.virtualVersions

Can be set (true or false) to control whether or not all versions of an image will be generated on the fly. A true value means that automatic image version generation will be disabled.

Note: This property is most likely to be invalid for ECE v5.0. Image version generation is moved to the presentation layer now. There is only automatically generated images available now. New configuration options similar to this might be introduced in near future.

Example:

```
multimedia.image.virtualVersions=false
```

6.17 multimedia.media.versiontypes

Multiple versions of multimedia files are not currently supported. The **`feature`** resource **must** however contain the following three property definitions, exactly as specified below:

```
multimedia.media.versiontypes=a
multimedia.media.versiontype.default=a
multimedia.media.versiontype.a=Default Version
```

6.18 plugin.[pluginName].enabled

Specifies whether or not a named plugin is enabled. For example:

```
plugin.word_count_plugin.enabled=false
```

Disables a plugin called **`word_count_plugin`**.

6.19 pool.autoRemoval

Enables the automatic **pool removal service** for this publication. Pool is a synonym for section page The pool removal service tries to keep the pools at a reasonable size (the default being 200). The articles at the bottom of the columns are removed first to minimize the impact on the visible portion of the pool.

Example:

```
pool.autoRemoval=true
```

6.20 pool.limit

Sets the maximum number of articles to leave in the pool for the pool removal service. Default value for this is 200. This variable is only used if the automatic pool removal service is enabled.

Example:

```
pool.limit=500
```

6.21 publication.previewURL

Specifies an alternative publication URL to be used for previews generated by Content Studio. This makes it possible to prevent previews being generated on your production servers, which can be a good idea if you have several layers of caching (for example web caches, or ESI). Setting this property forces all previews to be generated on using the publication URL you specify, which can be on a different server.

The following example forces all previews to be generated on a server called **`escenic-publishing`**:

```
publication.previewURL=http://escenic-publishing:8080/mypublication/
```

6.22 publication.previewUsesSectionUrl

If **`publication.previewUsesSectionUrl`** is set to **`true`**, the Content Engine will honour section URLs when generating previews.

By default, section URLs are ignored when displaying previews, but in some cases it is important that previews use section-specific URLs.

Example:

```
publication.previewUsesSectionURL=true
```

6.23 relation.articles.includeCrossRelatedArticles

If **`relation.articles.includeCrossRelatedArticles`** is set to **`true`**, then lists returned by the **`PresentationArticle`** bean's **`articles`** property will include **foreign** content items (when appropriate). This is the default. If **`relation.articles.includeCrossRelatedArticles`** is not set or is set to any other value than true, then foreign content items are not included in such lists.

A foreign content item in this context is a content item that:

- Has its home section in a different publication.
- Has not been cross-published to the current publication.

If content item A in publication P is related to content item B in a different publication, and content item B has **not** been cross-published to P, then the related content item will by default be excluded from the list. If content item B **has** been explicitly cross-published to publication P, however, then it will be included.

This property allows you to ensure that such foreign relations will always be returned. The ultimate effect of setting this property is to enable links between publications via "related articles" lists.

For a description of **`PresentationArticle.articles`**, see **Escenic Content Engine Bean Reference**, **section 2.2.7** .

6.24 studio.crop

This feature can be used to specify custom crop masks for Content Studio. By default, Content Studio offers a number of standard masks for cropping images. One of these masks, called **Free crop**, can be reshaped, but all the others (**Landscape**, **Portrait** and so on) have fixed aspect ratios. If these fixed crops masks do not meet your needs, then you can define crop masks of your own by specifying **`studio.crop`** features like this:

```
studio.crop.name=width:height
```

where:

- *name* is the name to be displayed in Content Studio.
- *width* is the relative width of the crop mask.
- *height* is the relative width of the crop mask.

For example:

```
studio.crop.wide=5:2
```

> Note that If you specify a custom crop map in this way, then it replaces **all** the default fixed crop masks. So even if you only want to add one custom crop mask to the list, you will probably need to

define several **`studio.crop`** features: one to define your custom crop mask, plus several others to recreate the default crop masks that you use.