CUE Content Store

Resource Reference

7.17.2-3

Table of Contents

1 Introduction

This manual contains reference material describing the CUE **publication resources**. The publication resources are text files containing important system configuration information that determines the structure and other characteristics of a CUE 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-specific resources:

`content-type`
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 file 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 Store's behavior.

Publications also have access to a set of shared publication resources:

story element types
XML files defining the block level elements of which stories are composed – elements such as headline, lead text, paragraph, image, pull quote and so on. Each file contains the definition of one story element type. These resources are created using the same XML elements as the **`content-type`** resource

storyline templates
XML files defining sets of story element types from which stories may be composed. A story is based on a storyline template, which determines what story element types it may contain. These resources are created using elements from the **`storyline-template`** namespace.

publication types
XML files defining different publication types.

container types
XML files defining different container types.

workflow definitions
XML files defining custom workflows.

search filter definitions
XML files defining custom search filter panels.

capability definitions
XML files defining CUE editor capabilities.

content card definitions
XML files defining custom content cards

In order for changes to a publication resource to take effect, the resource file must be uploaded to the Content Store using the web administration interface. For a description of how to do this, see Creating/Updating Publication Resources.

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 CUE.
- 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 8). 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.

1.2 XInclude Support

XInclude is a W3C standard for including XML snippets in XML files. You can use it to reduce duplication and improve the maintainability of your publication resources. Any element in a resource file that has an **`id`** attribute can be re-used other places in the same resource file by referencing it with an XInclude **`include`** element. The following field element, for example:

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

has the **`id titleFields`**, and can therefore be re-used in other places by simply entering:

```
<xi:include xpointer="titleFields"/>
```

In order to use XInclude in a resource file, you must declare its namespace. To declare the namespace in the root element of a content-type resource, for example, you would add the following (highlighted):

```
<content-types
  xmlns="http://xmlns.escenic.com/2008/content-type"
  xmlns:ui="http://xmlns.escenic.com/2008/interface-hints"
  xmlns:xi="http://www.w3.org/2001/XInclude"
  version="4">
```

Note the following restrictions:

- Although XInclude supports inclusion between files, this is not supported in CUE resource files. You can only include elements from the same file.
- Only the following element types have id attributes in CUE resource files:

  `field`
  `relation-type`

  These are therefore the only element types that can be re-used using XInclude.

2 content-type

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

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

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 one of the following elements:

- **content-types**
- **story-element-type**
- **storyline-template**

2.1 allow-content-types

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

Syntax

```
<allow-content-types>
  <ref-content-type/>+
</allow-content-types>
```

2.2 allow-story-element-type

Defines that the annotation may have one child story element.

Syntax

```
<allow-story-element-type>
  <ref-story-element-type/>+
  ANY-FOREIGN-ELEMENT*
</allow-story-element-type>
```

Child Elements

ref-story-element-type: section 2.31.

Only one form of the **`ref-story-element-type`** element may be used: **Standard ref-story-element-type** (section 2.31.2).

2.3 allow-story-element-types

> This element is deprecated. Its functionality is provided by children of the **`constraints`** element (itself a child of the **`elements`** element).

Defines that the story element type may have one or more child story elements.

Syntax

```
<allow-story-element-types
    max="integer"?
    min="integer"?
  >
  <ref-story-element-type/>+
  ANY-FOREIGN-ELEMENT*
</allow-story-element-types>
```

Child Elements

`ref-story-element-type`: section 2.31.

Only one form of the **`ref-story-element-type`** element may be used: **Standard ref-story-element-type** (section 2.31.2).

Attributes

`max="`*`integer`*`"` (optional)
The maximum number of allowed child elements.

`min="`*`integer`*`"` (optional)
The minimum number of allowed child elements.

2.4 allow-storyline-template

Defines the storyline template that are allowed in this element's owning **`content-type`**. Any storyline added to a content item of this **`content-type`** must use this storyline template. The allowed storyline template are defined by **`ref-storyline-template`**.

Syntax

```
<allow-storyline-template>
  <ref-storyline-template/>
</allow-storyline-template>
```

2.5 allow-storyline-templates

Defines the storyline template that is allowed in this element's owning **content-type**. Any storyline added to a content item of this **content-type** must use this storyline template. The allowed storyline template is defined by **ref-storyline-template**.

The usage of this element is deprecated. Use **allow-storyline-template** instead.

Syntax

```
<allow-storyline-templates>
  <ref-storyline-template/>
</allow-storyline-templates>
```

2.6 allowed-story-elements

A list of element types that may be inserted into a storyline or complex story element.

Syntax

```
<allowed-story-elements>
  <ref-story-element-type/>+
  ANY-FOREIGN-ELEMENT*
</allowed-story-elements>
```

Child Elements

ref-story-element-type: section 2.31.

Only one form of the **ref-story-element-type** element may be used: **Standard ref-story-element-type** (section 2.31.2).

2.7 annotation

Defines an annotation for the field. This element is currently only used in fields defined within a story element. story elements.

Syntax

```
<annotation
    name="text"
  >
  <allow-story-element-type>...</allow-story-element-type>*
  ANY-FOREIGN-ELEMENT*
</annotation>
```

Attributes

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

2.8 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 CUE.

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

2.9 base-story-element

The default story element of a storyline or complex story element (typically a plain paragraph element of some kind). This element is entered when the CUE user presses the **Enter** key.

Syntax

```
<base-story-element>
  <ref-story-element-type/>
</base-story-element>
```

Child Elements

ref-story-element-type: section 2.31.

Only one form of the **ref-story-element-type** element may be used: **Standard ref-story-element-type** (section 2.31.2).

2.10 complex

Provides a wrapper for the members of a complex field.

Syntax

```
<complex>
  (<ref-field-group/>|<field>...</field>|ANY-FOREIGN-ELEMENT*)+
</complex>
```

2.11 constraints

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

2.11.1 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>
```

2.11.2 Element constraints

Sets constraints on the elements that can be inserted into a storyline or complex story element.

Syntax

```
<constraints>
  <minimum>...</minimum>?
  <maximum>...</maximum>?
  <required-story-elements>...</required-story-elements>?
  <allowed-story-elements>...</allowed-story-elements>
</constraints>
```

2.11.3 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>*
  <allow-content-types>...</allow-content-types>*
  <allow-storyline-templates>...</allow-storyline-templates>*
  <allow-storyline-template>...</allow-storyline-template>*
</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>
  ```

2.11.4 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>
  ```

2.11.5 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>
```

2.12 content-type

Defines a CUE **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 CUE content item editors.

Syntax

```
<content-type
    name="NCName"
    workflow="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.

`workflow="`*NCName*`"` (optional)
The name of the workflow to use for **`content-type`** element. If no workflow is specified, the default workflow will be used.

2.13 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>*
  <field>...</field>*
  <content-type>...</content-type>+
  ANY-FOREIGN-ELEMENT*
</content-types>
```

Child Elements

`field-group`: section 2.19, **`relation-type-group`**: section 2.35, **`field`**: section 2.17, **`content-type`**: section 2.12.

The following forms of the **`field`** element may be used: **Complex field** (section 2.17.7), **Basic field** (section 2.17.1), **Boolean field** (section 2.17.3), **URI field** (section 2.17.19), **Schedule field** (section 2.17.17), **Number field** (section 2.17.15), **Link field** (section 2.17.13), **Enumeration field** (section 2.17.10), **Date field** (section 2.17.8), **Collection field** (section 2.17.5), **Inherited field** (section 2.17.12).

Attributes

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

2.14 default-story-elements

Elements that CUE is to insert by default when a storyline or complex story element is created. The specified elements are inserted in the specified order, immediately after any required elements. The CUE user may delete default elements and may insert other elements before them and between them.

Syntax

```
<default-story-elements>
  <ref-story-element-type/>+
</default-story-elements>
```

Child Elements

`ref-story-element-type`: section 2.31.

Only one form of the **`ref-story-element-type`** element may be used: **Standard ref-story-element-type** (section 2.31.2).

2.15 elements

A container for a storyline content model. A storyline content model contains rules and constraints defining the story element types that are allowed and/or required in a storyline template or complex story element type.

Syntax

```
<elements>
  <base-story-element>...</base-story-element>?
  <constraints>...</constraints>

  <default-story-elements>...</default-story-elements>?
  ANY-FOREIGN-ELEMENT*
</elements>
```

Child Elements

base-story-element: section 2.9, **constraints**: section 2.11, **default-story-elements**: section 2.14.

Only one form of the **constraints** element may be used: **Element constraints** (section 2.11.2).

2.16 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 CUE 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 CUE, 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.

2.17 field

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

2.17.1 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"
    id="NCName"?
    type="basic"
    mime-type="text"
    search="(store|index-store)"?
  >
  <array/>?
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
  <rep:representations>...</rep:representations>?
  <options>...</options>?
</field>
```

Child Elements

`array`: section 2.8, **`constraints`**: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7, **`rep:representations`**: section 2.39, **`options`**: section 2.25.

Only one form of the **`constraints`** element may be used: **Text constraints** (section 2.11.5).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

`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 CUE 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.

`search="(store|index-store)"` (optional)
Specifies that this attribute's parent **`field`** element is to be included in search results, and optionally indexed.

Allowed values are:

`store`
The field's content will only be included in search results.

`index-store`
The field's content will be both indexed and included in search results.

2.17.2 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"
    id="NCName"?
    type="basic"
    mime-type="text"
```

```
    search="(store|index-store)"?
  >
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
  <rep:representations>...</rep:representations>?
</field>
```

Child Elements

`constraints`: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7, **`rep:representations`**: section 2.39.

Only one form of the **`constraints`** element may be used: **Text constraints** (section 2.11.5).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

`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 CUE 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.

`search="(store|index-store)"` (optional)
Specifies that this attribute's parent **`field`** element is to be included in search results, and optionally indexed.

Allowed values are:

`store`
The field's content will only be included in search results.

`index-store`
The field's content will be both indexed and included in search results.

2.17.3 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 CUE.

Syntax

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

Child Elements

`array`: section 2.8, **`constraints`**: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7, **`options`**: section 2.25.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.4 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 CUE.

Syntax

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

Child Elements

`constraints`: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.5 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 CUE by graphic components called **tokens**.

Syntax

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

Child Elements

`array`: section 2.8, **`constraints`**: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7, **`options`**: section 2.25.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

`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 CUE) from which the user can choose one value. You can include the name of the current publication in the resource URI using the placeholder **`{publication}`**. If the current publication is called **`mypub`**, for example, then Content Store will convert the URL **`http://mycompany.com/myfeed?publication={publication}`** to **`http://mycompany.com/myfeed?publication=mypub`**. This makes it possible for the addressed service to send different feed content for different publications.

`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**.

2.17.6 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 CUE by graphic components called **tokens**.

Syntax

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

Child Elements

constraints: section 2.11, **parameter**: section 2.27, **annotation**: section 2.7.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

`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 CUE) from which the user can choose one value. You can include the name of the current publication in the resource URI using the placeholder **`{publication}`**. If the current publication is called **`mypub`**, for example, then Content Store will convert the URL **`http://mycompany.com/myfeed?publication={publication}`** to **`http://mycompany.com/myfeed?publication=mypub`**. This makes it possible for the addressed service to send different feed content for different publications.

`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`**.

2.17.7 Complex field

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

Syntax

```
<field
    name="NCName"
    id="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.

id="*NCName*" (optional)
The id of the **field** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.8 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"
    id="NCName"?
    type="date"
  >
  <array/>?
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
  <options>...</options>?
</field>
```

Child Elements

array: section 2.8, **constraints**: section 2.11, **parameter**: section 2.27, **annotation**: section 2.7, **options**: section 2.25.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.9 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"
    id="NCName"?
    type="date"
  >
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
</field>
```

Child Elements

`constraints`: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.10 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"
    id="NCName"?
    type="enumeration"
    multiple="(true|false)"?
  >
  <array/>?
  <enumeration>...</enumeration>+
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
  <options>...</options>?
</field>
```

Child Elements

`array`: section 2.8, **`enumeration`**: section 2.16, **`constraints`**: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7, **`options`**: section 2.25.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*`NCName`*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

`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 CUE. If set to **`false`** or unspecified then the field will only accept a single choice; it is displayed as a combo box in CUE.

2.17.11 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"
    id="NCName"?
    type="enumeration"
    multiple="(true|false)"?
  >
  <enumeration>...</enumeration>+
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
</field>
```

Child Elements

`enumeration`: section 2.16, **`constraints`**: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7.

Only one form of the **`constraints`** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*`NCName`*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

`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 CUE. If set to **`false`** or unspecified then the field will only accept a single choice; it is displayed as a combo box in CUE.

2.17.12 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 Store will try to retrieve a value from the **`field`** it inherits from.

Syntax

```
<field
    name="NCName"
    id="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.

`id="`*`NCName`*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

`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`**.

2.17.13 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"
    id="NCName"?
    type="link"
  >
  <array/>?
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
  <relation>...</relation>
  <options>...</options>?
</field>
```

Child Elements

`array`: section 2.8, **`constraints`**: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7, **`relation`**: section 2.33, **`options`**: section 2.25.

Only one form of the **`constraints`** element may be used: **Link constraints** (section 2.11.3).

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.

`id="`*`NCName`*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.14 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"
    id="NCName"?
```

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

Child Elements

`constraints`: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7, **`relation`**: section 2.33.

Only one form of the **`constraints`** element may be used: **Link constraints** (section 2.11.3).

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.

`id="`*`NCName`*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.15 Number field

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

Syntax

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

```
    <options>...</options>?
  </field>
```

Child Elements

`array`: section 2.8, **`format`**: section 2.20, **`constraints`**: section 2.11, **`parameter`**: section 2.27, **`annotation`**: section 2.7, **`options`**: section 2.25.

Only one form of the **`constraints`** element may be used: **Number constraints** (section 2.11.4).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.16 Number field (Simplified)

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

Syntax

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

Child Elements

format: section 2.20, **constraints**: section 2.11, **parameter**: section 2.27, **annotation**: section 2.7.

Only one form of the **constraints** element may be used: **Number constraints** (section 2.11.4).

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.

id="*NCName*" (optional)
The id of the **field** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.17 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"
    id="NCName"?
    type="schedule"
  >
  <array/>?
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
  <options>...</options>?
</field>
```

Child Elements

array: section 2.8, **constraints**: section 2.11, **parameter**: section 2.27, **annotation**: section 2.7, **options**: section 2.25.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.11.1).

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.

id="*NCName*" (optional)
The id of the **field** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.18 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"
    id="NCName"?
    type="schedule"
  >
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
</field>
```

Child Elements

constraints: section 2.11, **parameter**: section 2.27, **annotation**: section 2.7.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.11.1).

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.

id="*NCName*" (optional)
The id of the **field** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.19 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"
    id="NCName"?
    type="uri"
  >
  <array/>?
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
  <options>...</options>?
</field>
```

Child Elements

array: section 2.8, **constraints**: section 2.11, **parameter**: section 2.27, **annotation**: section 2.7, **options**: section 2.25.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.11.1).

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.

id="*NCName*" (optional)
The id of the **field** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.17.20 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"
    id="NCName"?
    type="uri"
  >
  <constraints>...</constraints>?
  ANY-FOREIGN-ELEMENT*
  <parameter/>*
  <annotation>...</annotation>*
</field>
```

Child Elements

constraints: section 2.11, **parameter**: section 2.27, **annotation**: section 2.7.

Only one form of the **constraints** element may be used: **Boolean constraints** (section 2.11.1).

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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

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

2.18 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.

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 CUE.

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

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.

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

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.

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.

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'`.

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

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

Syntax

```
<field
    name="NCName"
    id="NCName"?
    ( type="basic" mime-type="text" search="(store|index-store)" |
 type="(boolean|uri|schedule)" | type="number" | type="link" | type="enumeration"
 multiple="(true|false)"? | type="date" | type="collection" src="text" mime-
type="text" select="(content|title|locator)" select="link" linkrel="text" )
  >
  <array/>?
```

```
  (<constraints>...</constraints>?ANY-FOREIGN-ELEMENT*<parameter/>*<annotation>...</
annotation>*<rep:representations>...</rep:representations>?|<constraints>...</
constraints>?ANY-FOREIGN-ELEMENT*<parameter/>*<annotation>...</
annotation>*|<format>...</format>?<constraints>...</constraints>?ANY-FOREIGN-
ELEMENT*<parameter/>*<annotation>...</annotation>*|<constraints>...</constraints>?ANY-
FOREIGN-ELEMENT*<parameter/>*<annotation>...</annotation>*<relation>...</
relation>|<enumeration>...</enumeration>+<constraints>...</constraints>?ANY-FOREIGN-
ELEMENT*<parameter/>*<annotation>...</annotation>*|<constraints>...</constraints>?ANY-
FOREIGN-ELEMENT*<parameter/>*<annotation>...</annotation>*|<constraints>...</
constraints>?ANY-FOREIGN-ELEMENT*<parameter/>*<annotation>...</annotation>*)

  <options>...</options>?
</field>
```

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 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>
```

- 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>
```

- 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>
```

- 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.

`id="`*NCName*`"` (optional)
The id of the **`field`** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

`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 CUE 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.

search="(store|index-store)" (optional)
Specifies that this attribute's parent **field** element is to be included in search results, and optionally indexed.

Allowed values are:

store
The field's content will only be included in search results.

index-store
The field's content will be both indexed and included in search results.

type="(boolean|uri|schedule)"
Defines the type of the field.

Allowed values are:

boolean

uri

schedule

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

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

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 CUE. If set to **false** or unspecified then the field will only accept a single choice; it is displayed as a combo box in CUE.

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

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 CUE) from which the user can choose one value. You can include the name of the current publication in the resource URI using the placeholder **{publication}**. If the current publication is called **mypub**, for example, then Content Store will convert the URL **http://mycompany.com/myfeed?publication={publication}** to **http://mycompany.com/myfeed?publication=mypub**. This makes it possible for the addressed service to send different feed content for different publications.

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**.

2.19 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 2.17.

The following forms of the **field** element may be used: **Complex field** (section 2.17.7), **Basic field** (section 2.17.1), **Boolean field** (section 2.17.3), **URI field** (section 2.17.19), **Schedule field** (section 2.17.17), **Number field** (section 2.17.15), **Link field** (section 2.17.13), **Enumeration field** (section

2.17.10), **Date field** (section 2.17.8), **Collection field** (section 2.17.5), **Inherited field** (section 2.17.12).

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.

2.20 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>
```

2.21 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>
```

2.22 maximum

Either the maximum value that may be entered in a numeric **`field`** or the maximum number of story elements that may be inserted into a storyline or complex story element.

Syntax

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

2.23 mime-type

An allowed MIME type for the binary data referenced in a link **`field`**. As well as the MIME type, this element may contain a priority setting, separated from the MIME type by a semicolon as follows:

```
<mime-type>image/jpeg; priority=1</mime-type>
```

This priority setting can be used by CUE to determine which content type to use in cases where a dropped binary object is an allowed binary type for more than one content type. CUE then chooses the content type with the highest priority setting.

Syntax

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

2.24 minimum

Either the minimum value that may be entered in a numeric **`field`** or the maximum number of story elements that may be inserted into a storyline or complex story element.

Syntax

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

2.25 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 CUE 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 2.17.

The following forms of the **field** element may be used: **Basic field (Simplified)** (section 2.17.2), **Boolean field (Simplified)** (section 2.17.4), **URI field (Simplified)** (section 2.17.20), **Schedule field (Simplified)** (section 2.17.18), **Number field (Simplified)** (section 2.17.16), **Link field (Simplified)** (section 2.17.14), **Enumeration field (Simplified)** (section 2.17.11), **Date field (Simplified)** (section 2.17.9), **Collection field (Simplified)** (section 2.17.6).

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**.

2.26 panel

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

Syntax

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

Child Elements

field: section 2.17, **ref-field-group**: section 2.29.

The following forms of the **field** element may be used: **Complex field** (section 2.17.7), **Basic field** (section 2.17.1), **Boolean field** (section 2.17.3), **URI field** (section 2.17.19), **Schedule field** (section 2.17.17), **Number field** (section 2.17.15), **Link field** (section 2.17.13), **Enumeration field** (section 2.17.10), **Date field** (section 2.17.8), **Collection field** (section 2.17.5), **Inherited field** (section 2.17.12).

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.

2.27 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 CUE or used by CUE in any way. It can, however be retrieved by template developers and used for a variety of purposes.

The element can also be used inside **ui:decorator** in which case it defines a parameter (name/value pair) to be used by the decorator defined in this element's parent **ui:decorator** element.

Syntax

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

Attributes

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

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

2.28 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.

2.29 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.

2.30 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.

2.31 ref-story-element-type

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

2.31.1 Required ref-story-element-type

A reference to one of the **story-element-type**s in this storyline template. This form of the **ref-story-element-type** element (which has an **allow-empty** attribute) may only appear as the child of **required-story-elements**.

Syntax

```
<ref-story-element-type
    name="text"
    allow-empty="(true|false)"?
  />
```

Attributes

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

allow-empty="(true|false)" (optional)
If set to **true** then the CUE user is not required to enter a value in this **story-element-type**. It can be left empty. This option effectively allows you to place "optional" story elements before required story elements. Such "optional" elements are not really optional (they still have to be present in the specified position) but since they can be left empty they can easily be made invisible in the output content.
```

2.31.2 Standard ref-story-element-type

A reference to one of the **story-element-type**s in this storyline template.

Syntax

```
<ref-story-element-type
    name="text"
  />
```

Attributes

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

2.32 ref-storyline-template

A reference to one of the storyline templates defined in the **storyline-template** shared resource.

Syntax

```
<ref-storyline-template
    name="text"
  />
```

Attributes

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

2.33 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 Store plug-ins may define other values for this element.

Syntax

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

Examples

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

2.34 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 **relations** "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 CUE 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 CUE user can drop content items (in this case the images he wants to appear in these locations).

Syntax

```
<relation-type
    name="NCName"
    id="NCName"?
    max="integer"?
  >
  <allow-content-types>...</allow-content-types>*
  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.

id="*NCName*" (optional)
The id of the **relation-type** element. Set this attribute if you want to be able to reference this element using XInclude. Otherwise it is not required.

max="*integer*" (optional)
The maximum number of allowed relations in this relation type.

2.35 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.

2.36 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/>
```

2.37 rep:output

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

2.37.1 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.

2.37.2 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 CUE 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 CUE 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.

2.38 rep:representation

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

2.38.1 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

2.38.2 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 CUE, 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 2.37.

Only one form of the **rep:output** element may be used: **Derived Image Version output** (section 2.37.1).

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).

2.38.3 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 2.37, **`rep:crop`**: section 2.36, **`rep:resize`**: section 2.40.

Only one form of the **`rep:output`** element may be used: **Image Version output** (section 2.37.2).

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.

2.39 rep:representations

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

2.39.1 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 2.38.

Only one form of the **`rep:representation`** element may be used: **Custom representation** (section 2.38.1).

Attributes

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

2.39.2 Image Version rep:representations

Defines a set of different versions of an image that are created and maintained using the standard Content Store **`image-versions`** mechanism. This mechanism allows the CUE 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 2.38.

The following forms of the **rep:representation** element may be used: **Image Version representation** (section 2.38.3), **Derived Image Version representation** (section 2.38.2).

Attributes

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

2.39.3 Inherited Image Version rep:representations

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

This variant of the **representations** element should only be used in a **field** that is the child of a **summary** element.

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"
    inherit="true"
  />
```

Attributes

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

inherit="true"
This attribute may only be set to **true** in a **field** that is the child of a **summary** element. If it is set to **true** then image representations are inherited from the content item field with the same name as this field, and this element may not contain any **representation** elements of its own.

2.40 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/>
```

2.41 required

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

Syntax

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

2.42 required-story-elements

Elements that are **required** in a storyline or complex story element. The specified elements must appear at the beginning of the storyline or complex story element, in the specified order. They are inserted by default by CUE when the storyline or complex story element is created, and the CUE user is not allowed to delete them or insert other elements either before or between them.

Syntax

```
<required-story-elements>
  <ref-story-element-type/>+
</required-story-elements>
```

Child Elements

`ref-story-element-type`: section 2.31.

Only one form of the **`ref-story-element-type`** element may be used: **Required ref-story-element-type** (section 2.31.1).

2.43 story-element-type

Defines a CUE **story element type**. A story element type defines a particular type of story element. It defines:

- The **`field`**s a story element is composed of.
- How the story element is presented in CUE.

The following forms of the **`field`** element may be used: Basic field, Number field, Enumeration field, Date field, Link field, Custom field and Boolean field. URI fields may also be used, but only as **settings** fields (that is, **`field`** elements with child **`<ui:editor-style ="settings">`** elements).

Syntax

```
<story-element-type
    name="NCName"
  >
```

```
    <field>...</field>*
    <allow-story-element-types>...</allow-story-element-types>*
    <elements>...</elements>?
    ANY-FOREIGN-ELEMENT*
  </story-element-type>
```

Attributes

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

2.44 storyline-template

Defines the structure of a particular type of storyline. This structure determines the editing options offered to users in the CUE editor.

Syntax

```
<storyline-template
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT*
  <elements>...</elements>
</storyline-template>
```

Attributes

name="*NCName*"
The name of the **storyline-template** element.

2.45 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 CUE 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.

> You cannot use rich text fields (that is, **basic** fields with the MIME type **application/xhtml+xml**) in summaries.

Syntax

```
<summary>
```

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

Child Elements

`field`: section 2.17, **`ref-relation-type-group`**: section 2.30.

The following forms of the **`field`** element may be used: **Complex field** (section 2.17.7), **Basic field** (section 2.17.1), **Boolean field** (section 2.17.3), **URI field** (section 2.17.19), **Schedule field** (section 2.17.17), **Number field** (section 2.17.15), **Link field** (section 2.17.13), **Enumeration field** (section 2.17.10), **Date field** (section 2.17.8), **Collection field** (section 2.17.5), **Inherited field** (section 2.17.12).

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>
```

2.46 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.

{random}
Replaced by a random string used to allow the url of an article to change whenever the article is changed.

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 Store 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.

2.47 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 publication-type

The **publication-type** schema defines the content of a **publication-type** resource.

The purpose of a **publication-type** resource is to define a **publication type**, used when creating publications.

Namespace URI

The namespace URI of the **publication-type** schema is **http://xmlns.escenic.com/2019/publication-type**.

Root Element

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

3.1 auto-publish-storylines

Configures the auto-publish feature for storylines. Note that this setting can be overridden for individual publications in the **feature** publication resource (See autoPublishStorylines).

> Note that the auto-publishing feature only works in very tightly defined circumstances, in order to ensure that it cannot result in the unintentional publishing of content. For full details, see Auto-publishing.

Allowed values are:

disabled
: Auto-publishing is disabled in publications of this type.

full
: Auto-publishing is **fully** enabled in publications of this type. "Fully enabled" means that all published destination content items will be automatically republished if any upstream content item is published. If a destination content item is in the state **draft-published** (or any similar staged state) then it will only be republished if all of its storyline changes are inherited. If the storyline has been directly modified by a user since the last time it was published, then it will not be republished. This limitation prevents unintentional publishing of local changes. Other kinds of local change, such as changes to a content item's relations or tagging changes will not block autopublishing.

cautious
: Auto-publishing is **partly** enabled in publications of this type. The auto-publishing feature's behavior is almost the same as for the **enabled** setting, but with the following difference: a destination content item will never be automatically republished if it has ever been modified since it was first published. In other words, this is a more cautious setting, offering more robust protection against unintentional publishing of unreviewed content.

  cautious is the default setting.

Syntax

```
<auto-publish-storylines>
  (disabled|full|cautious)
</auto-publish-storylines>
```

3.2 feature

Specifies whether or not a feature should be enabled.

Syntax

```
<feature
    name="(createDefaultInbox|createDefaultFrontpage|createIncomingSection)"
    enabled="(true|false)"
  />
```

Attributes

`name="(createDefaultInbox|createDefaultFrontpage|createIncomingSection)"`
The name of the feature.

Allowed values are:

`createDefaultInbox`
If enabled, a default inbox is created for each section in the publication.

`createDefaultFrontpage`
If enabled, a default front page is created for each section in the publication.

`createIncomingSection`
If enabled, a section with the unique name **`ece_incoming`** is included in the publication and used as the default home section for new content items.

If disabled, the publication's root section is used as the default home section.

`enabled="(true|false)"`
Enables or disables the feature.

3.3 features

Specifies the features to be provided by publications of this type.

Syntax

```
<features>
  <feature/>+<auto-publish-storylines>...</auto-publish-storylines>?
</features>
```

3.4 publication-type

Defines a publication type.

Syntax

```
<publication-type
    name="NCName"
  >
  <features>...</features>?
  ANY-FOREIGN-ELEMENT*
</publication-type>
```

Attributes

name="*NCName*"
The name of the publication type.

4 container-type

The **container-type** schema defines the content of a **container-type** resource. The purpose of a **container-type** resource is to define a specific type of content item container.

Namespace URI

The namespace URI of the **container-type** schema is **http://xmlns.escenic.com/2019/container-type**.

Root Element

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

4.1 allowed-destination

Specifies a publication/content type combination that may inherit content from one of a container's **first-destination**s (base content items). When a CUE user creates a variant of a base content item, a dialog displaying some or all of its **allowed-destination**s is displayed, from which the user can select the required option. How many of the destinations are actually displayed for the user depends on his access rights and the organizational units selected in his CUE settings.

Syntax

```
<allowed-destination
    content-type="NCName"
    publication="string"
    max="int"?
    inheritance-mode="(inherit|reconcile)"?
  />
```

Attributes

content-type=" ***NCName*** **"**
The content type of this allowed destination.

publication=" ***string*** **"**
The publication of this allowed destination.

max="*int*" (optional)
The maximum number of times this **allowed-destination** may be used. Once this limit is reached, CUE will no longer offer it as a possible destination.

inheritance-mode="(inherit|reconcile)" (optional)
Specifies what kind of inheritance is to be used for this destination. The value set overrides any inheritance mode set on the **container-type** or **first-destination** elements. The allowed values are:

inherit
Inheriting storylines cannot be modified. To modify an inheriting storyline, the user must explicitly break its inheritance. The storyline will then effectively become a copy of its

base storyline's current state. It can now be freely modified, but will no longer reflect changes made to the base storyline.

reconcile
Inheriting storylines can be modified. It is possible to add new story elements and to modify or delete existing inherited elements. Adding a new story element will not affect other story elements inherited from the base storyline; they will continue to reflect changes made in the base storyline. Similarly, editing or deleting an inherited story element will only break inheritance for that story element: all other inherited story elements will continue to reflect changes made in the base storyline.

4.2 container-type

Defines a container type. Although the **container-type** syntax for its **field** and **relation-type** children is very permissive, to get correctly working containers, a **container-type** should have at least the following child elements:

- One field, used as the container name or **slug**. The container definition must then include a **ui:title-field** element to identify the slug field. The slug and any other container fields are displayed on their own editor tab (called **Main**) in CUE when any of the container's content items are edited.
- One **relation-type**, which must have the name **com.escenic.container.items**. This defines a relation drop zone that is also displayed on the CUE **main** editor tab, and is used to hold relations to all of the container's content items. This allows CUE users to navigate between the content items in a container. CUE automatically adds relations to the drop zone when content items are added to the container.

Syntax

```
<container-type
    name="NCName"
    inheritance-mode="(inherit|reconcile)"?
  >
  <first-destination/>+
  <ct:field>...</ct:field>*
  <ct:relation-type>...</ct:relation-type>*
  <copy-fields>...</copy-fields>?
  ANY-FOREIGN-ELEMENT*
</container-type>
```

Child Elements

ct:field: section 2.17, **first-destination**: section 4.5, **ct:relation-type**: section 2.34, **copy-fields**: section 4.3.

The following forms of the **ct:field** element may be used: **Complex field** (section 2.17.7), **Basic field** (section 2.17.1), **Boolean field** (section 2.17.3), **URI field** (section 2.17.19), **Schedule field** (section 2.17.17), **Number field** (section 2.17.15), **Link field** (section 2.17.13), **Enumeration field** (section 2.17.10), **Date field** (section 2.17.8), **Collection field** (section 2.17.5), **Inherited field** (section 2.17.12).

Attributes

name=" ***NCName*** **"**
The name of the container type.

inheritance-mode="(inherit|reconcile)" (optional)
Specifies what kind of inheritance is to be used for this container type. The allowed values are:

inherit (default)
Inheriting storylines cannot be modified. To modify an inheriting storyline, the user must explicitly break its inheritance. The storyline will then effectively become a copy of its base storyline's current state. It can now be freely modified, but will no longer reflect changes made to the base storyline.

reconcile
Inheriting storylines can be modified. It is possible to add new story elements and to modify or delete existing inherited elements. Adding a new story element will not affect other story elements inherited from the base storyline; they will continue to reflect changes made in the base storyline. Similarly, editing or deleting an inherited story element will only break inheritance for that story element: all other inherited story elements will continue to reflect changes made in the base storyline.

4.3 copy-fields

Specifies the name of a field that is to be copied from a base content item to a variant. The value is copied, not inherited and can therefore be edited in the variant content item. The **source-content-type** and **target-content-type** specify the context in which the copy operation should be performed.

Syntax

```
<copy-fields
    source-content-type="NCName"
    target-content-type="NCName"
    source-field="text"
    target-field="text"?
  />
```

Attributes

source-content-type=" ***NCName*** **"**
Only copy the specified **source-field** if the base content item is of this type.

target-content-type=" ***NCName*** **"**
Only copy the specified **source-field** if the variant content item to be created is of this type.

source-field=" ***text*** **"**
Allowed values are:

metadata.authors
Specifies that the base content item's authors are to be copied.

metadata.tags
Specifies that the base content item's tags are to be copied.

metadata.relations
Specifies that all the base content item's relations are to be copied. If **target-field** is not specified, then each relation type is copied to the corresponding relation type in the

target content item. Any relation types that do not exist in the target are not copied. If **target-field** is specified, then all relations of all types are copied to the specified type in the target content item.

metadata.relations. ***relation-type***
Specifies that all the base content item's relations of the specified type are to be copied. If **target-field** is not specified, then the relations are copied to the corresponding relation type in the target content item (if it exists). If **target-field** is specified, then the relations are copied to the specified type in the target content item.

metadata.homeSection
Specifies that the base content item's home section is to be copied.

If **target-field** is not specified, then a section with the same unique name in the target publication will be used as the home section of the target content item. If a section with the same unique name does not exist, then a section with the same unique name as the parent section will be used as the home section.

If **target-field** is specified, then the section with the specified unique name will be used as the home section of the target content item. If the section does not exist or the user does not have access to it, no home section will be set.

field. ***field-name***
Specifies that a named field (a standard content item field, not a story element) is to be copied. If **target-field** is not defined, the field must exist in both the source and target content types.

target-field=" ***text*** **" (optional)**
Allowed values are:

field. ***field-name***
The field to which **source-field** should be copied.

metadata.relations. ***relation-type***
A named relation type to which the relations specified in **source-field** should be copied.

metadata.homeSection. ***section-unique-name***
The unique name of the section to be used as the home section of the target content item.

If not specified, **source-field** is used as **target-field**.

4.4 constraints

Contains a list of one or more **allowed-destination** elements specifying the destinations that may inherit content from a given **first-destination**. If a **first-destination** has a **constraints** element, then only the listed destinations are made available to the CUE user when adding subsequent destinations. If no **constraints** element is specified, then the CUE user is allowed to select subsequent destinations from all the publications and content types to which he/she has access.

Syntax

```
<constraints>
  <section 4.1/>+
</constraints>
```

4.5 first-destination

Specifies a possible publication/content type combination for the base content item of a container of this type. A **container-type** can have many such **first-destination** elements, so that it can be used across many publications and organizational units. When a user creates a container in CUE, a dialog displaying some or all of these first destinations is displayed, from which the user can select the required option. How many of the destinations are actually displayed for the user depends on his access rights and the organizational units selected in his CUE settings.

Syntax

```
<first-destination>
    content-type="NCName"
    publication="string"
    inheritance-mode="(inherit|reconcile)"?
  <section 4.4>...</constraints>*
</first-destination>
```

Attributes

content-type=" ***NCName*** **"**
The content type of this first destination.

publication=" ***string*** **"**
The publication of this first destination.

inheritance-mode="(inherit|reconcile)" (optional)
Specifies what kind of inheritance is to be used for this destination. The value set overrides any inheritance mode set on the **container-type** element. The allowed values are:

inherit
Inheriting storylines cannot be modified. To modify an inheriting storyline, the user must explicitly break its inheritance. The storyline will then effectively become a copy of its base storyline's current state. It can now be freely modified, but will no longer reflect changes made to the base storyline.

reconcile
Inheriting storylines can be modified. It is possible to add new story elements and to modify or delete existing inherited elements. Adding a new story element will not affect other story elements inherited from the base storyline; they will continue to reflect changes made in the base storyline. Similarly, editing or deleting an inherited story element will only break inheritance for that story element: all other inherited story elements will continue to reflect changes made in the base storyline.

5 content-card

The **content-card** schema defines the structure of a content card definition. A content card definition specifies what content item information is displayed in a CUE content card, and where it is displayed.

You can if you wish create a number of different content card definitions and then use different content card definitions for different content types.

You only need to create content card definitions if you want to define custom content card structures: if no specific content card structure is requested then CUE will use its default structure.

Namespace URI

The namespace URI of the **content-card** schema is **http://xmlns.stibodx.com/2020/content-card**.

Root Element

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

5.1 content-card

Defines a named content card structure.

Syntax

```
<content-card
    name="NCName"
  >
  <field>...</field>+
</content-card>
```

Attributes

name="*NCName*"
The name of this content card definition.

5.2 field

Contains a child **template** element defining what is to appear in one area of the content card.

Syntax

```
<field
    name="(title|summary|date)"
  >
  ANY-FOREIGN-ELEMENT*<template>...</template>
</field>
```

Attributes

name="(title|summary|date)"
The name of the content card area to be defined.

Allowed values are:

title
The title area at the top of the card. Whatever appears here will be displayed as a title in a large font.

summary
The main area of the card.

date
The date area of the card, at the bottom right. Whatever appears here will be displayed in a small font.

5.3 template

A Twig template defining what is to appear in this area of the content card. The Twig template is passed to CUE for processing: the templates can only contain references to values that are available to CUE as the properties of an object called **item**. The content item title, for example, can be referenced as follows:

```
{{ item.title }}
```

For detailed information about what the templates can contain, see Twig Template Processing.

Syntax

```
<template>
  text
</template>
```

6 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.

6.1 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.

6.2 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).

6.3 imageDef

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

Syntax

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

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

6.4 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.

6.5 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.

6.6 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.

6.7 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.

6.8 parameter

> Not currently used.

Syntax

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

Attributes

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

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

6.9 pluginGenerator

Not currently used.

Syntax

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

Attributes

class="..."
Not currently used.

6.10 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.

7 layout-group

The **layout-group** schema defines the structure of the allowed CUE **layout-group** publication resource. The purpose of the **layout-group** resource is to define a set of layouts for use on CUE section pages. These layouts are composed of **group** and **area** elements, and include external references to **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.

7.1 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>
```

7.2 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.

7.3 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.

7.4 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.

7.5 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>
```

7.6 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.

7.7 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.

7.8 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.

8 interface-hints

The **interface-hints** schema defines additional elements used by CUE components such as the CUE editor and the Content Store. They can be included at various points in CUE publication resources such as the **content-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 by application user interfaces and the 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**.

8.1 additional-editor

When it appears as the child of a **summary** element in a **content-type** resource, this element specifies that some additional content is to be displayed in the summary in CUE. The additional content must be provided by a CUE web component specified in the CUE configuration files. The content of this element must be the name of the web component configured in CUE.

Syntax

```
<additional-editor>
  text
</additional-editor>
```

8.2 alignment

Controls alignment options for in-line images.

Syntax

```
<alignment
    enabled="(true|false)"?
    default="(left|center|top|right)"?
  />
```

Attributes

enabled="(true|false)" (optional)
Enables/disables an alignment option for in-line images.

Allowed values are:

true
The inline image alignment option is enabled: an alignment option will be displayed in the in-line image **Properties** dialog in CUE. (Default)

false
The inline image alignment option is disabled: no alignment option will be displayed in the in-line image **Properties** dialog in CUE.

default="(left|center|top|right)" (optional)
Determines the default alignment used if no selection is made by the CUE user. If this attribute is not specified and the user makes no selection then no alignment is performed.

You can set this attribute even if **enabled='false'**. In this case whatever alignment you specify as the default is fixed and cannot be modified by the CUE user.

Allowed values are:

left

center

top

right

8.3 allow-source-editor

When it contains the value **true**, this element specifies that source editing of rich text fields should be enabled in CUE. When it contains **false** (the default), it specifies that source editing should not be allowed. A **allow-source-editor** element only has any effect when it appears as the child of a **content-type**, a rich text **field** element or a complex / array **field** element. When it appears as the child of a **content-type** element, it applies to all rich text fields defined in that content type. When it appears as the child of a complex or array **field** element, it applies to all of that element's descendant rich text fields. When it appears as the child of a rich text **field** element, it only applies to the field itself.

allow-source-editor settings made at the field level can override settings made at the content type level. Within a nested field structure involving complex and array fields, settings made at the lowest level can override settings made at the top level. Settings made at intermediate levels will, however, be ignored. In other words, for a structure like this:

content type
 array field
  complex field
   rich text field

allow-source-editor elements inserted at levels 1, 2, and 4 will work. **allow-source-editor** elements inserted at level 3 (or any further intermediate levels) will be ignored.

Note that this element only affects the CUE rich text editor, not the storyline editor.

Syntax

```
<allow-source-editor>
  (true|false)
</allow-source-editor>
```

8.4 auto-complete

Identifies an auto-complete filter that may be used within a search filter definition. An **auto-complete** element may only appear as the child of a **filter** element in a search filter definition. It causes the field representing the filter in CUE to be given type-ahead functionality (assuming that the **filter** element also has child **term** elements containing Lucene terms providing the information needed to implement type-ahead).

Syntax

```
<auto-complete>
  text
</auto-complete>
```

8.5 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 CUE. 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>
```

8.6 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 **lang** attribute must be used to specify the language of each **boolean-label**.

Syntax

```
<boolean-label
    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

`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`**.

8.7 break-externally-owned

A visual hint on a **`transition`** telling the client that when making this transition, the content will be owned by an external system (CUE Print) and this should thus be indicated in the UI.

Syntax

```
<break-externally-owned/>
```

8.8 content-card

When it appears as the child of a **`content-type`** element in a **`content-type`** resource this element specifies the name of a content card definition that is to be used when displaying content cards for content items of this type.

Syntax

```
<content-card>
  text
</content-card>
```

8.9 content-length-constraint

A named set of length constraints that can be applied to storyline content items.

Syntax

```
<content-length-constraint
```

```
    name="text"
    default="text"?
  >
  <minchars>...</minchars>?<maxchars>...</maxchars>?<minwords>...</
minwords>?<maxwords>...</maxwords>?
</content-length-constraint>
```

Attributes

`name="`*text*`"`
The name of the content length constraints.

`default="`*text*`"` (optional)
If present and set to "yes", then this constraint set is used as the default. If more than one **`content-length-constraint`** element in a **`content-length-restrictions`** set has **`default="yes"`** set, then the first one is used as the default.

8.10 content-length-restrictions

When it appears as the child of an **`elements`** element in a storyline template, this element is a container for a set of named content length constraints that can be applied to storyline content items based on the template. One of these named sets can then be selected by the CUE user. In order for this to be possible, the content type definition must contain a text field definition containing a **`story-size`** element.

Syntax

```
<content-length-restrictions>
  <content-length-constraint>...</content-length-constraint>+
</content-length-restrictions>
```

8.11 count

When it appears as the child of a **`story-element-type`** element or of a **`field`** element in a story element type definition, this element specifies that CUE is to keep count of the characters and words in the story element or field. The counts maintained by CUE may be displayed below the story element or field and/or added to a total for the whole storyline and displayed on a **Metrics** metadata panel.

Syntax

```
<count
    show="(true|false)"
    for="text"?
  >
  <minchars>...</minchars>?<maxchars>...</maxchars>?<minwords>...</
minwords>?<maxwords>...</maxwords>?
</count>
```

Attributes

`show="(true|false)"`
Determines whether or not the count is displayed below the story elements or fields.

Allowed values are:

`true`
The count is displayed.

`false`
The count is not displayed.

`for="`*`text`*`"` (optional)
A space-separated list of summary identifiers, specifying the CUE **Metrics** panel summaries to which the count is to be added. The summary identifiers must have been defined in a CUE **`storylineMetrics`** configuration.

8.12 custom-editor

When it appears as the child of an **`elements`** element in a storyline template, this element defines a custom editor panel to be displayed alongside the storyline editor when storylines of this type are edited in CUE.

Syntax

```
<custom-editor
    webcomponent="text"
    width="text"
    minresolution="text"
  />
```

Attributes

`webcomponent="`*`text`*`"`
The name of the web component that implements the custom editor panel. The name must match the name of a component defined in the **`customComponents`** section of a CUE configuration file.

`width="`*`text`*`"`
The width of the custom editor panel, specified in CSS units. The value specified must include the unit, which should normally be `%`. The custom editor panel will then occupy this percentage of the horizontal screen space available to the storyline editor in CUE.

`minresolution="`*`text`*`"`
A window width, specified in CSS units. The value specified must include the unit, which should normally be **`px`**. The custom editor panel will then only be displayed if the width of the whole CUE window is equal to the specified value or greater.

8.13 dam-status

When it appears as the child of a **`field`** element, this element specifies that the field is to be used to store DC-X status information. It is used by the CUE DC-X integration extension.

Syntax

```
<dam-status/>
```

8.14 decorator

Defines a **decorator** for this element's parent **`content-type`** or **`group`** element. Decorators are Java programming constructs that can be used by CUE 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" fail-on-error="(true|false)"?
  >
  <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).

`fail-on-error="(`true|false`)"` (optional)
If set to **`true`**, then the Content Store will throw an exception if the decorator fails to decorate the article in the presentation layer.

The default value **`false`** means that any exceptions the decorator might throw while decorating article are logged and then ignored.

Setting this attribute to **`true`** only has meaning for **article decorators** (that is **`decorator`** elements that are children of **`content-type`** elements.

8.15 default

Indicates that the UI item should be the default when displayed.

Syntax

```
<default/>
```

8.16 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/>
```

8.17 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/>
```

8.18 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 **lang** attribute must be used to specify the language of each label.

Syntax

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

Attributes

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.

8.19 editor

Declares this element's parent **field** element to be a **field editor**. Field editors are user interface extensions for CUE. The field editor type (and therefore the target editor application) is indicated by the **type** attribute.

Syntax

```
<editor
```

```
    type="(enrichment-service|web-component)"
    name="text"
  />
```

Attributes

type="(enrichment-service|web-component)"
Specifies the field editor type:

Allowed values are:

enrichment-service
An **enrichment-service** field editor is a CUE interface extension, provided by an HTTP-based **enrichment service**. For more information about enrichment services and how to create them, see the latest **CUE User Guide**.

web-component
A **web-component** field editor is a CUE interface extension, written using HTML, Javascript and CSS. For more information about web components and how to create them, see the latest **CUE User Guide**.

name="*text*"
The name of the field editor. This name is mapped to the URL of the service provider or component.

8.20 editor-style

Determines how the item represented by the parent element will be displayed in editing applications. Currently this element may only appear in the following contexts:

- As the child of a **relation-type** element in a **content-type** resource, in which case it may only have one of the values **gallery** or **relation-gallery**: it is ignored if it has any other value. When used in this way, it causes relations of the type defined by the parent element to be treated like ordinary fields in CUE. That is, they can be displayed in the main area of the CUE editor together with fields, rather than in the metadata panel where relations are normally displayed. A **ref-relation-type** or **ref-relation-type-group** element must be used to specify **where** in the editor the relation type should appear. For details see http://docs.escenic.com/ece-pub-design-guide/7.17/displaying_relations_in_the_main_editor_area.html.

  The **gallery** setting may only be used for image relations. The **relation-gallery** setting, however, may be used for all relation types.
- As the child of a **filter** element in a **search-filter** resource, in which case it may have one of the following values:

  date authors sections tags

  In this case the **editor-style** affects the appearance and functionality of the search filter field represented by the parent **filter** element.
- As the child of a **field** element in a storyline element type definition. In this case it may only contain the value **settings**: it is ignored in any other location or if it has any other value. When used in this way, it causes the field defined by the parent element to be displayed in the storyline element's **Settings** dialog rather than as part of the storyline element itself. For an example of this usage, see http://docs.escenic.com/ece-pub-design-guide/7.17/table_element_type.html.

Syntax

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

8.21 element-flow

When it appears as the child of an **elements** element in a storyline template or complex element type, this element can be used to control what happens when the user presses the **Enter** key in a CUE storyline editor. Normally, pressing **Enter** causes a new story element of the base element type to be inserted. The **element-flow** element allows you to modify this behavior. If the insertion caret is in an element of the type specified with the **element** attribute, then an element of the type specified with the **next** attribute is inserted.

Syntax

```
<element-flow
    element="NCName"
    next="NCName"
  />
```

Attributes

element="*NCName*"
The element type that is to be given a special "next" element type.

next="*NCName*"
The element type that is to be inserted if **Enter** is pressed when the insertion caret is in an element of type **element**.

8.22 element-style

Indicates to CUE that a story element type is to be handled in a special way.

Allowed values are:

gallery
Indicates to CUE that this story element type is to be handled as a gallery.

image
Indicates to CUE that this story element type is to be handled as an image.

list
Indicates to CUE that this story element type is to be handled as a bulleted or numbered list.

video
Indicates to CUE that this story element type is to be handled as a video.

Syntax

```
<element-style>
  (gallery|image|list|video)
</element-style>
```

8.23 element-unwrap

When it appears as the child of an **elements** element in a storyline template or complex element type, this element defines a function that unwraps a wrapped story element of a specified type, and associates this operation with a keyboard shortcut. In the case of a bulleted list element type, for example, it can be used to specify that when the insertion caret is in a list item (that is, a **paragraph** story element inside a **list_bulleted** story element), pressing a given shortcut combination will remove it from the parent **list_bulleted** story element and move it up to the same nesting level (either to a parent list or to the storyline's root level as an ordinary paragraph).

It is possible to limit the operation of the unwrap operation so that the target story element cannot be unwrapped completely (that is, moved to the storyline's root level) using the **unwrap-mode** attribute.

The keyboard shortcut that will initiate the unwrap operation is specified in the child **keystroke** element.

The **element-unwrap** element can be used in any context where nesting of story elements is required, not just for lists.

Syntax

```
<element-unwrap
    element="NCName"
    content="NCName"?
    unwrap-mode="(default|keep-in-group)"?
  >
  <keystroke>...</keystroke>
</element-unwrap>
```

Attributes

element="*NCName*"
The wrapped story element type from which unwrap operations can be initiated by pressing a keyboard shortcut – and also the story element that will be unwrapped.

content="*NCName*" (optional)
If this attribute is specified, then the unwrap operation will only be executed if the content of the current story element matches the value of this attribute. In the context of lists, you can use this (for example) to only allow empty list items to be unwrapped, by specifying a **content** value of "".

unwrap-mode="(default|keep-in-group)" (optional)
If specified, determines how the unwrap action works.

Allowed values are:

default
No special limitations are applied to the operation of the unwrap action.

keep-in-group
The target story element cannot be moved to the storyline's root level.

8.24 element-wrap

When it appears as the child of an **elements** element in a storyline template or complex element type, this element defines a function that wraps one specified type of story element inside another type of story element, and associates this operation with a keyboard shortcut. In the case of a bulleted list element type, for example, it can be used to specify that when the insertion caret is in a list item (that is, a **paragraph** story element inside a **list_bulleted** story element), pressing a given shortcut combination will start a sublist by wrapping the list item in a new list story element.

The keyboard shortcut that will initiate the wrap operation is specified in the child **keystroke** element.

The **element-wrap** element can be used in any context where nesting of story elements is required, not just for lists.

Syntax

```
<element-wrap
    element="NCName"
    in="NCName"
    content="NCName"?
  >
  <keystroke>...</keystroke>
</element-wrap>
```

Attributes

element="*NCName*"
The story element type from which wrap operations can be initiated by pressing a keyboard shortcut, and that will be wrapped in another element type (specified with the **in** attribute).

in="*NCName*"
The story element type that will be used as a wrapper.

content="*NCName*" (optional)
If this attribute is specified, then the wrap operation will only be executed if the content of the current story element matches the value of this attribute. In the context of lists, you can use this to enable user-friendly, intuitive shortcuts for the creation of sublists. A **content** value of **"-"** together with a **SPACE** keystroke makes it possible for the user to start a new bulleted sublist by simply typing **"- "** at the start of a new list item.

8.25 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/>
```

8.26 field-set

Groups a set of fields together for display purposes in CUE. 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.

8.27 focus-field

Nominates one of the fields/story elements in a content type or storyline as the **focus field**. The focus field is the field/story element that gets the focus when a content item is first opened. **focus-field** must either be:

- The child of a **content-type** element, in which case the value it contains must be the name of one of the **field**s in that **content-type**.
- The child of the top **elements** element in a storyline template, in which case the value it contains must be the name of one of the **story element**s in the template.

For content types and storyline templates that do not contain a **focus-field** element, the first headline or title in the content item gets the focus, and if there is no headline or title, then the first field/story element in the content item gets the focus.

Syntax

```
<focus-field>
  text
</focus-field>
```

8.28 group

Defines a content type group. Content type groups are used by CUE 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 CUE, search results are filtered to show only content items of the types in the group.

Groups that contain **`ref-opensearch`** elements may also contain search results from an external system.

The **`group`**'s child **`ref-content-type`** and **`ref-opensearch`** elements reference the content types and/or OpenSearch external search definitions that belong to the group.

Syntax

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

Examples

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

8.29 group-prefix-label

When it appears as the child of a **`story-element-type`** element, this element defines a prefix to be added to the labels of all the story element's sub-elements. Specifying a **`group-prefix-label`** of **`"List"`** in a bulleted list story element type, for example, will result in the **`paragraph`** elements used as list items in bulleted lists appearing with the label "List Paragraph" rather than just "Paragraph".

Syntax

```
<group-prefix-label>
  text
</group-prefix-label>
```

8.30 hidden

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

Syntax

```
<hidden/>
```
```

8.31 icon

Contains the definition of an icon that can be used by CUE and other application user interfaces.

An **icon** element can appear as the child of one of the following elements:

- **content-type**, in which case it defines the icon to be used to represent content items of this type in the user interface.
- **publication-type**, in which case it defines the icon to be used to represent publications of this type in the user interface.
- **story-element-type**, in which case it defines the icon to be used to represent story elements of this type in the user interface.
- **annotations**, in which case it defines the icon to be used to represent this annotation.
- **macro**, in which case it defines the icon to be used to represent this macro in the user interface.
- **state** (in a workflow definition, see Custom Workflow Definitions). In this case it defines the icon to be used to represent this state in the user interface.
- **dashboard** (in a dashboard definition, see Dashboard Definitions). In this case it defines the icon to be used to represent a dashboard in the user interface.

In general, the **icon** element must contain the URI of an image you want to use as an icon. If the image is located on the CUE server, then you can specify a relative URL such as:

```
<ui:icon>icons/custom-gallery.png</ui:icon<
```

If the image is located elsewhere then you must specify an absolute URL, for example:

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

For good results in CUE, an icon image must have the following characteristics:

- PNG format
- Monochrome: black on a transparent background. CUE automatically inverts the colors where necessary.
- 32x32 pixels in size

You can in many contexts, however, use SVG vector drawings or Unicode characters as icons rather than images, in which case resolution is irrelevant.

When used as the child of a **content-type**, **publication-type**, **story-element-type** or **annotation** element the icon element may contain any of the following:

- A predefined icon name. For **publication-type**s, the following icons are provided:

  online
  print
  digital
  facebook
  twitter

  and for **story-element-type**s:

  embed

paragraph
headline
quote
photo
video
factbox
relation
leadtext
gallery
table
map
interview
question
answer

No predefined icons are provided for content types or annotations.

- The relative or absolute URL of an image (PNG file) or of an SVG file containing an icon definition.
- One or more Unicode characters. An icon will then be generated from the specified characters. You can specify the characters either literally or as numeric entities (**`&#x2191;`** for example).
- An in-line SVG icon definition - for example:

```
<ui:icon><![CDATA[
  <svg version="1.1"
       xmlns="http://www.w3.org/2000/svg"
       xmlns:xlink="http://www.w3.org/1999/xlink"
       id="Capa_1" x="0px" y="0px" viewBox="0 0 58 58"
       style="enable-background:new 0 0 58 58;" xml:space="preserve">
    <ellipse style="fill:#23A24D;" cx="29" cy="50" rx="29" ry="8"/>
    <path style="fill:#EBBA16;"
          d="M41.676,5.324L41.676,5.324c-7.098-7.098-18.607-7.098-25.706,0h0

 C9.574,11.72,8.855,23.763,14.282,31l14.541,21l14.541-21C48.792,23.763,48.072,11.72,41.676,5.324z
 M29,24c-3.314,0-6-2.686-6-6 s2.686-6,6-6s6,2.686,6,6S32.314,24,29,24z"/>
    </svg>]]>
</ui:icon>
```

Note that the inline SVG must be supplied as **`CDATA`**.

When used as the child of a **`macro`** element:

- The icon can be smaller (it will be scaled down to 16x16 pixels by CUE).
- 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 (**`&#x2191;`** for example).

When used as the child of a **`state`** element:

- The **`icon`** element must **not** contain an image URI. Instead, it must contain one of the following predefined icon names:

  approved
  canceled
  copy-editing
  deleted

draft
new
on-hold
proof-reading
proof-reading-not-done
published
ready
revert
reverted
slot-editing
source-editing
submitted
un-publish
working
fire
image-landscape
image-landscape-pencil
videofile
videofile-pencil
videofile-transcode
audiofile
audiofile-pencil
audiofile-transcode
general-transcode
paper-plane
pencil-text
document-pencil
podcast-sign
postit-stack
presenter
light-bulb
return-cross
rocket
shutter
camera
speech-bubbles
star-empty
star-solid
heart
heart-broken
star-polygon-check
stop-hand
hand-in-circle
no-park
no-park-hand
subtitles
open-captions
closed-captions
table-watch
bell-silent
bell-ringing

```
busy-bee
calender-sign
books
text-check
document-check
abc-check
user-check
magnifying-check
magnifying-star
trophy
upload-cloud
download-cloud
broadcast-tower
camera-profile
chart
meter
code-terminal
coffee
coins-wall
giving-money
color-droplet
color-paint-ruler
color-rgb
contrast
brightness-and-contrast
crop-sign
face
fibonacci
layout-sign
giving-hand
giving-hand-arrow
giving-hand-check
giving-hand-cross
glasses
hat-sun-glasses
megaphone
gps-pin
dog
hand-like
microscope
sitting-at-table
```

When used as the child of a **dashboard** element:

- The **icon** element may either contain an image URI or one of the following predefined icon names:

```
text
image
image print-ready
gallery
video
assignment
```

event
list
person
page
storyFolder
pdf
psd
jpg
svg
ai
doc
xls
unknown

Syntax

```
<icon
    use-in-playout="(true|false)" color="NCName"?
  >
  text
</icon>
```

Examples

- 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 **macro** element.

  ```
  <ui:icon>↑</ui:icon>
  ```

- This example creates an icon from one of the state icon images provided with the Content Store. The icon will be displayed in green. This form of the icon element is ony valid when used as the child of a **state** element in a workflow definition file. Both icon names (as opposed to URLs) and the **color** attribute only work when the **icon** element appears as the child of a **state** element.

  ```
  <ui:icon color="green">new</ui:icon>
  ```

Attributes

use-in-playout="(true|false)" (optional)
This attribute only has any meaning when used in an **icon** element that is the child of a **content-type**. In this context, when set to **true**, it indicates that when a content item of this type is published, this icon is to be displayed in the CUE playout bar instead of the usual publication icon.

color="*NCName*" (optional)
When the icon element appears as the child of a **state** element, this attribute can be used to specify the color of the icon. The following colors may be specified:

red
grey
pink
orange

purple
yellow
light-blue
blue
light-green
green

If no color is specified, then the icon will be displayed in its default color(s).

When the icon is used in other contexts (as the child of a **content-type** or **macro** element) this attribute is ignored.

8.32 inherits-from

Contains the name of the content type **field** element from which this element's parent **field** inherits its characteristics.

> This element is used to provide information about inheritance in field definitions returned by the Content Store web service. You cannot use it to invoke inheritance when defining content types. To do this you should use the **field** element's **inherits-from** attribute. For details, see Inherited field.

Syntax

```
<inherits-from>
  text
</inherits-from>
```

8.33 inline

Defines options that can be applied to in-line images included in this element's parent **field** element. This element only has any effect if it is specified as a child of a rich text field: that is, a **basic** field where the **mime-type** is set to **application/xhtml+xml**.

Syntax

```
<inline>
  <alignment/>?
</inline>
```

8.34 keystroke

Contains the definition of a key combination that can be used by application user interfaces to insert/invoke the content/functionality represented by this element's parent element, which may be any of the following:

macro
Specifies the keystroke combination that will execute the macro.

story-element
Specifies the CUE semantic shortcut keystroke to be associated with the parent **story-element**. This effectively defines a semantic shortcut for converting to this story element. In other words, adding the keystroke 'h' to the definition of a headline story element will mean that CUE users can (for example) convert a paragraph story element to a headline story element by pressing **Shift Shift t h**.

annotation
Specifies the keystroke combination that will apply the annotation to a selected text range.

element-wrap or **element-unwrap**
Specifies the keystroke combination that will trigger the specified wrap/unwrap operation.

When used in **macro**, **annotation**, **element-wrap** and **element-unwrap** elements, 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 suitable for macros:

```
alt 8
shift alt PAGE_DOWN
```

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

When used in **story-element** elements, only a single alphabetic character in the range A-Z (or a-z) may be specified. Anything else will be ignored. The semantic shortcuts are case-insensitive, so it does not matter whether you specify H or h.

Syntax

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

8.35 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 **lang** attribute must be used to specify the language of each label.

Syntax

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

Attributes

`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.

8.36 list-style-field

When it appears as the child of a **`field`** element in a list story element definition, it indicates that the parent **`field`** element holds the list's list style name.

(A list story element is a **`story-element`** element containing a **`ui:element-style`** element that is set to the value **`list`**.)

Syntax

```
<list-style-field/>
```

8.37 macro

Defines a macro to be added to the toolbar 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 entry to be added to the menu of the macro button to in the field's toolbar. The **`step`** sub-element defines the action to be performed by the macro and the **`description`** sub-element defines the menu entry's label. An optional **`keystroke`** sub-element allows a keyboard shortcut to be associated with the macro, and an optional **`icon`** sub-element defines an icon to be displayed next to the entry label.

Syntax

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

  <description>...</description>

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

Attributes

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

8.38 maxchars

The maximum number of characters allowed in a storyline, story element or field.

Syntax

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

8.39 maxwords

The maximum number of words allowed in a storyline, story element or field.

Syntax

```
<maxwords>
  integer
</maxwords>
```

8.40 minchars

The minimum number of characters required in a storyline, story element or field.

Syntax

```
<minchars>
  integer
</minchars>
```

8.41 minwords

The minimum number of words required in a storyline, story element or field.

Syntax

```
<minwords>
  integer
</minwords>
```

8.42 opensearch

Defines a way of providing third party open search provider.

Syntax

```
<opensearch
    name="..."
    src="..."
    mime-type="..."?
  />
```

Attributes

`name="...`
The name of this search. It must match the name of the search plug-in component created to handle external searches of this type.

`src="...`
A internal name used to identify an external OpenSearch service. It is mapped to the URL of the external service's OpenSearch descriptor by the **NamedServiceResolver** component.

`mime-type="...` (optional)
A MIME type identifier that determines which search URI template to choose from the list of templates specified in the external service's OpenSearch descriptor. The default is **application/atom+xml**.

8.43 panel

A panel in which the field occurs. A field can occur in one or more panels. A panel can contain another panel. Currently, there can be only two levels of panels, future versions of ECE may support more.

Syntax

```
<panel
    name="text"
  >
  text<panel>...</panel>*
  <label>...</label>?
  <description>...</description>?
</panel>
```

Attributes

`name="`*text*`"`
The name of the panel.

8.44 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**.

8.45 presentation

This element is to hold presentation related information in the same way as what CSS is to HTML, separating presentation from content.

Currently, this element is only relevant in the context a **vdf:model** document.

Syntax

```
<presentation>
  <panel>...</panel>*
</presentation>
```

8.46 preview

When it appears as the child of a **field** element, this element specifies that the field can be included in content item previews. CUE content item previews are pop-up dialogs that offer users a quick way to view the content of stories listed in a source monitor. Only text, rich text and image fields are actually previewable in CUE, so adding **preview** to other types of **field** will have no effect.

Syntax

```
<preview/>
```

8.47 priority

When it appears as the child of a **story-element-type** element in a story element type resource, this element specifies the priority of the story element type. It is used by CUE to determine the order in which story element types appear in the storyline editor insert menu. Elements with high priority values are placed before elements with low priority values.

Syntax

```
<priority>
  integer
```

```
</priority>
```

8.48 read-only

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

Syntax

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

8.49 ref-content-type

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

Syntax

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

8.50 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 CUE. 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**.

8.51 search-filter-name

When it appears as the child of a **relation-type** element in a **content-type** resource or as the child of a **link** type **field** element in a **story-element-type** resource, this element specifies

the name of a custom search filter that is to be used when displaying the CUE asset picker dialog associated with the relation or link. The custom search filter will replace the standard search filter panel otherwise displayed by CUE asset picker dialogs.

Syntax

```
<search-filter-name>
  text
</search-filter-name>
```

8.52 separate-elements

When it appears as the child of a **`storyline-template`** element and contains the value **`true`** this element specifies that the top-level story elements in storylines of this type are to be displayed as separate fields rather than as a continuous document.

Including a **`separate-elements`** element in a storyline template only has this effect if all of the template's top-level story element types are **complex elements** (that is, story elements that contain other story elements).

The **`print.xml`** storyline template supplied with the Content illustrates the use of this element.

Syntax

```
<separate-elements>
  (true|false)
</separate-elements>
```

8.53 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.

8.54 story-size

When it appears as the child of a `field` element, this element specifies that the field is to be used to select storyline length constraints. Irrespective of where the field is included in a content type definition, it will always be displayed on the **Metrics** metadata panel in CUE. Irrespective of the field's type, it will always be displayed as a drop-down selector in CUE.

Syntax

```
<story-size/>
```

8.55 style

Contains a CSS style definition that can be applied to this element's parent `field` element by application user interfaces. It is mostly used to style the content of rich text fields (`basic` fields where `mime-type` is set to `application/xhtml+xml`) in CUE. You can however also use it in a limited way to style the content of other field containing text (plain-text basic fields, number fields, URI fields and so on). If used with fields other than rich text fields, then you must observe the following limitations on your CSS code:

- Only use the CSS selector `body`

  Only use the CSS properties that are used are `font-size` and `font-weight`

  Only use the `font-size` units pixel (`px`) or percentage (`%`)

  Only use the `font-weight` values `normal` and `bold`

The `style` field is not used by any application other than CUE, and has no effect if specified as the child of other kinds of element.

Syntax

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

Examples

- This example increases the size and weight of the text in a title field.

```
<field mime-type="text/plain" type="basic" name="title">
  <ui:style>
      body {
      font-size: 20px;
      font-weight: bold;
      }
  </ui:style>
</field>
```

- This example adds a nice CSS table format to a rich text field.

```
<field mime-type="application/xhtml+xml" type="basic" name="body">
  <ui:label>Body</ui:label>
  <ui:layout rows="10" cols="40"/>
  <ui:style>
      body {
          font-size: 20px;
          font-family: Georgia, Times-Roman, serif;
          width: 100%;
          border-collapse: collapse;
      }
      table {
          font-family: Arial, Helvetica, sans-serif;
      }
      td {
          text-align: left;
          border: 1px solid #98bf21;
      }
      th {
          font-family: "Arial Black";
          font-size: 24px;
          text-align: left;
          padding: 10px 10px 10px 10px;
          background-color: #A7C942;
          color: #ffffff;
      }
      tr:nth-child(even) td {
          color: #000000;
          background-color: #EAF2D3;
      }
  </ui:style>
</field>
```

- 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>
```

8.56 summary

Nominates a story element type as a potential summary field. A **summary** element may only appear as the child of a **story-element-type** element. If present, it means that story elements of this type **may** be used as a content card summary in CUE: the first story element of this type to appear in a story will be used as the story's summary.

Syntax

```
<summary/>
```

8.57 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 CUE can use the presence of **`tag-scheme`** elements to determine which tags they make available to users editing content items. CUE does not display a **Tags** tab for content types that have no **`tag-scheme`** elements.

Syntax

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

8.58 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`**.

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>
```

8.59 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 CUE, 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>
```

8.60 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>
```

8.61 visibility

If present, this element governs the visibility of its parent in application user interfaces. It only has meaning as the child of an element that represents a user interface component (a **`field`** element in a **`content-type`** resource, for example).

In CUE, fields are hidden if they contain a **`visibility`** element that is set to **`hidden`** or **`expert`**.

Allowed values are:

`hidden`
: The parent element's user interface component should be hidden in all applications.

`expert`
: The parent element's user interface component is not typically used on a day-to-day basis, and should be hidden in all applications, unless it is already in use. A field, for example, should be hidden unless it already contains data. Applications may offer users an option that shows these expert components.

`advanced`
: The parent element's user interface component should be visible in applications aimed at advanced users. Applications that hide advanced components may offer users an option that shows them.

  This is the default setting used when **`visibility`** is not specified.

`beginner`
: The parent element's user interface component should be visible in all applications.

Syntax

```
<visibility>
  (hidden|expert|advanced|beginner)
</visibility>
```

8.62 whitelisted-elements-onpaste

When it appears as the child of a rich text **`field`** element in a **`content-type`** resource, this element specifies a whitelist of HTML formats that will **not** be deleted when pasting formatted text into the field in CUE.

The list must be space-separated, and each format must be specified as an HTML element name, optionally followed by an attribute name enclosed in square brackets. An attribute name may optionally be followed by an **=** sign and a value. For example:

```
h1 h2 h3 b i p br a[href] a[target] a[rel] a[class=myclass]
```

If a whitelist is specified in this way, then CUE will strip any formats that do not match the patterns in the list from content pasted into the field. If no whitelist is specified, then CUE still performs format stripping, using its built-in default whitelist.

Syntax

```
<whitelisted-elements-onpaste>
  text
</whitelisted-elements-onpaste>
```

9 acl

The **acl** schema defines elements that may be used to specify **Access Control Lists**. It may only be used within a **/escenic/workflow/content/** shared resource. The purpose of the **acl** is to specify:

- The **role** that is allowed to access content within a particular state
- What members of the role is allowed to do with content in a particular state.

Namespace URI

The namespace URI of the **acl** schema is **http://xmlns.escenic.com/2018/acl**.

9.1 content

A container for the roles of a content item. This element may contain either one or more **role** elements.

Syntax

```
<content>
  <role>...</role>+
</content>
```

9.2 permission

Defines a permission for the parent **role** element. The valid values are

read
: Users with this role are allowed to read content items.

write
: Users with this role are allowed to write to content items and to create new content items.

Syntax

```
<permission
    time-control="(scheduled|expired|none)"?
  >
  text
</permission>
```

Attributes

time-control="(scheduled|expired|none)" (optional)
: Specifies that this permission is only granted for time-controlled content items that are in a specified time control sub-state. It only makes sense to use this attribute in combination with the **published** state, since time control is not relevant for other states.

  Allowed values are:

scheduled
The specified permission is only granted for content items that are scheduled for publishing but not yet actually published.

expired
The specified permission is only granted for content items that were published but have now expired.

none
No effect: specifying this value is the same as omitting the **time-control** attribute.

9.3 private

Defines that a state is considered to be private. Content items in a private state will only be accessible to the authors of the content item.

Private states are usually defined as the initial state of a content item. Adding a private state to a workflow will give the journalists the possibility to work on a story without sharing it with colleagues before he feels the story is ready to be shared. It will also gives multiple journalists the possibility to share and cooperate on a single story without sharing it with everyone in their organization.

Syntax

```
<private>
  text
</private>
```

9.4 role

Defines a role.

Syntax

```
<role
    name="..."
  >
  <private>...</private>*<permission>...</permission>+
</role>
```

Attributes

name="..."
The name of the role. Valid values are

reader

journalist

editor

10 search-filter

The **search-filter** schema defines the content of a search filter definition.

The purpose of a **search-filter** resource is to define the content and functionality of a CUE search filter.

Namespace URI

The namespace URI of the **search-filter** schema is **http://xmlns.escenic.com/2018/search-filter**.

Root Element

The root of a **search-filter** file must be a **search-filter** element.

10.1 default

Syntax

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

10.2 default-term

A default term to be included in all searches. The term must be specified in pure Lucene syntax with no placeholders and no Escenic specific parts (like meta- and field-).

The **default-term** element is intended for use in search filters that will be used by source monitors. It allows you "specialize" the results returned by the search filter. Specifying a **default-term** value of **+state:draft** will ensure that only draft content items are found, no matter what other filters the user sets in CUE.

You should not include a **default-term** element when editing the **main** search filter definition used by the default search page, otherwise you will limit what users can use it to search for.

Syntax

```
<default-term>
  text
</default-term>
```

10.3 disable-container

Prevents storyline search results from being per container: a separate search results is listed for each storyline in the container that matches the search criteria. When inserted as the child of the

root **search-filter** element, container aggregation is completely disabled for all the filters in the definition. When inserted as the child of a **filter** element:

- It only affects the the results of that specific **filter**
- It only has any effect if the **filter** in question also has a **facet** field defined.
- If a **value** attribute is specified, it may only affect some results.

Syntax

```
<disable-container
    value="NCName"
  />
```

Attributes

value="*NCName*"
Specifies a filter value (that is, a facet field value) that will disable the container aggregation of search results. A **filter** element may contain multiple **disable-container** elements, each with different **value** attributes. The **value** attribute is ignored if the **disable-container** element the child of a **search-filter** element.

10.4 facet

Defines a search facet. Note that to make faceting work you need to add a configuration property in one of the Content Store's configuration layers as well as adding this element to your filter definition. For details, see Indexing Fields.

> For date facets, you can specify a default option in the usual way, by including a **ui:value-if-unset** element as a child of this element. This only works, however, for date facets. It will be ignored for other facets.

Syntax

```
<facet
    field="NCName"
  >
  <option>...</option>+<query-template>...</query-template>
</facet>
```

Attributes

field="*NCName*"
The name of the Solr field associated with this search facet. It must be the name of a field specified in the Content Engine's Solr schema.

10.5 filter

Defines one field or control in a CUE search filter panel plus the corresponding Content Store search functionality.

In the simplest case, a filter element can just contain a **facet** element. This works fine for filters based on fields that can contain a limited number of known values (content type and state fields, for example).

You can create binary filter elements that appear in CUE as checkboxes with filter elements that contain a single **term** element like this:

```
<filter name="state-deleted">
  <ui:label>Include Deleted Contents</ui:label>
  <term>meta-state:deleted</term>
  <ui:editor-style>custom</ui:editor-style>
</filter>
```

Where a filter is based on a field that can contain a much larger range of values and you want to offer type-ahead assistance in CUE, you will need to include one or more child **term** elements plus a **ui:auto-complete** and a **ui:editor-style** element. The **term** elements in this case are used to construct the search needed to fill the type-ahead drop-down menu.

The optional **lucene** element gives you complete freedom to write your own filter query using Lucene query syntax.

Specifying the optional **ui:value-if-unset** element gives you the possibility to define a default value for the filter. This makes it possible to for instance specify a default date interval for the filter.

Syntax

```
<filter
    name="NCName"
  >
  <facet>...</facet>?<disable-container/>?<term>...</term>*<lucene>...</lucene>?ANY-
FOREIGN-ELEMENT*
</filter>
```

Attributes

name="*NCName*"
The name of the **filter** element. Note that no two filters belonging to the same **search-filter** element may have the same name. If the **filter** element contains no **ui:label** element, then the name will be used as a default field label in CUE.

10.6 lucene

You can use this element to specify Lucene syntax. Whatever you enter is used to create additional URI parameters that are passed on to Solr.

You can enter a series of one or more *key=value* expressions, each on a separate line. Thus, this field cannot be used to add terms to Solr.

To add a filter query that provides CUE with a facet count for the content type called **default**, for example, you could specify:

```
<filter name="by-ct-default">
  <lucene>
    facet.query=contenttype:default
```

```
  </lucene>
</filter>
```

Syntax

```
<lucene>
  text
</lucene>
```

10.7 option

Defines an option to be presented to the CUE user as a possible facet value. The content of this element is displayed as the UI label of the option. Its value is specified in the **value** attribute.

Syntax

```
<option
    value="NCName"
  >
  text
</option>
```

Attributes

value="*NCName*"
The value of a facet option. This value is use to replace any **%1$s** placeholders in the facet's query template.

10.8 query-template

An Open Search query template. The specified value must be a valid Open Search query with **%1$s** placeholders identifying where a user-selected option is to be inserted.

Syntax

```
<query-template>
  text
</query-template>
```

10.9 search-filter

The root element of a CUE search filter definition file. A CUE search filter is a collection of user interface controls that can be used to filter the result set returned by a search. A **search-filter** element contains a series of **filter** elements defining the controls/fields in the filter. It also contains a series of **sort-by** elements, each defining a sort option to appear in the search results user interface and an optional **default-term** element specifying a default term to be included in the definition of all filters.

Syntax

```
<search-filter
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT*<filter>...</filter>+<sort-by>...</sort-by>+<default-term>...</
default-term>?<disable-container/>?ANY-FOREIGN-ELEMENT*
</search-filter>
```

Attributes

name="*NCName*"
The name of the search filter definition.

10.10 sort-by

Defines one search result sorting option to be made available to CUE users. Note that in order for the option to appear correctly in the CUE user interface, it must have a child **ui:label** element (for a tooltip) and **ui:icon** element (for a button).

Syntax

```
<sort-by
    name="NCName"
  >
  <term>...</term>+<default>...</default>?ANY-FOREIGN-ELEMENT*
</sort-by>
```

Attributes

name="*NCName*"
The name of the sorting option.

10.11 term

This element can be used in two different ways:

- **In a `filter` element** it is used to hold a series of one or more Lucene terms used to construct the filter. For example:

```
+title_ngram:"{{term}}" +meta-contenttype:com.escenic.section +meta-orderby:"score
 desc"
```

  The Lucene terms can contain the following Mustache-style placeholders:

  `{{term}}`
  This placeholder is replaced by the value entered by the CUE user in this filter field.

  `{{publication}}`
  This placeholder is replaced by the name of the current publication.

  A **`term`** element can also contain the following CUE **`meta-terms`** along with standard Lucene terms:

  `+meta-`*`field`*`:`*`value`*
  This kind of meta-term is used to generate a Solr term query using the specified field and value. **`+meta-contenttype:image`**, for example, is converted to **`q=contenttype:image`**.

  `+meta-orderby:"`*`field order`*`"`
  This kind of meta-term is used to generate a sort specification that is added to the Solr sort parameter: **`+meta-orerby:"creationdate desc"`**, for example, will add **`creationdate+desc`** to the submitted sort parameter

  If a **`filter`** element contains multiple **`term`** elements, they are simply concatenated to form a more complex filter. In other words, you can include all your filter terms in a single **`term`** element or split them into several **`term`** elements as you choose.

- **In a `sort-by` element** it is used to hold a sorting specification, consisting of a **sorting method definition** followed by a **sort order keyword**. The sorting method definition can either be:
  - One of the following Solr fields:

    `alphabetical`
    `creationdate`
    `activepublishdate`
    `publishdate`
    `firstpublishdate`
    `lastmodifieddate`

score

or the name of any other indexed field – that is, any basic field that is defined in a **content-type** resource with the attribute **search="index-store"** . For further information about this, see Basic field.

Please note that **activepublishdate** is the **activatedate** field if set, otherwise the **publishdate** field.

- A Solr expression that returns a sortable sequence of values. This means you can, for example, make use of Solr function queries.

The sort order keyword must be either **ascending** or **descending**.

The following example sorts by creation date, with the most recent first:

```
creationdate descending
```

This example makes user of the Solr function query **def()** to sort by **activepublishdate**, using the **lastmodifieddate** field as a default or fallback where the **activepublishdate** field is not available:

```
def(activepublishdate, lastmodifieddate) descending
```

Syntax

```
<term>
  text
</term>
```

11 dashboard

The **dashboard** schema defines the content of a dashboard resource.

The purpose of a dashboard resource is to define a dashboard (a panel containing predefined searches) for display in CUE.

Namespace URI

The namespace URI of the **dashboard** schema is **http://xmlns.escenic.com/2020/dashboard**.

Root Element

The root of a **dashboard** file must be a **dashboard** element.

11.1 dashboard

Defines a CUE dashboard.

Syntax

```
<dashboard
    name="NCName"
  >
  <required-capability>...</required-capability>?
  <update-interval>...</update-interval>?
  <ref-search-filter>...</ref-search-filter>+
  ANY-FOREIGN-ELEMENT*
</dashboard>
```

Attributes

name="*NCName*"
The name of the dashboard. This name forms part of the dashboard URI.

11.2 ref-search-filter

A predefined search, the results of which are displayed in the dashboard.

Syntax

```
<ref-search-filter
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT**
</ref-search-filter>
```

Attributes

name="*NCName*"
The name of the predefined search filter to use. Note that the name of a search filter is the name specified inside the search filter resource file, not the name given to it when it is uploaded to the Content Store (although they may well be identical).

11.3 required-capability

The name of a CUE capability. This dashboard will only be made available to CUE users for whom this capability is enabled. If no **required-capability** element is present, then the dashboard will be made available to all users.

Syntax

```
<required-capability>
  NCName
</required-capability>
```

11.4 update-interval

The dashboard's update interval, specified in seconds. It determines how frequently the dashboard's search results are updated. If omitted, CUE's default update interval of 30 seconds is used.

Syntax

```
<update-interval>
  integer
</update-interval>
```

12 capability

The **capability** schema defines elements that may be used to specify **capabilities**. It may only be used within a **/escenic/capability/** shared resource. The purpose of a **capability** resource is to specify CUE editor capabilities that may be enabled or disabled for specific users or user groups.

Namespace URI

The namespace URI of the **capability** schema is **http://xmlns.escenic.com/2020/capability**.

Root Element

The root of a **capability** file must be a **capabilities** element.

12.1 capabilities

The root element of a capability resource.

Syntax

```
<capabilities>
  <capability>...</capability>*
  <capability-group>...</capability-group>*
</capabilities>
```

12.2 capability

Defines a capability to be displayed in the capabilities section of Web Studio.

Syntax

```
<capability
    name="NCName"
    ref-capability-group="NCName"
  >
  ANY-FOREIGN-ELEMENT*
</capability>
```

Attributes

name="*NCName*"
The name of the capability. This name must match a capability name defined in a CUE component/enrichment service configuration.

ref-capability-group="*NCName*"
The name of the capability group to which the capability belongs.

12.3 capability-group

Syntax

```
<capability-group
    name="NCName"
  >
  ANY-FOREIGN-ELEMENT*<capability-group>...</capability-group>*
</capability-group>
```

Attributes

name="*NCName*"
The name of the capability group.

13 role

The **role** schema defines the content of a CUE role resource. The purpose of a role resource is to define a role (such as journalist, reader or editor), and to specify the global access rights or **permissions** to be granted to CUE users with that role.

In addition to these global permissions, roles are also assigned "dynamic" permissions to content items in **workflow definitions**. The permissions granted in workflow definitions are not constant: they depend on the state of a content item.

Namespace URI

The namespace URI of the **role** schema is **http://xmlns.escenic.com/2020/role**.

Root Element

The root of a **role** file must be a **role** element.

13.1 action

Specifies an action allowed by a permission. The following actions may be specified:

```
    access_own_content_only
add_group
add_pool
add_to_inbox
add_to_list
add_user
create
create_tag
delete
delete_tag
preview
publish
read
remove_from_inbox
remove_from_list
remove_group
remove_pool
remove_user
update_tag
write
write_state
```

Not all actions may be used in all contexts.

Syntax


```
<action>
  text
</action>
```

13.2 parameter

A parameter for the permission. Permissions do not usually require any parameters to be specified.

Syntax

```
<parameter>
  text
</parameter>
```

13.3 permission

Defines a permission or data access right.

Syntax

```
<permission
    ( class="..."
 type="(publication|article|section|pool|user|usergroup|person|profile|contenttype|tagstructure)" |
 type="(publication|article|section|pool|user|usergroup|person|profile|contenttype|tagstructure)" )
  >
  <parameter>...</parameter>?
  <action>...</action>*
</permission>
```

Attributes

`class="..."`

`type="(publication|article|section|pool|user|usergroup|person|profile|contenttype|tagstructure)"` (optional)

The permission type. It determines what kind of object the permission applies to.

Allowed values are:

`publication`
The permission applies to publications.

`article`
The permission applies to content items. Note that this permission type is not used in most roles, since access to content items is mainly controlled by workflow definitions. It is, however, used in special cases where a global restriction needs to be added to the access rights granted by workflow definitions. A freelancer role definition, for example, would typically be assigned an article permission containing the **`action access_own_content_only`**. This restricts the freelancer's rights so that he can only access his own content items.

`section`
The permission applies to sections.

pool
The permission applies to lists, inboxes and section pages.

user
The permission applies to users.

usergroup
The permission applies to user groups.

person
The permission applies to persons.

profile
The permission applies to user profiles.

contenttype
The permission applies to content types.

tagstructure
The permission applies to tag structures.

type="(publication|article|section|pool|user|usergroup|person|profile|contenttype|tagstructure)"
The permission type. It determines what kind of object the permission applies to.

Allowed values are:

publication
The permission applies to publications.

article
The permission applies to content items. Note that this permission type is not used in most roles, since access to content items is mainly controlled by workflow definitions. It is, however, used in special cases where a global restriction needs to be added to the access rights granted by workflow definitions. A freelancer role definition, for example, would typically be assigned an article permission containing the **action access_own_content_only**. This restricts the freelancer's rights so that he can only access his own content items.

section
The permission applies to sections.

pool
The permission applies to lists, inboxes and section pages.

user
The permission applies to users.

usergroup
The permission applies to user groups.

person
The permission applies to persons.

profile
The permission applies to user profiles.

contenttype
The permission applies to content types.

tagstructure
The permission applies to tag structures.

13.4 role

Defines a particular role (such as journalist, reader or editor).

Syntax

```
<role
    name="NCName"
    type="(publication|section|contenttype|tagstructure)"?
  >
  <permission>...</permission>*
  ANY-FOREIGN-ELEMENT*
</role>
```

Attributes

`name="`*NCName*`"`
The name of the role.

`type="(publication|section|contenttype|tagstructure)" (optional)`
The type of the role. It determines to which level of the CUE publication structure the role's permissions apply.

Allowed values are:

`publication`
The role's permissions apply to publications.

`section`
The role's permissions apply to publication sections. This is the most commonly-used type setting for custom roles.

`contenttype`
The role's permissions apply to content items.

`tagstructure`
The role's permissions apply to tag structures.

14 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 Store'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.

14.1 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
```

14.2 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.

14.3 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.

14.4 article.presentation.content.inlineInternalLink.removeLinkText

This property determines how an inline relation is presented in publications if the related content item is not published. If **`article.presentation.content.inlineInternalLink.removeLinkText`** is set to **`true`** (the default value), then the relation link does not appear in published output at all. If **`article.presentation.content.inlineInternalLink.removeLinkText`** is set to **`false`**, then the link **text** appears in the published output, but it is displayed as ordinary text, not as a link.

14.5 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 Store. The limit above which fields will be compressed is defined by the **`article.presentation.gzip.threshold`** property (see section 14.6).

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

Example:

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

14.6 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 14.5) 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
```

14.7 autoPublishStorylines

This property allows you to configure a publication's storyline auto-publishing feature. It overrides any **`auto-publish-storylines`** setting made in a **`publication-type`** resource. The property may be set to one of the following three values:

`disabled`
: Storyline auto-publishing is disabled.

`full`
: Storyline auto-publishing is **fully** enabled in publications of this type. "Fully enabled" means that all published destination content items will be automatically republished if any upstream

content item is published. If a destination content item is in the state **draft-published** (or any similar staged state) then it will only be republished if all of its storyline changes are inherited. If the storyline has been directly modified by a user since the last time it was published, then it will not be republished. This limitation prevents unintentional publishing of local changes. Other kinds of local change, such as changes to a content item's relations or tagging changes will not block autopublishing.

cautious
Storyline auto-publishing is **partly** enabled in publications of this type. The auto-publishing feature's behavior is almost the same as for the **full** setting, but with the following difference: a destination content item will never be automatically republished if it has ever been modified since it was first published. In other words, this is a more cautious setting, offering more robust protection against unintentional publishing of unreviewed content.

14.8 bootstrapOnStartup

The Content Store 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 Store 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 Bootstrapping.
- 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.

14.9 catalog.orderBy

If **`catalog.orderBy`** is set to **`date`**, then objects in CUE 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 CUE catalogs are sorted by name in ascending alphabetic order.

Example:

```
catalog.orderBy=date
```

14.10 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
```

14.11 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
```

14.12 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
```

14.13 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
```

14.14 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.

14.15 com.escenic.image.quality

Specifies the quality setting to be used when converting images to PNG (the format used internally by the Content Store). 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
```

14.16 com.escenic.cue.spellcheck.language

Specifies the publication's content language, as an IETF BCP47 language tag. The specified language is used by CUE for spelling- and grammar-checking purposes.

Example:

```
com.escenic.cue.spellcheck.language=de-DE
```

14.17 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/
```

14.18 multimedia.archive.orderBy

This property is a deprecated synonym for **`catalog.orderBy`** (see section 14.9). If both **`catalog.orderBy`** and **`multimedia.archive.orderBy`** are specified, then **`catalog.orderBy`** is used.

14.19 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
```

14.20 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
```

14.21 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`**.

14.22 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
```

14.23 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
```

14.24 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/
```

14.25 publication.previewUsesSectionUrl

If **`publication.previewUsesSectionUrl`** is set to **`true`**, the Content Store 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
```

14.26 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.

14.27 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.