# Templating - Proposal

Your Rating: ☆☆☆☆☆ Results: ★★★☆☆ 81 rates

**In progress for 4.5**

Introduces new templating in preparation for Magnolia 5.0. Implementation tracked in SCRUM@jira.

## Goals

- improve the templating to make it more intuitive
- prepare the new page editing
- align JSP and FreeMarker templating
- provide maximum possible backward compatibility

## Summary

- rename paragraphs to (page) components
- introduce areas
    - sub elements of a template
    - container for components
- provide a new minimal set of directives
    - area, render, edit
    - better attribute names, aligned to the JCR API
- provide a set of standard functions to allow more complex operations
- streamline content expressions
    - implementation is based on the Map interface
    - allow using the JCR API: cmsfn.asNode(content).getProperty("bal")
- align component, area and page template definitions
    - use the same renderer
    - use the same definition beans

## Paragraphs become (page) components

- for new Magnolia users and developers it was always difficult to grasp the term 'paragraph' as used in Magnolia
- CMS like Magnolia are known as component based system

## Align page and component templates

Definitions

- use a single registry
    - to avoid checks like in the rendering engine
- use a common template definition object
- drop the differentiation of components and page templates
    - structure the templates with folders
    - components/content/textImage, pages/article
    - the path is used to reference a component/dialog

Renderers

- one renderer interface and a single renderer registry
- same set of context objects
- JSP: decide based on situation if we forward or include (forward if main content is the same as the current)

## Template Definitions (Enhancements)

- additional context objects can be configured per template
- dialog can be configured in the template definition rather than being referenced

## Template Variations

Template variations are similar to the former sub-templates.

- all templates can have variations (including areas and paragraphs)
- configured as a Map

Default variations

- json: renders a json representation of the content
- xml: renders a xml representation of the content
    - formated similar to the JCR document view

Selection mechanism

- the variation is set in the aggregation state
- by default this is the extention
- other example: variations per device

Inheritance (from the base template)

- the template is merged with the 'base' template
- works the same way as in STK: current template and site's prototype

## Areas

- templates have areas (regions having components)
- configured as a map in the template definition
- list available components (like in STK)
- defines the template script to be use
- enabled flag
- can have its own edit dialog

### Types

| Type | Description | Structure | Context |
|---|---|---|---|
| list | • collection of components<br>• creates an area node which contains the components nodes<br>• minimal and maximal number of components can be set<br>• the area bar has an "add component" button | `page (mgnl:page)`<br>`  - main (mgnl:area)`<br>`    - component[0]`<br>`    - component[1]`<br>`    - component[2]` | |
| • content: area node<br>• components: list of components | | | |
| single | • an area which holds one and only one single paragraph<br>• creates one single node which is the paragraph's node<br>  ◦ no container node like for lists<br>• optional single areas<br>  ◦ allows the explicit creation and deletion of the paragraph<br>• the area bar is the edit bar<br>  ◦ the edit bar is not going to be rendered | `page (mgnl:page)`<br>`  - stage (mgnl:area)`<br>`    - title`<br>`    - component (mgnl:`<br>`component)`<br>`      - title` | |
| • content: area node<br>• component: the component node | | | |
| no-component | • areas where no component can be added<br>• examples: navigation, bread crumbs, .. | `page (mgnl:page)`<br>`  - navigation (mgnl:area)`<br>`    - levels` | |
| • content: area node | | | |

Template script

- can be inlined (in the area tag)
- content object is the area's node

Non existing areas

- content is null if the area is not existent
- the script can render a placeholder

Model

- an area has a model like all templates do

Available components

- list of roles, groups and users
- enabled flag
- bean: ordered map

Sub Areas

- areas can have sub areas

Auto generations

- both single and list types can be auto generated
    - on creation of the page
- if a generated component has a *required* property it is not deletable
- if a generated component has its *orderable* property set to *false* it can't be moved

Inheritance

- can be configured
- see chapter 'inheritance'

## Tags/Directives

- minimal set of directives
- use jcr naming: node, property, path

### Common attributes for passing the content to use

| Attribute | Description | Default Value |
|---|---|---|
| content | a Node or ContentMap | |
| workspace | the workspace used if the uuid or path is defind | same as of the current content |
| uuid | | |
| path | the path in the workspace | |

### cms:area

- type list: replacement of the iterator tag and new bar
- type single: combines the new and edit bar

| Attribute | Description | Default Value |
|---|---|---|
| name | the area name, will work on def.areas[MAGNOLIA5:name ] | |
| area | an area definition object | |

| | | |
|---|---|---|
| components | list of available components | by area definition |
| script | | by area definition |
| placeholderScript | | null, by area definition |
| type | list or single | list, by area definition |
| dialog | | null, by area definition |
| inherit | | false |

**Context**

In addition to the 'normal' context objects the followings objects are provided

- components: list of the area's components, includes inherited components and respects the order information
- component: in case of a single area, might be inherited

**Inline Areas**

- Status: Proposal, not implemented yet, see MAGNOLIA-4586
- Allow definitions directly in the template script (without definition)

```
# every thing is configured at def.areas[name]
[@cms.area name="main"/]

# inline
[@cms.area name="main" components="paragraphs/content/textImage, paragraphs/content/linkList"]
   <div id="main">
   [#list components as component]
       [@cms.render content=component ]
   [/#list]
   </div>
[/@cms.area]

# inline single
[@cms.area name="stage" components="paragraphs/stages/stageA, paragraphs/stages/stageB" type="single"]
   <div id="stage">
       [@cms.render content=component ]
   </div>
[/@cms.area]
```

## cms:component

- Renders a component
- Similar to the former includeTemplate
- Needs an existing node

| Attribute | Description | Default Value |
|---|---|---|
| editable | if any editing elements should be shown. mainly useful if content is inherited | cmsfn.isFromCurrentPage() |
| template | name of the template definition to use | the template defined on the node |

## cms:edit

- Renders an edit bar (or button)
- Requires a node (is not a new bar)
- Can edit any node (for instance data module content)

| Attribute | Description | Default Value |
|---|---|---|
| dialog | name of the edit dialog | |

|  |  |  |
|---|---|---|
| format | button or bar | bar |

```
[@cms.edit content=contact dialog="data/contact"/]
```

In practice, you rarely need to add the `cms:edit` to a script. It is injected into editable areas and components automatically.

### cms:context

- former attribute tag used for cms:render
- ctx.name
- attributes are removed after the rendering
    - former values are restored

```
[#assign counter = counter +1]

[@cms.render] content=child]
    [@cms.context name="counter" value=counter/]
[/@cms.render]


# in the paragraph script
${ctx.counter}
```

## Functions

- Freemarker: cms context object provides functions
- JSP: static class delegating to the context object
    - standard prefix cmsfn

| Function | Description |
|---|---|
| asJCRNode(content) | to allow calls to the jcr API |
| asContent(node) |  |
| parent(content, [MAGNOLIA5:type]) | if the type is passed the first matching ancestor is returned |
| children(content, [MAGNOLIA5:type]) |  |
| uuid(content) |  |
| wrap(content) | add wrappers used in the renderer |
| link(content) |  |
| edit(content, propertyName) | adds editor markup attributes |
| decode(text) | decode content if the text is plain HTML code |
| .... |  |

Utils

- other function libraries can be added
    - like cmsutil.*
- cmsfn.* should just provide the essential functions

## Content expressions

- align freemarker and JSP
- use a Map instead of content (Magnolia)

- drop ContentModel -> allow method calls on JCR nodes
    - but still support TemplateNodeModel for ?children, ?parent, ...

**ContentMap**

- extends Map interface
    - does not implement Content or Node
    - has a asJCRNode() method
- all values are encoded
- all links are processed
- resolution
    - property
    - child node
- content.@uuid == cmsfn.uuid(content)
- content?children == cmsfn.children(content)

**Binaries**

- are now nodes (nt:resource)
- content.image --> cmsfn.link(content.image) to create a link
- content.image.size returns the size property

**Namespaces**

- content["mgnl:template"]
- content.template --> if no property "template" exists it checks if an other (with namespace) matches

## Compatibility

- a separate compatibility package is needed (to keep new code clean)

Template and 'paragraph' definitions

- definition has to be migrated (converter will be provided)
- type is: jsp4x, ftl4x

Scripts (if of type jsp4x or ftl4x)

- old directives/tags are provided
- content (of type Content) is passed
- a single template script cannot mix old and new style (also true for includes)

Editor

- render the edit bars as before (where the cms:edit tag is)
- new bar -> render them as before (no autocreation of areas)
- no area bars

## Inline (Magnolia 5)

- annotated tags can be edited inline

```
<h2 ${cmsfn.edit(content, "title"}>${content.title}</h2>
```

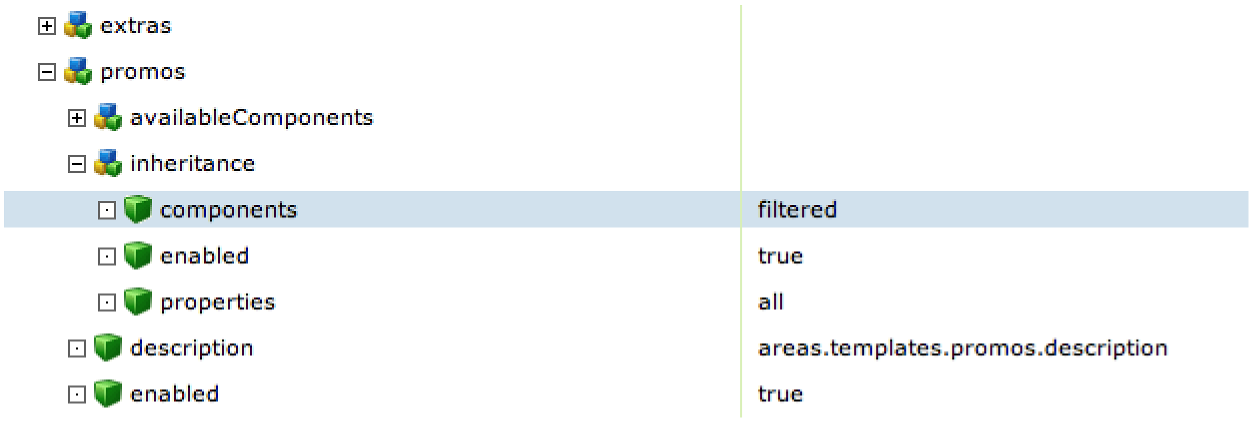
## Inheritance

Inheritance is supported by areas. In addition a set of functions and standardized content properties support custom solutions.

An area with inheritance enabled inherits properties and components from areas in its parent pages. Inheritance can be set to include only properties, only components or both.

By default properties and only components with a property 'inheritable' set to true are included.

## Configuration



```
extras
promos
    availableComponents
    inheritance
        components                 filtered
        enabled                    true
        properties                 all
    description                    areas.templates.promos.description
    enabled                        true
```

Options for components

- all
- none
- filtered, all components with a property 'inheritable' set to true (**default**)

Options for properties

- all (**default**)
- none

Component order

- a property 'nodeComparatorClass' set to a class name of a class implementing java.util.Comparator<Node>

Component filtering

- a property 'predicateClass' set to a class name of a class extending info.magnolia.jcr.predicate.AbstractPredicate<Node>

## Editing

- inherited paragraphs are not editable

## Functions

- cmsfn.inherit(content, innerPath): ContentMap
- cmsfn.inheritProperty(content, innerPath): String
- cmsfn.inheritList(content, innerPath, aggregate): Collection<ContentMap>
    - aggregate: collect over several ancestors
    - respects the orderdering information
- cmsfn.isFromCurrentPage(content)
- cmsfn.isInherited(content)

## Special properties and mixins

TODO: use mixins?

TODO: list all special properties

## References

TODO

- property mgnl:reference
- a wrapper replaces the node with the referenced node

- transparent for the template
- properties can be overriden
- very similar (or the same?) as the extends feature in configurations

## Editor markup

The editor markup is only rendered if the page is shown in author mode. The editor uses this elements to inject his components into the page.

**separators**
Added by the Area- or EditComponent to mark the boundaries

```
<!-- cms:begin cms:content="ws:path" -->
<!-- cms:end cms:content="ws:path" -->
```

**area**

```
<cms:area content="ws:path" name="name"paragraphs="paragraphs/textImage, ..." type="single" dialog="areas/main">
```

**edit bars**

```
<cms:edit content="ws:path" format="bar" dialog="paragraphs/textImage">
```

**inline**
Bind an element to a property to make it directly editable

```
<h2 ... cms:edit="inline" cms:content="ws:path@prop"">
```

## Editor

### Magnolia 4.5

TODO

### Magnolia 5

- the content is rendered in an iframe
- this isolates the editor's and page's javascripts, css ...
- no initialization code in the page