Concept taglib cleanup, extraction, rewrite

Your Rating: ☆☆☆☆☆ Results: ★★★☆☆ 77 rates

Implemented in 4.5

Simplification of the current tag library. Implementation tracked in MAGNOLIA-2993@jira.

Rationale

The current tag library has a few shortcomings. It is outdated, too complex to use and to maintain. New concepts have been introduced at higher levels to solve similar problems.

- inheritance logic, which can now be enabled by `InheritanceContentWrapper` (used with `MagnoliaTemplatingUtilities#inherit`, `STKUtil`, templating models, ...
- legacy code for support of nested paragraphs, which is not needed anymore.
- it is only a tag library, so it's not useable outside the context of JSP templates. We're "lucky" that FreeMarker provides support for taglibs; support for other templating engines would require duplicating everything that's in those tags)
- the templating module now provides a set of objects (content, mgnl(MagnoliaTemplatingUtilities), ctx, ...) to templates.
- the templating module now supports model classes, thus giving the ability to move complex logic out of the templates.

We need to provide solutions for this, and clean up the existing tag library (deprecation).

Goals

- ✓ Integration with other templating engines
- ✓ Testability
- ✓ Maintainability
- [ ] Customizability/extensibility - **still need to provide guidelines for this** - Corner cases that are currently possible to reach, by accident or not, will probably be not supported. On the other hand, it should be easy to extend/replace/add functionality such that such corner cases can provided for on a case-by-case basis.
- ✓ Hide complexity, sensible defaults - no extraneous parameters "just in case"; parameters are "typed".
- ✓ Permissions checked consistently. The permissions needed to render a component are now consistently checked in `AbstractAuthoringUiComponent#shouldRender`
- ✓ i18n. More consistency: we need use the current renderable's i18nBasename if any, or fallback to the message bundle of m-m-templating-components.
- ✓ Stronger parameter types; in the "old" taglib, most parameters could be strings. In this new incarnation, we're attempting to be a little stricter, to avoid blurry cases. For example, the FreeMarker directives do not support for comma-delimited strings, since they can natively process arrays (`foo=['a', 'b', 'c']`. The conversion is left to the component "wrappers". The components themselves are strictly typed.
- [ ] Magnolia 5 - Whatever we introduce should still make sense with Magnolia 5: the underlying implementation can be changed/swapped, but the templates should ideally be 1:1 compatible.
- [ ] Deprecate current tags - at least those we provide a direct replacement for. See list of potential replacements below. Provide examples for replacements.
- [ ] Generated documentation - We currently have 2 different formats that document the taglib; neither is great. We should stick to a single one, and improve it. (for ex: http://dev.magnolia-cms.com/ref/latest/magnolia-taglib-cms/tagreference.html has some html tags in the attributes' description which should not have been encoded)

Status

- Part of the main Magnolia project: `magnolia-module-templating-components`.
- Currently not in the webapps by default; in `add-ons` of bundles.
- Can be used, and can replace existing tags or be used in conjunction with.
- Both taglibs can co-exist. See below.
- The behavior of the current tags and magnolia-gui classes has been painstakingly analyzed and reverse-engineered. I attempted to comment as much as possible of the quirks needed in the new components classes, to avoid leaking those quirks to the public api (i.e the jsp taglib and freemarker directives). The rendering, or output, of these components is mostly irrelevant for this issue; this currently "simply" delegate to the magnolia-gui components. The important thing being that this becomes a hidden implementation detail, which we will be able to replace/update /improve later without the hassle of having to figure out each and every corner case the current taglib covers.

Introduced components

- PageEditBar (~= MainBar)
- EditBar
- NewBar
- SingletonParagraphBar (wrap a paragraph containing an edit bar; displays a new bar with a specific label if the paragraph does not exist yet)

Still to be added

The following constructs have not been introduced yet:

- include/render
- iterate
- header tags (`cms:links`)

Guidelines

The least amount of parameters to a new component, the better. Ideally, for example, the dialog name for edit-paragraph could be deduced and allowed paragraphs for new-button could be deduced from configuration (site definition, ...)
We allow overriding these when it makes, but defaults should at least be provided when possible.
Analyze existing use-cases and think of better/simpler solutions instead of copying what already exists.

Shortcomings

- JSP: we currently use NodeMapWrapper, which causes two problems:
    - it does not expose subnodes
    - nodes returned by mgnl or models or whatever are not magically wrapped, unlike what happens with FreeMarker.
    - upgrading to jsp 2.1 (or 2.2) we can do the following and hope it can solve the issue:

```
        final JspApplicationContext jspAppCtx = JspFactory.getDefaultFactory().
getJspApplicationContext(servletContext);
        jspAppCtx.addELResolver(new ELResolver());
```

- JSP: despite 2.2 supporting method calls in EL, it seems (maybe it's a bug?) that it can not differentiate between methods with the same name but different parameter types (probably related to coercion of null types)
    - use explicit method names (meh)
    - provide a wrapper around MagnoliaTemplatingUtilities which hides the "duplicate" methods and exposes only one with Object argument (s)

TO DOs

- **Call for feedback**. Poll on which tags are used and how.
- How-to for extensibility.
- Do we keep the FreeMarker directives as a shared directive, or do we let template renderers add it to the rendering context ?
    - they currently don't, but might (should?) depend on an hypothetical RenderingState, in which case it might be saner to let the renderer construct them (avoiding the dependency on a threadlocal/static)
    - we'd need some mechanism to configure objects to be added the rendering context (provider/factory)
- http://openutils.svn.sourceforge.net/viewvc/openutils/trunk/openutils-mgnlutils/src/main/java/it/openutils/mgnlutils/el/MgnlUtilsElFunctions.java?revision=1798&view=markup
  This and our `info.magnolia.cms.taglibs.CmsFunctions` class could somewhat be redundant with `MagnoliaTemplatingUtilities` and `STKUtil`. Consolidation needed ?

Existing tags to consider

Here's a list of existing for which we should either document replacement scenarios (how to do this) or provide actual replacement.
Main taglib:

- out: formatting should be doable via JSTL `<fmt:format*>` tags. How about link resolving and binaries ? Util methods in `MagnoliaTemplatingUtilities` might be in order.
- loadPage, unloadPage, set, setNode: are there still use case for these ?
- query, simplesearch and searchResultSnippet: we need to extract this and provide simple tags/methods in mgnl or model.
- user: `${ctx.user}`
- isEmpty, isExisting and their negative variants: could be dropped if the shortcomings of NodeMapWrapper are addressed ?

Util taglib:

- breadcrumb, simpleNav: could be merged and provided as a comp too, although it is highly template dependent (i.e. produces "public" html, so templaters want controller over it)
- img: could be improved/useful for other modules too; could be made a component? could be useful for FM too
- poweredBy: simple enough to provide; util method in mgnl or component.
- redirect: why is this even needed as a tag ? (scriplet is as easy for this kind of use-case ?) How about FM?
- xmp: `${fn:escapeXml}`
- convertNewLines: ?
- ahref: use link methods of `MagnoliaTemplatingUtilities`.
- StrToObj: does this have any use outside the context of TableTag ?
- TableTag: use displaytag if you need this 😎
- Text2Image, scaleImg

More ideas

- We could also introduce a simple "open dialog" button/bar. If we did so, is there still a usecase to enable/disable move and delete on edit bar?
- If we had a rendering state, holding the current renderable, and if said renderable exposed its dialog name, we could use it as a default for edit-paragraph.
- Templating made area-aware; Area be a 1st class citizen; perhaps currentArea and currentNode/Paragraph be added to an hypothetical TemplatingState; or currentNode be updated with area when iterating, etc.: target=always current node, no need for "container"
- If areas and STKUtil's getAllowedParagraphs became first-class citizens, we could also deduce the list of paragraphs to be added by new-bar.
- Provide some sort of "adapter" to aid migration: wrapping the current("old") taglib to throw/log messages, possibly behaving transparently on the public instance, but only warning developers on author instances
- Consider using `tag` files; maybe this would help extensibility/customizability for the jsp pendant.

Implementation details

- UI component classes are in info.magnolia.module.templatingcomponents.components; they extend AbstractAuthoringUiComponent, implement AuthoringUiComponent. When we introduce non-ui components, some renaming might be in order.
- JSP and FreeMarker wrapper are in their respective subpackages, extending their respective abstract class, which provides helper method.
- JSP tags use the JSP 2.0 SimpleTag API - removes concerns about pooling, etc.
- FreeMarker wrappers are `TemplateDirectiveModel}}s, which are like macros to templates. They're made available via {{info.magnolia.module.templatingcomponents.freemarker.Directives`, which is a Map exposed as a FreeMarker shared variable (currently named `ui`)
- Wrappers currently simply rely on static factory methods (`make`) to pass their parameters; all they have to care about is validate the parameters' types and mandatoriness; any other special treatment should happen in the factory method. Default values are also provided by the `make()` methods, except certain cases(booleans for instance)

Naming

- Templating components
- We might introduce the name "templating constructs" for things like "include", "iterator" and "header links".
- "wrapper". We use the term "wrapper" to indicate a JSP Tag, FreeMarker directive or other template-engine specific object used in templates, which delegates to an actual component.

Examples

Using the new tags or directives

See the various templates at http://svn.magnolia-cms.com/svn/community/bundle/trunk/magnolia-integration-tests/magnolia-integration-tests-module/src/main/resources/mgnl-files/templates/test/manual/

Using `mgnl`

FreeMarker

```
[#if mgnl.editMode]this only appears on author instances, in edit mode.[/#if]
```

You can also call methods:

```
${mgnl.getContent("data", "/contacts/JMustermann").officePhone}
```

JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
[...]
<c:if test="${mgnl.editMode}">this only appears on author instances, in edit mode.</c:if>
```

With JSP2.2 and EL2.2, you can also call methods:

```
${mgnl.getContent("data", "/contacts/JMustermann").officePhone}

${mgnl.createLink('website', 'abc')} // outputs nothing (jsp/el's handling of null) but error is logged
Is last: ${mgnl.siblings(content).last}
```

If you're stuck with JSP 2.1, you can still do the following and use scriptlets:

```
<% MagnoliaTemplatingUtilities mgnl = (MagnoliaTemplatingUtilities) pageContext.findAttribute("mgnl"); %>
[...]
<%=mgnl.getContent("data", "/contacts/JMustermann").officePhone%>
```

Taglib declarations

```
<%@ taglib prefix="cms" uri="http://magnolia-cms.com/taglib/templating-components" %>
<%@ taglib prefix="old" uri="cms-taglib" %>
```

OR:

```
<%@ taglib prefix="new" uri="http://magnolia-cms.com/taglib/templating-components" %>
<%@ taglib prefix="cms" uri="cms-taglib" %>
```

work equally.

Tooling support

JSP

In IntelliJ, you can add the following at the top of your JSP files: (the editor will hint you to do it anyway)

```
<%--@elvariable id="mgnl" type="info.magnolia.module.templating.MagnoliaTemplatingUtilities"--%>
```

FreeMarker

With IntelliJ, add a file called freemarker_implicit.ftl in a source root of your project (src/main/resources for example) to get autocompletion:

```
[#ftl]
[#-- @implicitly included --]
[#-- @ftlvariable name="page" type="info.magnolia.cms.core.Content" --]
[#-- @ftlvariable name="content" type="info.magnolia.cms.core.Content" --]
[#-- @ftlvariable name="ctx" type="info.magnolia.context.Context" --]
[#-- @ftlvariable name="state" type="info.magnolia.cms.core.AggregationState" --]
[#-- @ftlvariable name="def" type="info.magnolia.module.templating.RenderableDefinition" --]
[#-- @ftlvariable name="mgnl" type="info.magnolia.module.templating.MagnoliaTemplatingUtilities" --]
[#-- @ftlvariable name="model" type="..." --]
[#-- @ftlvariable name="actionResult" type="java.lang.String" --]
```

It's also something IntelliJ will hint you about, so no panic. As you can see, the one variable that can't be defined here is `model`, since it's an arbitrary class, configured per paragraph. Also note that if needed, you can override these definitions (they're only hints for the editor), for example if you need auto-completion for a paragraph that uses a specific RenderableDefinition subclass. Lastly, note that this does not provide auto-completion for content property names, so `${content.foobar}` will still be highlighted, unless you provide hints for each and every property you want to use in your template.

Eclipse provides similar functionality, via the FreeMarker plugin configuration dialog; it saves a `.freemarker-ui.xml` in your project.

MagnoliaTemplatingUtilities for JSP considerations

We essentially have three options:

- Use a jsp function taglib
  - change the methods to be static: unfavored option, as MagnoliaTemplatingUtilities is likely to soon depend on a RenderingState instance, the context, etc.
  - provide a static wrapper: would hide the hackery required to acquire said dependencies but would be some (granted, not much) extra maintenance work and is less extensible.
- Wrap it in a tag:
  - works (http://svn.magnolia-cms.com/svn/community/sandbox/mgnl-wrapper-tag/) but:
  - the method selection is based on the number of parameters, which doesn't play well with createLink(Content node) vs createLink (NodeData nd), for example.
  - when a method returns an instance of Content or NodeData, it is not wrapped, but it should (see how `${hello.text}` fails in the test jsp)
  - doesn't seem "right" for function-like calls (createLink, getContent)
- JSP2.2/EL2.2 provide for non-static calls
  - Tomcat 7 will have support for this (despite http://wiki.apache.org/myfaces/HowToEnableEl22 and http://stackoverflow.com/questions /2333605/using-el-2-2-with-tomcat-6-0-24 I can't get this to work in Tomcat 6)

    - Tested this successfully with Glassfish3, (JavaEE6), but like the tag idea, it seems the method selection is based on the number of parameters, which doesn't play well with createLink(Content node) vs createLink(NodeData nd), for example.

Notes on the "old" implementation (m-taglib-cms)

- `nodeCollectionName` is used inconsistently. In most cases, it ends up being concatenated with a node path (in some cases in javascript, for example with mgnlDeleteNode(), in other instances the final path is "calculated" by the end point (servlet), and in yet other instances, it's simply ignored.