STK remarks by the VPRO

List Model class has no constraints for 'offset' or 'max'

The class info.magnolia.module.templatingkit.paragraphs.AbstractItemListModel is the parent to all overview page model classes. It fetches some data, fiters it and sorts it, and finally creates a pager instance to return a measured amount of the data based on offset request parameter and pax page size configuration.

In our view this model scales badly. We need a way to constraint the query result. If you can translate your filtering and sorting needs into the query you can also use offset and max constraints, saving a lot of memory at least. We don't have that much experience with jcr yet, but collecting tens of thousends of content nodes for every page (of twenty items) in an overview pages just doesn't seem the right way to go about it.

Model classes

The template- and paragraph model classes that the stk provides are geared towards reuse, and indeed we find that most model classes we create extends one of these classes. Yet we find the way these classes are structured makes it hard to reuse them elegantly:

- Deep inheritance makes them often complex to understand.
- Deep inheritance makes them easy to break: not many methods are final, and because of above it is not always clear what's what. It is hard to override those classes safely.
- There is no java doc, which is awkward for classes that are geared towards reuse.
- Method granularity is sometimes not optimal: We often have to copy whole methods from a superclass and change one line.
- Deep inheritance makes them rigid. It is impossible to mix features from different model classes that don't share an ancestor, creating risk of duplicate code.

So we feel that this area could benefit greatly from a redesign, embracing the following guidelines:

- Favor composition over inheritance
- When using inheritance, use the template pattern where parts of the class that are not intended for inheritance are final, so it becomes safe to inherit them
- Document these classes to express their intended use.

This would break backwards compatibility, but perhaps this could be mitigated by providing the new api alongside the old one which would then be deprecated.

Sites, site definitions and themes

We feel this area needs some rethinking. The role of the site definition and coupling rules have changed by further enhancements on the multi site front, and it seems the components and their relationships actual use has changed (too much?) from its original intentions.

Here is an overview of the way we rank reusability (from high to low) of the various components. As you can see there is a lack of symmetry with the current existing component structure:

- Dam support config (always the same)
- Paragraph definitions
- Template definitions
- Theme (unique per site, we will create 'customizable' themes, that allow for partial customization and thus theme reuse over multiple sites)
- Mappings (unique per site)
- Site data (we may sometimes clone one site to start another)

So the current concept of sites, site definitions, themes and the way they depend on each other doesn't really work really well for us. Of cores all problems can be countered with the 'extends' mechanism, but we feel a clearer conceptualization of the way these parts actually relate could be beneficial. It should be a nice discussion.

Dam Support

I find it an excellent idea, but after working with it quite closely (I created dam support for the simple media module) I find that the system is overly complex. Getting my version of dam support to work was primarily a process of hit and mis, because, although I understood the concept of dam support perfectly, I couldn't wrap my head around the implementation, with delegation after delegation after delegation. I think simplifying the implementation serves two purposes:

- Less chance of introducing bugs. The complexity of the system makes it very bug prone, as it is probably quite hard to predict the (side) effects of changes.
- A simple implementation would stimulate adaptation and the creation of a wider range of dam handlers.

A final note: supposedly dam support can be used for any type of media, but I don't think the current implementation really supports that. What if you mix dam handlers that allow you to select audio streams with handlers that allow you to select images. It is not realistic to assume that both types could be used in the same place on a page, so you would need different dam configurations for different media, a configurable dam dialog control that makes it possible to distinguish between different dam configurations.

Image/media support

We found that the standard dam handlers available in stk are insufficient. Available are 'upload' and 'dms'. Here an overview of the minimal required feature set, annotated with the current support:

- Add images to pages on the fly ('upload')
- Reuse images ('dms')
- Thumbnail preview ('not supported')
- Search for images ('not supported')

So there is no implementation in stk that supports all these features. The Openminds Simple Media module does support them (and has some other nice features), so perhaps this module could be used as a model for an stk dam support back end, perhaps en extension to dms, or maybe the simple media module could find it's way into Magnolia (who knows). Another option is that Magnolia will make sure that the simple media works well with all features of Magnolia, (including the enterprise features).

Things you can't configure

There are two things we miss in site configurability:

- Paragraph definitions have 'parameters', which we use a lot. Template definitions don't have them, which is a shame.
- Site definitions could also benefit from the 'properties' model. As you often want to set some site-wide values.

Image variations via CSS styles

This system is overly complex and it is nearly impossible to work with it, as it is not possible to find out what kind of css style is created for a certain paragraph on a certain page. I asked about this on the user mailing list, and was advised to leave the system alone, and just use StkUtil.getAssetVariation() in stead. I would propose to get rid of it all together.

Development Workflow

Resource management

An overview of the problems we have with resources and development. In the core these problems all stem from the central point that we edit those resources on the file system, and storing them in the magnolia repository is an inconvenient extra step. Requirement: be able to add and edit resources on the fly without having to restart the server or having to edit the repository in parallel. Perhaps this dilemma could be solved by supporting two 'modes': file-centric mode and repository-centric mode. That would also create conceptual clarity.

There are three types of resources with different characteristics:

- **Theme images:** stored in the dms by default. When you add an image to your theme (in the jar) you have to manually add it to the dms or you'll get a 404. We created an install task to do that for us, but it should be in STK. But even with this we still have to add a new image to the dms if it is added during development (and you don't want to re-install your module).
  Keeping the images in the jar is no good, because you have to restart the server to get new images. Storing them on the file system is ok, if you can configure the development environment so that the images dir in the source checkout tree is observed by the application server, and the images dir are added to the web root file system. In production the images could be fetched from the jar, so you need a system that prefers the file system, and defaults to class loading.
  Still it would be nice to have a 'copy' of the images in the repository for on-the-fly fixes/changes, but for us this is a nice to have.
- **javascript and css:** Are loaded from the repository by default but this can be 'bypassed' so they are loaded from the file system. For those resources we have an install task that switches everything to 'bypass' (when in development mode), combined with a jetty configuration that adds the source folders for these resources to our web source path. But still resources added during development will have to be added to the repository (and switched to bypass) manually before loaded from the file system. For us it would be better if they would be loaded from the file system by default. That would cover both development (file system), and production (no file system, so fall back to repository).
- **Freemarker templates:** These are loaded from the file system by default, and if you want to override them in the repository you have to explicitly enable that. We have an install task that imports all

templates in the repository but they are not enabled so they are actually loaded from the classpath?? file system??? Not clear. Anyway this works pretty much ok for us.

We would very much like to have a discussion about this.

Usability

Site editing: main edit bar extension and page creation

A weak point in usability for editors is the way they have to use AdminCentral to get to the pages they want to edit. Then they have to get back to AdminCentral to publish it, or create a new page.

We are developing two extensions to the main edit bar. One that gives you feedback about the page status and lets you publish the page without leaving it, and one for creating a new page directly from the page you are editing. Template availability is taken into account. When the page is created, you navigate to it, and the main page dialog is opened, to make sure mandatory properties will be set.

To summerize: the following underlying problems are being addressed to some extent:

- Editors find it hard to create new pages. The fact they have to select a template (if they don't a random template will be assigned) is not obvious to them.
- Editors find it inconsistent to have to switch back to AdminCentral to publish a page.
- Auto generated paragraphs are only created when the page is requested for the first time. If an editor creates a page and then just publishes it, it has an illegal state. if a site visitor on a public node opens the page, the paragraphs can not be created, because the public user is not authorized to create the nodes.
- Fields in dialogs can be mandatory but it can not be enforced that the dialog is opened. We need a way to be sure certain properties are set when some page is created.
- When a page is created with the wrong template, and then opened, the auto generated paragraphs are generated, and changing the template does not change that, which is something most editors don't understand (it seems nothing has changed) and don't know how to remedy.

So on one hand it would be nice to create better tools for managing the site without AdminCentral, allowing editors to publish pages and create new ones. And on the other hand it would be nice (and will be necessary) if the page creation process, which is now split up into three unconnected phases (create- choose template, instantiate- create autogenerateds and configure- open the page dialog) would be unified in one comprehensible process.

Another great enhancement to auto generated paragraphs would be that apart from checking for their presence and if not: create them, a scan is made for auto generated paragraphs that are not required by the current template, and remove those.

Edit mode feedback

It would be very nice if the edit bar would display what type of paragraph it belongs to. the same thing applies to the dialogs. Especially when you add content to an area where only one type of paragraph is allowed, and you jump straight into that paragraphs dialog if you choose to add 'a' paragraph, It can be quite confusing.

Deployment - Hosting - Performance

Caching of multisite setup

We're in the process of defining our caching strategy. So far we have used the default caching and we got away with it all right, but as we are creating more sites we want something more sophisticated. We read all things Jan had to say on the subject in his blog, but there are still some uncertainties and we would very much like to discuss this subject with you, and perhaps hear of your future plans for caching and multi site support.

Asset compression and processing

We are wondering if it would be interesting to offer possibilities to automatically cat, compact and gzip javascript and CSS files. We think of something like:

1. Combine different javascripts into one file.
2. Minify this code
3. Gzip it.

Currently the first task is possible using a Freemarker script, but the rest is not. Minifying Javascript could be done using Google's Closure compiler, which is Apache licensed and written in Java, or maybe Yahoo's YUI Compressor.

Nils: there is mgnljawr.

Another thing we are thinking about is the possibility to make it easy to generate sprites for all images in a theme and use that in the css. An option for this might be to useSmartSprites, which is BSD licensed and written in Java.

Another possibility might be to include support for HAML and SASS (or LESS), alternatives for HTML and CSS writing that increase productivity and add extra options to write better HTML and CSS code. Of course, they could be in the same pipeline that gzips and minifies everything.