Concept - Migration to 4.5 (Version 1.2)

Your Rating: Results: 16 rates

In progress for 4.5.x

This is an effort to make the migration process simpler and to support the process of migration custom projects better.

Rationale/Status

The current Migration (using the migration tooling 1.1.x) is long-winded.

- a **monolithic script** to migrate our own modules and custom modules
    - its **not easy to split** the migration because some information has to be shared (especially in STK modules)
    - difficult to tailor the migration
- a standard setup (vanilla Magnolia and STK) still needs an execution of the scripts, the update are not implemented as update tasks
- groovy scripts are not easy debug/develop, we realized that we better use Java

Goal

Migrating of a project should not take longer than **20% of the original project effort.**

Concept

- get rid of the monolithic script, rewrite them as proper update tasks (in Java)
- each module uses ready made update tasks to process the update
    - the module's version handler executes the tasks explicitly
- a migration helper tool helps to prepare the manual migration of custom modules
    - preprocess freemarke files, detect areas
- the content migration is executed independently based on collected data during the migration process

Update of our Modules (STK, forms, ..)

- migrate the scripts to **Java**
- make **independent update tasks**
- each module triggers its own update by using the update tasks in their **version handler**
    - this is the **standard process** for all other updates
- shared **information is persisted** (persisted maps)
    - can be stored in the configuration workspace

Migration Helper Tool

The tool helps to **prepare the manual migration**. It will preprocess some files and produce a report.

Version Handler

- provide a skeleton which a developer can add to the version handler
- use our update tasks for the basic update
    - move paragraphs and page templates together
    - update the dialog id
- add a task for the creation of new areas (see below)

Templates

- the freemarker files are directly processed **on the filesystem** (point the tool to the path)
- **area scripts** get **extracted** and also created in the filesystem

This is what it will do:

- no code is deleted but commented out and the new code added below. This should help in the manual migration
- remove edit and new bars
- use cms:component instead of includeTemplate
- use cms:area instead of contentNodeIterator
- replace mgnl.* with cms.* and rewrite some of the functions

optional:

- provide a cleanup script to remove all the added comments

note: for each file a **manual verification** has to be performed

Areas

What it does:

- create skeleton with add area **update task** for the detected areas
    - a parametrized task per area
    - this task will **persist information** for the **content migration** (so that the node can be of type mgnl:area)

How it works:

- detect contentNodeIterator/newbar in the template scripts
    - extract code in a separate area script
    - add the area definition (in the generated script)
    - use the newbar's paragraphs attribute to configure the available components

Template Models

It is **currently a manual task** to migrate the template models (Java code):

- The **constructor** gets a Node passed (not a Content object as before)
- Methods used in the templates should return **ContentMap** not Content nor Node to allow nice syntax
    - its now a real map which allows the same syntax in JSPs
- ContentUtil.asContent(node) and Content.getJCRNode() can be used to **convert** the objects
- most methods on Content have an equivalent on Node

We should provide a tool which can do some of the changes automatically:

TBD: use AST? expressions? ....

Report

The helper tool creates a report to **ramp up the manual work**.

The report should look like:

- a report **page per template** (item)
    - link to the freemarker file
    - link to java files (the template model, version handler)
    - list detected areas
    - list potential problems if detected
- a **main page** linking to each of the pages
    - it should rate the templates based on: areas detected, lines of code, ...
    - this can be used to estimate the manuel work

Content Migration

The migration of large content has to be done **offline** and independently. This is why we have to **persist all information** about changes.

- the tool uses an input map and changes the templates and node types
- if the **property magnolia.update.content** is set to true, it is part of the normal update
    - this is mainly useful while testing/developing but should no be used in production

TBD: where do we store the information? I think in the repository?

Node Type Changes

Changing the template ids is rather easy, but it is technically bit more complicated to change the primary type of a node. There is no JCR or Jackrabbit API to perform such an operation.

We have few options:

A) Per node export/import. This is the current implementation.

B) Create an export file and process the xml file with a SAX operation. Then import it.

C) Use PersitanceManager level code to process the content.

TBD: decide what works best

Testing/Verification

Integration test for our update

- create an integration test which verifies the differences between a configuration export of an updated and freshly installed system
- use a document view export

Testing the custom migration

It is important that one can verify if the migration is completed. We propose the following

- create a **test/sample project** (before the migration) using all kind of templates/components
- use a crawler (like wget) to **export** the produced **HTML** pages
- reproduce the export on the migrated system
- use a **diff to verify** where you have differences or freemarker errors

Optional: support this process with tools