A paper in the InfoGlue Series

InfoGlue - Developer Manual

Author: Mattias Bogeblad

Version 2.9.0

Sponsors
We wish to thank **Nominet UK** (especially Jay Daley) for sponsoring the creation and release of this document to the community. You can read more about Nominet UK on www.nominet.org.uk.

Introduction

This document was written to give developers a useful guide to different topics when it comes to develop sites and functionality in InfoGlue. Hopefully the information herein will make developers more productive and comfortable with the various aspects of the system. Many developers will find it useful to read the entire document to get an overview of how InfoGlue works and some will only use it as a reference guide. Whatever the use, hope the information is valuable and easy to understand.

The document will not include information about pure user aspects or maintenance aspects as there are other documents that will specialize in these matters. The aim is to give developers detailed information about complex aspects of the system.

Feedback

This document is always a work in progress so mail any suggestions to bogeblad@yahoo.com and I will try to improve it as the platform evolves.

General concepts

What is InfoGlue

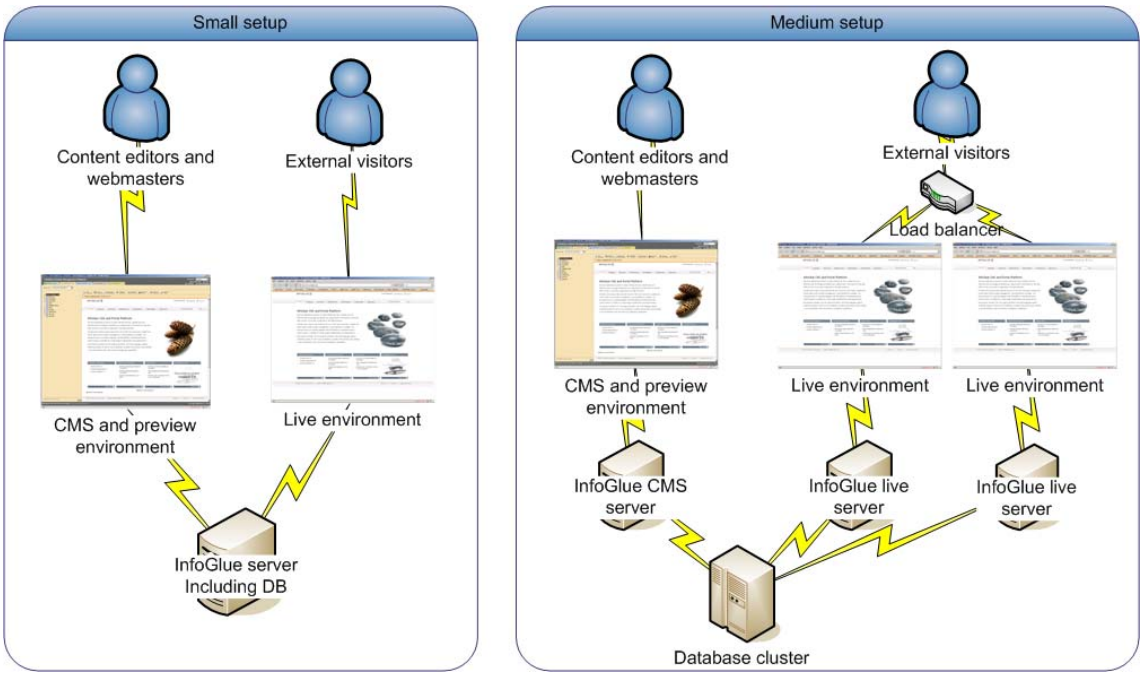
InfoGlue is a Content Management System. Content Management is very much another word for how to manage a company's or organization's information internally and externally. As content management is such a broad term we often say InfoGlue is a Web Content Management System. That is – InfoGlue helps organizations manage their information mainly targeted for the Web both internally, in intranets etc., and externally in public websites and extranets.

What parts are there in InfoGlue

InfoGlue is a pure Java platform. It is completely database driven which means both the management tool and the public sites are using information from a database. The platform consists of two main types of applications and they are the administrative tool and the delivery engine – the later in several instances.

The administrative tools are where you manage all aspects of your site. The delivery engines are specialized in presenting sites to users based on the data managed by the tools. By default InfoGlue installs 3 delivery engines. The first is the working version which presents the working version of the site. There is also a version called staging or preview which shows the site in a preview mode so the publisher can check that the site will look good after publication. The last delivery engine is the one that shows the live site to visitors.

Here are two sketches that shows first how a simple setup could look and then a more complex example with better redundancy:



How does it work

InfoGlue is a bit different than many other web content management systems. We have chosen to strongly separate the information that is to be shown to visitors from the form in which it is shown in. This is what in general terms are called separating content from presentation. The implication is that there are two different aspects to manage – the information itself on the one side, and the layout and flow of the websites presenting the information on the other side.

We usually call information in different forms in InfoGlue for "Content". A content in InfoGlue can be text, image, word-files, animations or anything else of informational value. There are no format limitations. Everything that concerns information/content is managed in what we call the content tool for obvious reasons. You can view it as a place were all information is stored and managed no matter where it should be used later on. You can for example use it as an image bank or a file-bank if you want to and never bother about the web at all.

Many other web content management systems are very page centric. This means that they require you to fill in the information specific to how it will look on a specific page in the site. They focus on how the information should be shown and not on how to keep the information structured. Often this results in having to type in the same information many times if it's to appear on several pages on the site or on different sites. That will never happen in InfoGlue as reuse is already prepared for. The people responsible for the information focus on the texts and not on the layout or website.

The presentation/site structure part of the InfoGlue system is what handles building specific websites and it is called the "Structure tool". With this tool you manage the site structure and chose which information to put on which pages and with which layout etc. You can also define links between pages among many other things.

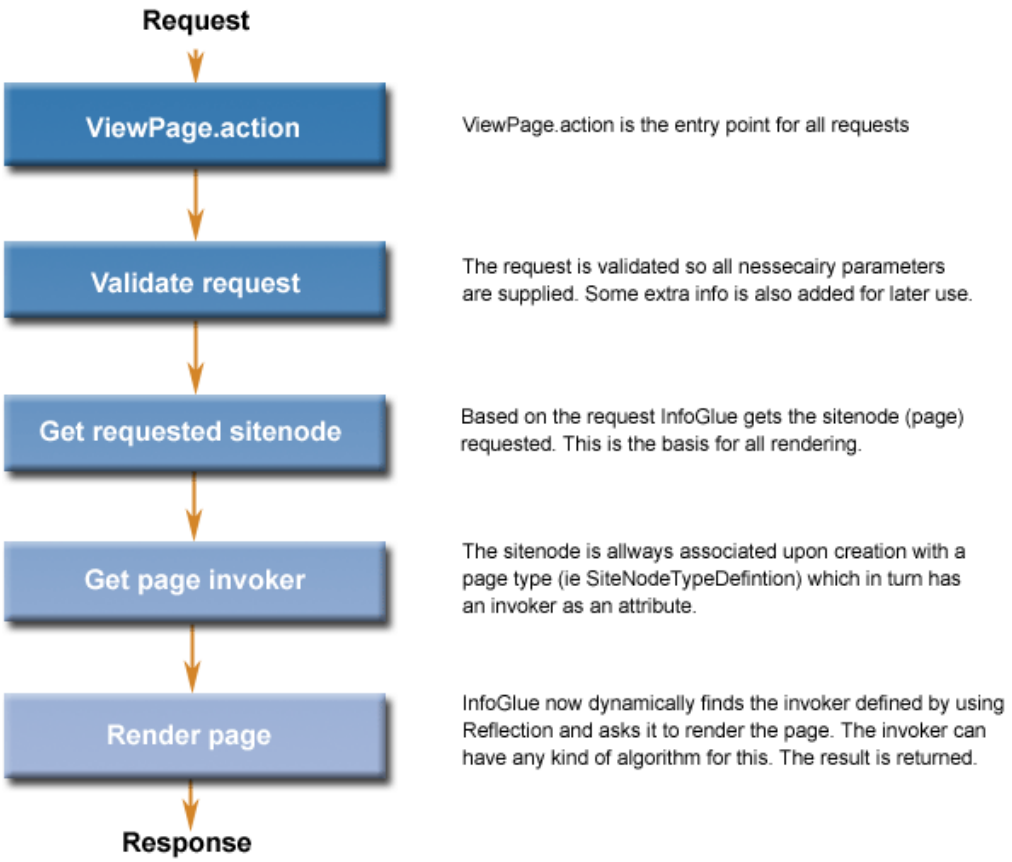
This division is very important to understand before working in InfoGlue. The concept is very strong once learned and much more flexible than page centric approaches but the learning curve is a bit steeper so don't give up if you find it hard in the beginning – it will pay off later.

InfoGlue pages behind the curtain

This chapter was meant to give users an overview of how a page is technically structured and rendered. It is very important to understand for developers and advanced users but ordinary users may skip this chapter.

The request/response cycle

A very important thing to understand in InfoGlue later on is how a page gets delivered to the user upon request. This information is mainly important for developers as they need to make some decisions affecting this process. We have tried to sum up the process very simple in this image:

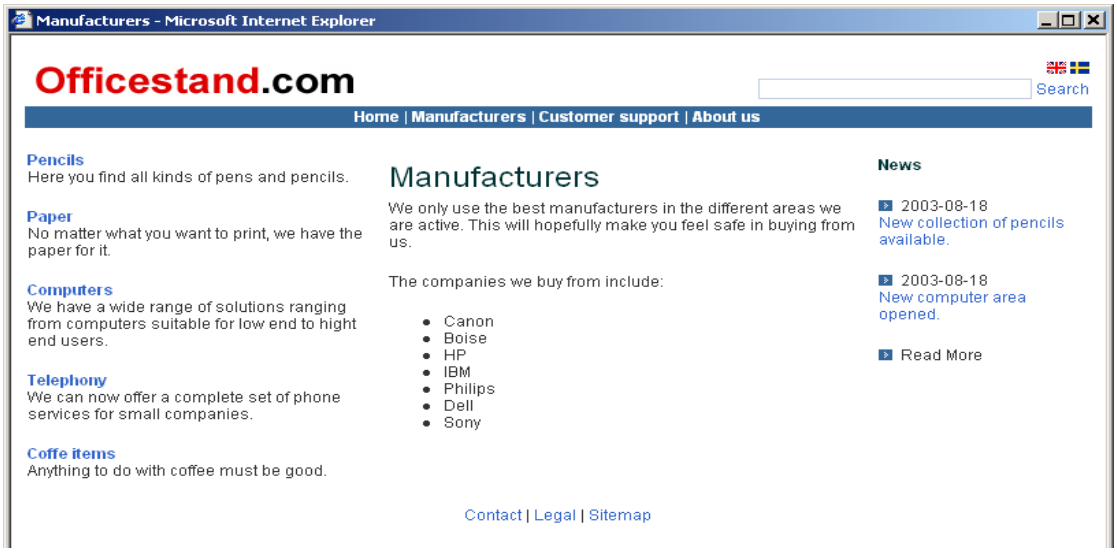


The important part to recognise here is the "Get page invoker"-step. Depending on which page type you have chosen for your page different invokers/algorithms can be used to render it. The types are defined in the management tool under "SiteNodeTypeDefinitions".

There are two default types defined in InfoGlue; one which renders pages in the old template-based way (Normal HTML page) and one that renders the page based on components on it. In addition you can even add your own invoker if needed. An example could be a WAP-page-type which behaves differently than HTML-pages does altogether or perhaps a redirect invoker which forwards the request sever side.

A page in detail

To describe we have the very simple page as an example. The basic look of a page would be something like the page below.



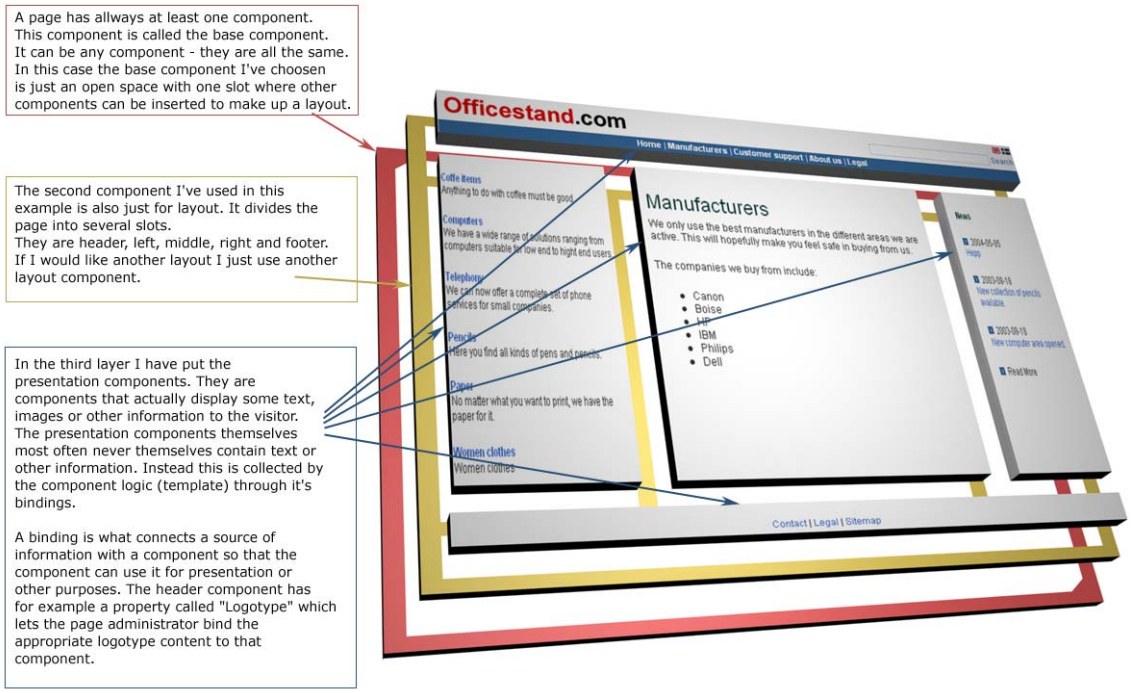
This page is very simple but the technique is the same no matter what layout we would use or what information we want to show. The page layout is controlled by components.

The page you are looking at could be structured in many different ways, depending on the developer's choices, but in this example it is composed out of many small/medium-sized components. A component can have any size and we could for example have one big component controlling the entire page if we want to but that would mean we could not manage the layout in detail for different pages. Instead we have used smaller components responsible for only a part of the page. For example there is a header component which is responsible for showing the logotype, flags and the navigation.

The concept of an InfoGlue component is very similar to the "includes"-concept found in many web environments like ASP/JSP. The difference is that the components can have dynamic properties and can be managed graphically in InfoGlue by non coders.

The structure of a page is better shown in this drawing. Note especially that it is by assigning contents and pages to appropriate components that you actually give the components enough data to present information to the user. The component themselves most often only contain layout logic and knows how to fetch the information you assign to it.

A simple page component structure



Hopefully this section has helped you to understand the basics of how a web page is built in InfoGlue. We will now take you through the different tools to show you where the work is done.

In all chapters we will assume that you have logged into the InfoGlue administrative tool. If you don’t know how to find it or have questions please consult you administrator.

Development process

InfoGlue is not some magic tool that let you skip the thinking process. On the contrary – by doing a good analysis before starting to build you can achieve a much better solution for your clients. Many things concern how you want the users to interact with the system or which way they feel comfortable in managing their site and the information in it. In this document we assume you have a functional spec as well as a design proposal which are to be implemented.

Site/Page analysis

The first thing to do is to go through all the pages and analyse them in detail. Questions to ask are:

- What types of information are there and how we can generalize this into information types.
- What parts of the pages are there, how can we create a layout system which is both simple and easy to maintain and still dynamic and extendable for users who want that. What components do we have to build to make it happen?
- What complex functionalities are there, how we solve it?
- Are there some external systems that should be integrated somehow and if so – how is this best done.
- Do we need to create new page types other than the default? (unlikely)

Content types

One of the most important artefacts that should come out of this is a list of suitable content types. A content type is what a developer would use a Class for in Java. It defines what attributes a certain information type should have. A typical content type is "Product" which perhaps has the attributes "Name", "Description" and "Price". It may also have an attachment labelled "Product Image". InfoGlue contains some default types for demo purposes but it's up to the project to set up ones suitable for the customer. This is done in the management tool without writing a single line of code and is described in the user manual.

Template / Component-list

The next thing the analysis should result in is a list of templates/components needed to visualize the site. Most often a page can be fragmented into smaller reusable templates/components but it all depends on the design. This list is a good starting point when estimating the development effort as well as a good document to continue working on as part of the site documentation. We usually describe a component both visually, functionally as well as how the users are to interact / supply it with information etc.

Tools

Built in tools

You can work with InfoGlue without any external tools at all. All templates/components and everything is there as pure text editable though the normal browser text-editor. When it comes to developers they however often have some other tools they prefer simply because it helps them in their job. Editing a JSP-based InfoGlue-component for example is of course easier if you use Eclipse with a good JSP-editor plug-in.

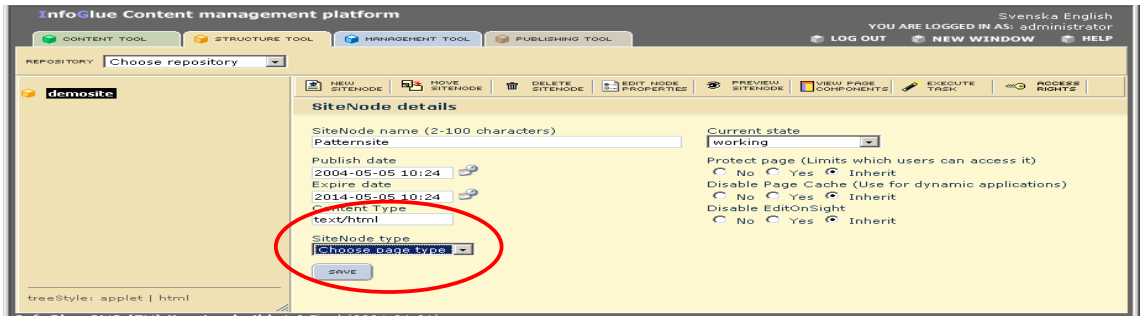
Plugins

When creating templates/components there is a lot of HTML/CSS/Javascript/JSP-coding that has to be done. InfoGlue therefore comes with a eclipse plugin which allows you to edit your code directly in eclipse and get automatic sync with the server of choice. Not to use this tool is plain stupid as it saves all developers a lot of time. The plugin is free to download and use from www.infoglue.org.

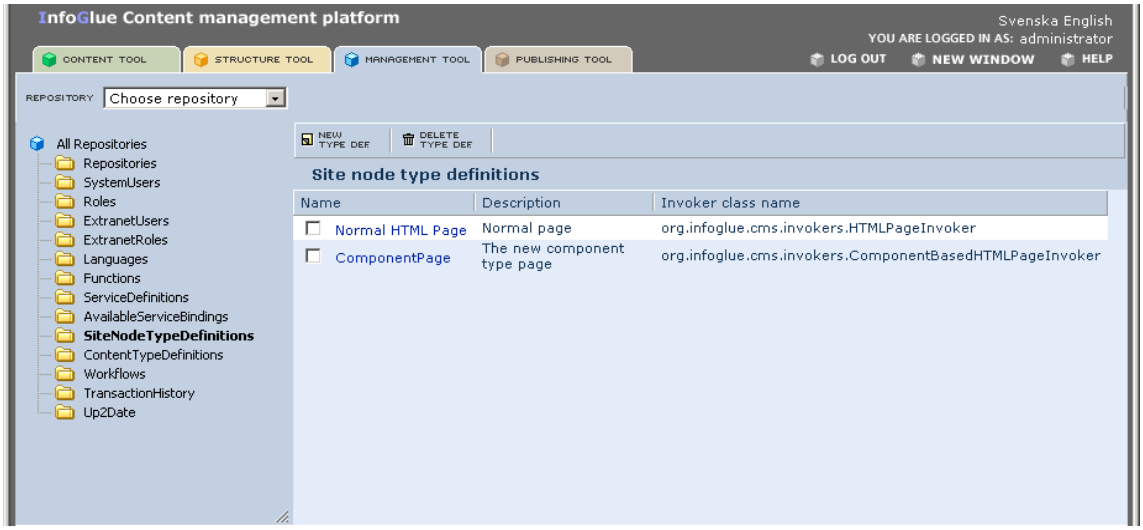
Building pages in InfoGlue

Basics about Page types

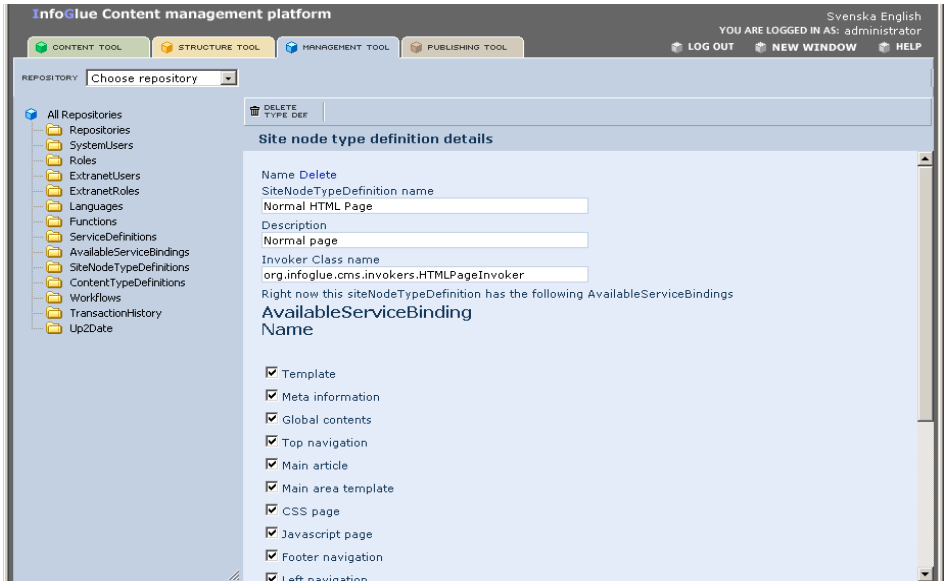
When you create a new site node (page) in InfoGlue you will have to state what site node type you want it to be based on. We already explained the implications in the part about the request/response cycle so you hopefully understand why it is important. You define it in the following field:



The types are defined in the management tool under "SiteNodeTypeDefinitions".

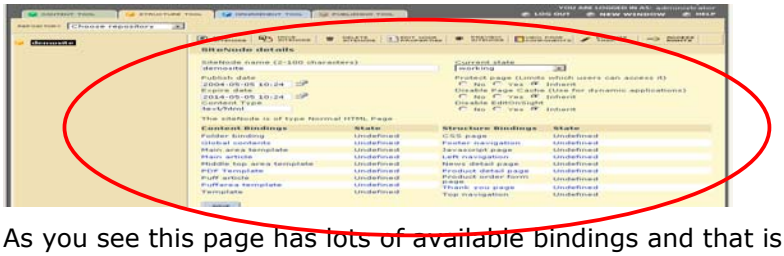


On each "Site Node Type Definition" you have the following fields:



The name and description field is just for presentation but the Invoker class name is important as this is the name of the invoker class used by InfoGlue for dynamic instantiation. In this view you can also state what Available Service Bindings this page type should show as available.

The available service bindings you mark here will be shown on the Site Node view:



As you see this page has lots of available bindings and that is pretty normal for the old way of building pages. The concept is strong but somewhat limited when it comes to complex pages with a lot of information from different sources. This way of defining global bindings can be successfully combined with the possibility of creating component-binding in a mixed mode solution. A content such as general labels or something else is probably better off defined on the root page level than in each component using it for example.

OBS: We recommend you only build sites with the component based system and not use the old way any more as that way will not be developed any further.

Basics about connecting content and other resources to pages

We have previously established how separated content and presentation is from each other. We have explained that the structural part of the system by default have no idea of what is stored in the content part unless we somehow tell it. We have also just shown how page types and available service bindings make up possible resource connectors. This gets us to the area of bindings.

When a page is rendered in InfoGlue it is done so by rendering the templates that page uses. When the system renders a template/component the template/component most often are meant to retrieve texts and images from the content part to show and structural information from the structure part so it can link to other pages in navigational elements. To give the template something to use as a reference InfoGlue has something called bindings.

A binding is essentially a reference associating a site node (page) with a resource of some kind. Most often it is one or many contents or one or many pages. A site node (page) can have any number of such references and they are created by the developers of the site to fit the project need.

The old way had global page bindings which all templates on a page shared which are shown in the image above. The component based technique allows you to have bindings on component level so it gets more object oriented. The benefits of this will be explained later on.

In a later chapter we will describe some common uses of bindings in their contexts so don’t worry if you think it’s hard to understand at the moment.

Components

Composite pattern

InfoGlue implements the composite pattern which means that a page can be built up of small fragments. The composite pattern is implemented so that a component can have so called "Slots" in it. It can have any number of slots and in each slot you can then put 0..n other components. You can have any number of compositions which means there is no limits to what you may do.

Creating a component

You create a new component just by creating a new content in the content tool with the content type "HTMLTemplate". The old templates from the old way of building are actually just as good components as new ones except that they don't operate on component-bindings but only on page-bindings if you don't modify their logic.

When you have created a component and added the HTML to the language version you want you should also create any slots needed, categorize it and set up its properties if needed.

Slots

A slot is as previously stated a place holder for other components. You don't need to do much to create a slot. Just add the tag **<ig:slot id="slotname"></ig:slot>** to the component html at the position you choose. There are a couple of optional attributes to the slot-tag: **inherit, disableAccessControl, disallowedComponentNames** and **allowedComponentNames**. Inherit is used to state if the slot should be inherited to pages below (if not overridden) or not. The allowedComponentNames and disallowedComponentNames attribute allows you to specify a list of component names that are allowed not allowed in this slot. Here is a full example:

```
<ig:slot id="rightColumn" inherit="false"
allowedComponentNames="PuffImage,Related Info,Searchform"></ig:slot>
```

Special tags

There is a tag for printing out the last modified date of a page. The date is the last modified date of all content on the page. Here is the syntax:

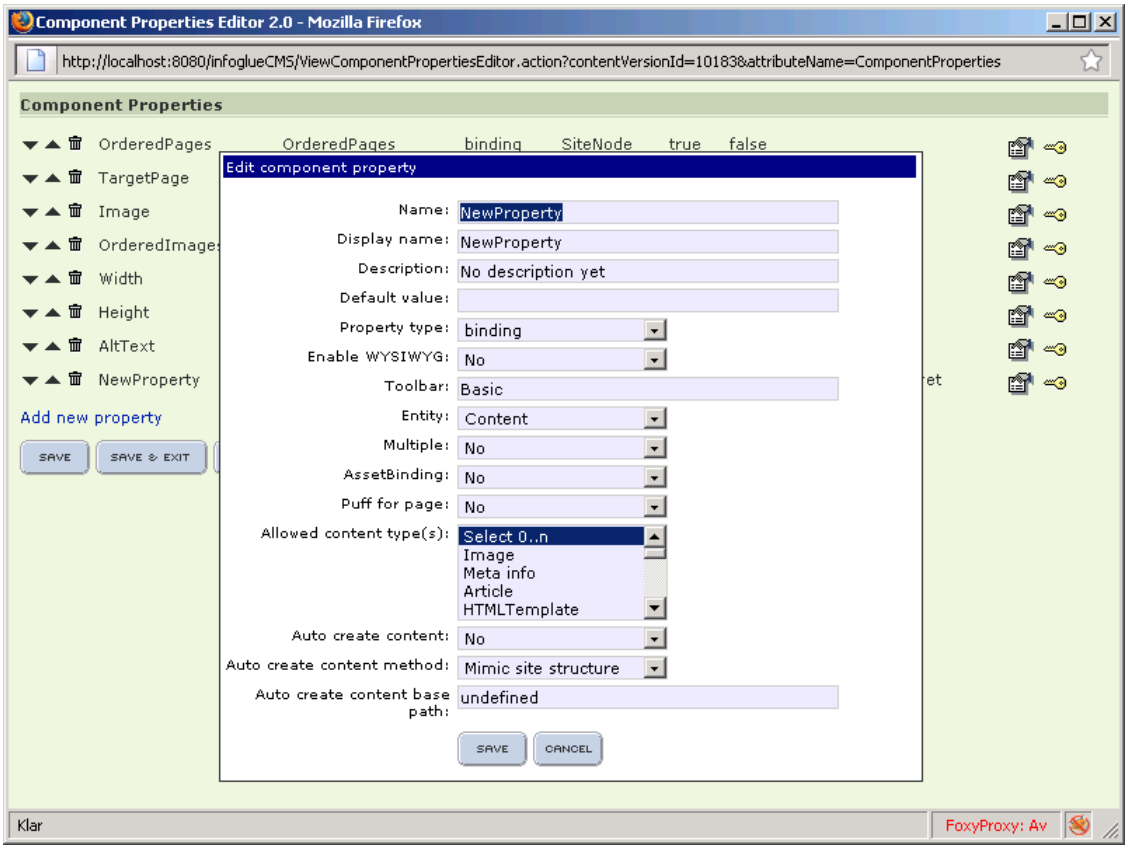
```
<ig:lastModifiedDateTime format="yyyy-MM-dd
HH:ss"></ig:lastModifiedDateTime>
```

Properties & bindings

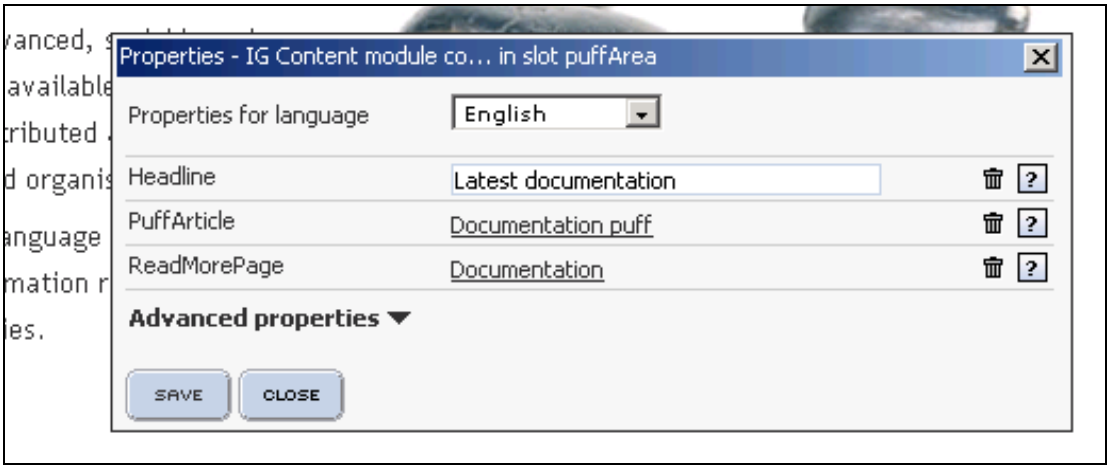
One of the main benefits of components compared to other systems where you are limited to jsp-files on disk or similar is that they can have properties which controls their behaviour. This means that an instance of the component on one page can have different settings than the same component located on another page or on the same page for that matter. This enables developers to write very dynamic and reusable components if needed. The users can configure their sites rather than having developers helping them all the time.

You define properties in the attribute field "ComponentProperties" in the HTMLTemplate-content you create for each component and the format is XML. Don't worry – there is a property editor built in so you don't have to hack away but it has to be turned on.

Below is an example of the dialog of one property when you use the property editor. Let us go through the fields:



Here is an example of a component property dialog in edit on sight for reference (not the same component as above):



Note: Under advanced properties some standard properties can always be found. They are not defined here.

Description of fields in component editor:

Name: Represents the internal name you want to use in the template to get the property. Also what the user in edit on sight sees if Display name is left empty.

Display name: This is the name the user see.

Description: This is a text which is shown if you mouse over the icon.

Default value (only applicable for non-bindings): Let's you set a default value if not set by user.

Property type: Most important – defines what kind of field should appear. The available are:

- Binding – used to bind 1..n content(s) or page(s).
- Textfield – simple textfield.
- Textarea – simple textarea – with or without WYSIWYG.
- Select – select / drop box field.
- Checkbox – 1..n number of checkboxes.
- Datefield – let's you allow users to select a date.
- Customfield – this allows you to insert you own HTML markup which means you can add anything including javascript logic.

Enable WYSIWYG (only applicable if textarea is shown): Enables the.

Toolbar (only applicable if textarea is shown and Enable WYSIWYG is true): Let's you select which toolbar to show.

Entity (only applicable for binding-type property): State if it's a binding to a content or a page/sitenode.

Multiple (only applicable for binding-type property): State if it's a binding to one or many of choosen entity type.

AssetBinding (only applicable for binding-type binding and entity=content): State if the property is a digital asset. Shows a different dialog.

Puff for page (only applicable for binding-type binding and entity=content): This is a little special. It was developed for users who want for example to reuse an article bound on a subpage for a startpage or similar and want a link in that component to automatically find the subpage where the contents main location is and link there. Having this set to true the property will handle if the bound content exists on several pages with dialog options.

Allowed content type(s) (only applicable for binding-type binding and entity=content): Let's you limit what kind of contents can be bound.

Auto create content (only applicable for binding-type binding and entity=content): If set to Yes the component will automatically create and bind a content to the property. This is very useful if you want to simplify for your users. With this they will not have to know much about the content tool or content structure.

Auto create content method (only applicable if Auto create content=true): State where automatical content should be placed. We recommend Mimic site structure.

Auto create content base path (only applicable if Auto create content=true): Let's you state a base path before the site structure minic kicks in. Example /Autoarticles

Custom markup (only visible if you choose type customfield): As said before you can set any markup you want including javascript. The only worry you have is to set the correct name for save and fetch of the values you set. To make it persistent your value has to be able to serialize as a string in an field of some type. The field should be defined with the name propertyName which will be automatically translated by infoglue to correct id. To get the previous saved value you just use "propertyValue" which will be replaced by the saved value by infoglue. Look at a simple example below of custom markup:

```
<div style="border: 1px solid red; width: 200px; height: 100px;">Example<br/><a
href="javascript:alert('Test')">Test</a>  <input type="text" class="propertytextfield"
name="propertyName" value="propertyValue" onkeydown="setDirty();"> </div>
```

Component caching

When it comes to component caching it's not all that hard. The reason for the mechanism is of course to be able to boast performance on partly dynamic pages. First you have to turn off the page caching on the page or the pages you wish to keep partly dynamic. Then you add a few properties to the components you wish to cache.

If you have a menu component which you want to cache you enter the advanced properties on that component. The cache-related properties are:

PropertyName: CacheResult
Type: Dropbox
Possible values: "true" / "false"
Description: If you set this to "true" the component will be cached until either it expires or a new publication is made. Subcomponents will also be cached with it.

PropertyName: UpdateInterval
Type: Dropbox
Possible values: -1 and up.
Description: If you have the CacheResult property set to true and you set this to a positive number the components result will be cached for as many milliseconds.

PropertyName: CacheKey
Type: Textfield
Possible values: Works like the page cache key and defaults to the same key. Use this with caution. It's optional.

Component tasks

In InfoGlue's structure tool we have shown that there is a context sensitive environment which we call "Edit On Sight". The right-click menu is not limited to the items in it actually but can be extended with component-specific tasks.

To use this you define an attribute field called "ComponentTasks" in the HTMLTemplate-content-type and for each component you wish to add context menu items for you set an xml in that field:

```
<?xml version="1.0" encoding="UTF-8"?>
<tasks>
<task name="Create subpage" openInPopup="false"
view="$componentEditorUrl/CreateSiteNodeWizardFinish.action?repositoryId=$repositoryId&amp;par
entSiteNodeId=$siteNodeId&amp;languageId=$languageId&amp;componentId=$componentId&amp;property
Name=dummy&amp;refreshAddress=$originalFullURL"/>
</tasks>
```

The example above is taken from a menu component and the task let's the user rightclick on the menu and select an menu item called "Create subpage" which with this configuration will start a wizard which creates a new subpage to the page the user is currently watching and then returns to the page again. Very nice if you want to empower the users without letting them into the main tools.

Intercomponent communication / data sharing

There is a need on most advanced sites to be able to share data between different components. As a simple example one might for example want a related info component pick up information that the main area component has knowledge of or affect the page title with some subcomponent data.

There are some possibilities already in InfoGlue but they are either that you can read other components properties or if subcomponents wants to read a parent components shared data. Before 2.9 anything else was impossible.

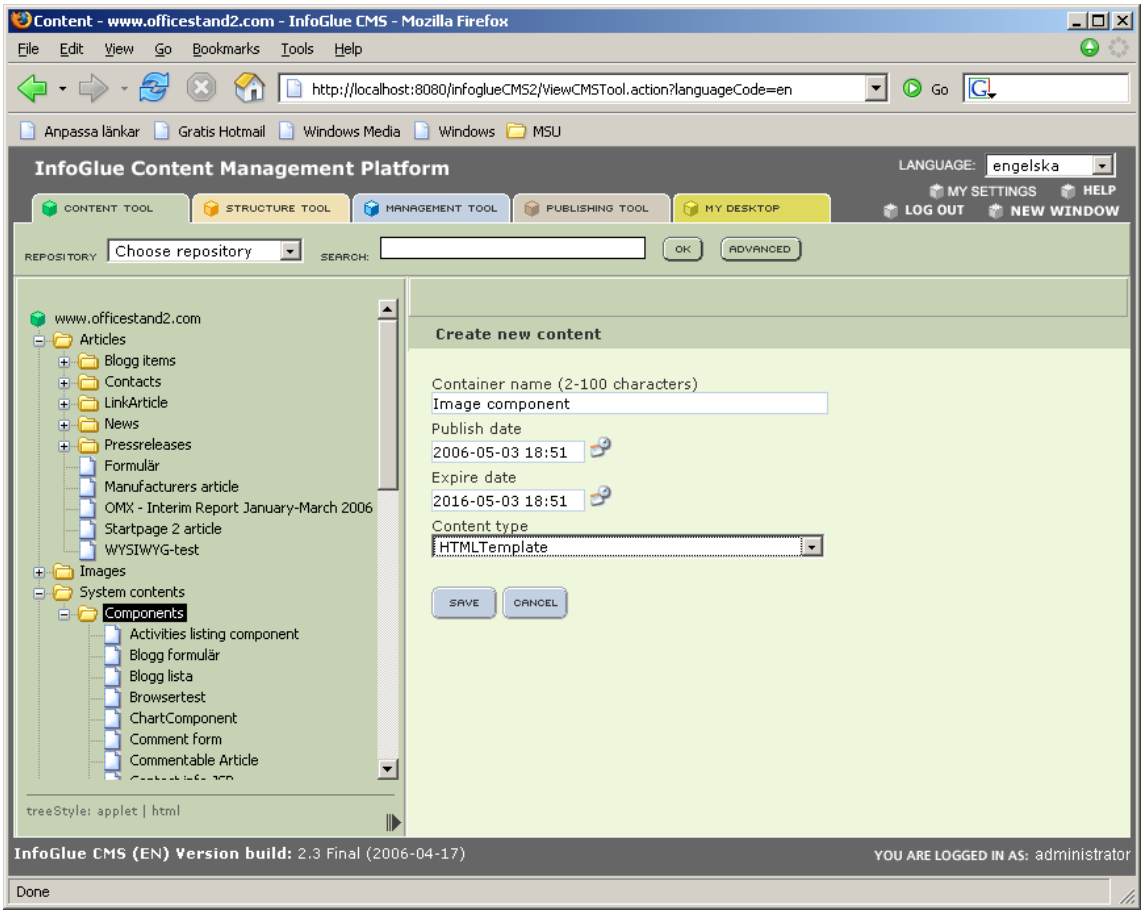
Since 2.9 there is a new solution. The solution is based on that every component can have a preprocess phase if wanted. If a component has a PreTemplate it (and all other components PreTemplates) will run first in the preprocessing phase where they will be free to put data into the page context for the normal presentation templates to read. Before the preprocessing all components on the page with a PreTemplate is ordered in any sortorder defined in advanced properties on each component. The PreTemplate can do anything the normal Template can except print to the browser so you should consider it the dataprocessing part and not a presentation part.

After the PreTemplate-phase the normal rendering is run just like earlier but now you can use the shared data now.

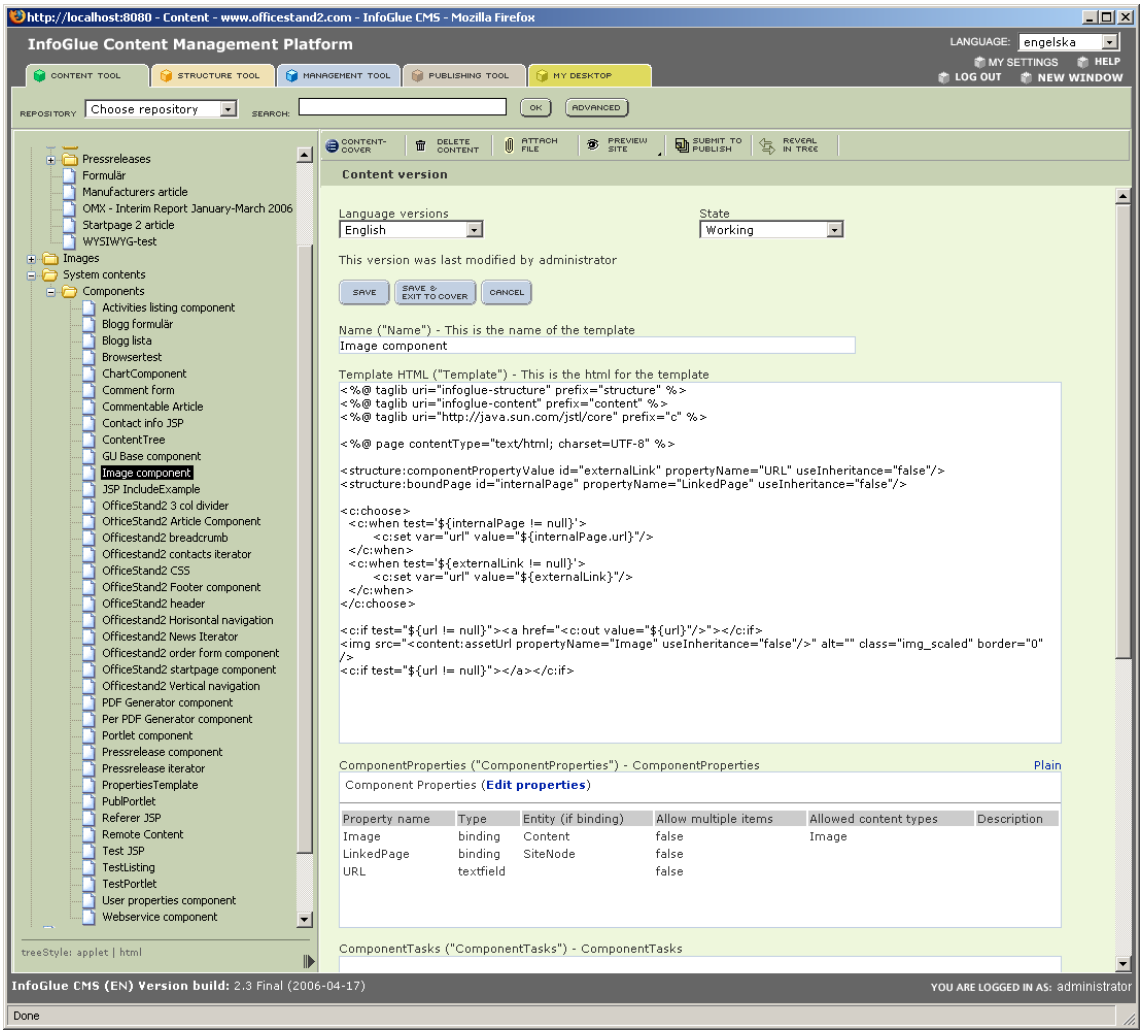
How it looks

Here is an example of how it looks at the different states when you create a simple component:

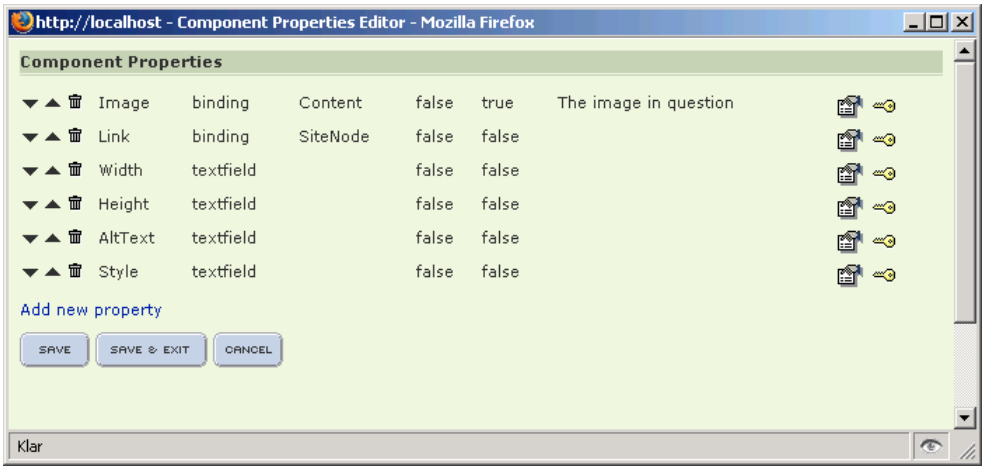
Create a new content with the content type HTMLTemplate.



The form for this content type looks like this:



The lowest field "ComponentProperties" is important and the editor for properties can be launched by clicking the "Edit properties"-link. If you wish to see the XML press the Plain-link. If you open the editor you will see this:



Hopefully the editor is rather self explanatory although it has some usability flaws.

Component description and thumbnail

Since InfoGlue 2.6.0 we have added a possibility to add a description and a thumbnail for components. It's all optional of course but could be nice for end users. The info is shown like this when adding components to a page (compare the AK Article Box component which is supplied with another thumbnail and a description with the other components):

- To add a description just add a content type attribute called "Description" on the HTMLTemplate content type. Then enter a description for those components you want to have a description for in the gui above.
- To add a thumbnail just attach a normal image with the asset key called "thumbnail" to the component content.

API

The component example shown above is JSP-based. You can develop components in JSP, Velocity or Freemarker in InfoGlue 2.3. The system recognizes JSP-templates on its own and let's tomcat compile them. You can of course write scriptlets which uses the BasicTemplateController as you do in velocity but an even more elegant solution is to use the many taglibs supplied with InfoGlue. Last in this document is a full reference documentation to the tags including examples of how to use them. This is from 2.1 the preferred way to build templates / components in InfoGlue although the other techniques will be supported and developed as well.

When it comes to Velocity and its close relative Freemarker there are two API:s to be aware of when it comes to components. You will want to use the $templateLogic-object a lot probably as it contains many good utility methods. Also you want to look into the new $templateLogic.componentLogic-object as that contains most of the methods you need for the component specific stuff.

The Javadoc-API:s is found on:

http://www.infoglue.org/infoglueDeliverLive/projects/infoglue_cms/Javadoc_API

Integration

This is a hard section to write as integration is such a huge area. Here we have tried to line up some common techniques we know people are using and describe pro / cons for each.

Simple integration via Iframes

Sometimes organisations have a need to embed existing applications as they are. In such cases using IFrames or ordinary frames could be a valid option. It really has nothing to do with InfoGlue as the technology is supplied by the browsers directly. We however thought it should be mentioned as it is commonly used as a way of achieving integration fast. The positive effects are that it works instantly in most cases and no modifications has to be done to the application. The negative side is that frames can be ugly, the size of the frame is fixed which can result in scrolling a part of the page and that the surrounding page often has a different style.

Simple integration via embedding

If the IFrame solution is inadequate for the particular needs another option is something we call backend embedding. The technique is rather basic and means that InfoGlue does a HTTP-request to the URL you specify like a browser would and then returns the resulting HTML for you to present or process. The benefits of this approach are:

- The user is unaware of that there are different parts of the page coming from different locations as the result is inserted in the html code and the page is delivered as one unit.
- You can process the resulting HTML and change styles etc. if you want to have a common style on the page.
- Security – you don't have to allow the user direct access to the embedded application – it is enough if the InfoGlue machine is authorized.
- You can have the embedded webapp call InfoGlue-methods like if it were a template if it wants to use images/labels etc. when embedded.

The drawbacks are:

The performance of the entire page is very dependent of the performance of the embedded webapp. If the webapp is interactive you will have to do pretty much pre/post-processing of the request and HTML to make it work back and forth. In some cases it will not work at all if the state handling is advanced in the embedded webapp.

Here is a small example-template that basically fetches the startpage from http://w3c.org and embeds it into the page. JSP has a taglib which does this as well.

```
<html>
<head><title>Embedded</title></head>
<body>

#set($remoteContent = $templateLogic.getUrlContent("http://www.w3c.org"))
Here is the W3C content: $remoteContent

</body>
</html>
```

Advanced integration via custom classes in velocity

If you wish to do embedding of external resources in a more advanced way you will probably want to integrate by using pre made or custom backend logic. A very common situation is when you have some internal or external web service or database you wish to fetch information from but you just want to fetch the data and still let InfoGlue handle the presentation.

The solution is to write custom classes. In InfoGlue this is very simple. As long as you have an empty constructor in your Java class and put it in the InfoGlue class path it will be usable from any template in your site.

Let's say you create a class like this:

```
import java.util.List;

public class MyClass
{
    public MyClass()
    {
    }

    public String getHelloWorld()
    {
        return "Hello World";
    }

    public List getOrders(String customerNumber)
    {
        //Here we could connect to the order database with jdbc and query it instead
        return null;
    }
}
```

Then after compiling it and putting the class file in the deliver class path you can reference it like this in the templates:

```
<html>
<head><title>Embedded</title></head>
<body>

#set($myClass = $templateLogic.getObjectWithName("MyClass"))
The class says $myClass.getHelloWorld() to us.

</body>
</html>
```

If you are using JSP you are probably used to importing custom classes so I will not describe that here.

Portlets

A fairly new standard InfoGlue is supporting is the Portlet standard also known as JSR 168. Since InfoGlue 2.0 InfoGlue has incorporated Apache Pluto which is the reference implementation of the portlet API. Portlets are covered on the Internet extensively why we will not leave a lot of room here.

We will state however that for an organisation that has competence in such development this option is probably one of the best integration options as it will allow you to build a completely decoupled web application but seamlessly integrate it in an InfoGlue page and having interaction etc. taken care of. In a later section we will show you how to build a simple portlet and deploy it in InfoGlue. Then you will have to read more on the subject yourself.

Tasks

Introduction

When implementing a content management system there are many aspects to handle. One of the most important ones is usability. As some aspects of site management may involve quite a few steps in the normal tools there are often a need to be able to help the users. One way is to create workflows which will be described later on. Another simpler way is by creating what we in InfoGlue call "tasks". A task is something you write to extend the normal user interfaces without having to change anything in InfoGlue itself.

A good example of where writing a task is suitable might be if you want to give the Vice President a possibility to write a monthly letter which should be published both on the web and sent to all customers on email without having to teach him all the steps involved using the standard tools. You can write a task which he runs which guides him through all the steps in a very user friendly way. The interface can be totally customized to make him/her feel at home and if the user has proposals those can be taken into consideration as well.

We have examples of people using tasks for the following purposes:

- Creating whole sub sites with predefined structure
- Link validation on all contents
- Export/Import of contents
- Sending weekly newsletter to all customers
- Publishing start page news/puffs easy.

The decision whether or not a task is required in a process is up to the developers and the users to agree upon.

Flow of a task

There are basically three steps in an ordinary task:

Initially it's the user input step. Here the central task executor will run the script located in the UserInputHTML-field of the task definition Usually this step is used to present the user with an introduction and let him/her contribute with information needed in the coming steps. You can have several calls to this step if you wish to split the step into smaller screens but this is handled in the same script in that case.

Second the task executes the script located in the attribute ScriptCode. This step normally contains the manipulating operations. This can mean creating content, pages, languages as well as any other operation available in the platform as well as in custom logic.

Thirdly – when the execution of step 2 is done the script in field UserOutputHTML is executed. Most often this is used to present the user with the result of the process including any files/assets/info the user needs.

A very central thing is of course to understand how to pass data between the different steps. When it comes to passing data the normal servlet API is usually what you need. You can always reach the request-object from your scripts. Also notable is that each step in a task is handled in a separate transaction automatically.

Portlet development

This section will guide you through creating a simple Hello World portlet and deploying it in InfoGlue 2.3. It will not show you how to build portlets with more sophisticated frameworks like Struts or Webwork as that is not a question for InfoGlue.

Step 1 – Create a Hello World Portlet.

Create a new Java project called HelloPortlet in your favourite development environment.

Create a file called org/infoglue/portlets/HelloPortlet.java and fill it with this:

```
package org.infoglue.portlets;

import java.io.PrintWriter;
import java.io.IOException;

import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;

/**
 * HelloWorld Portlet
 */

public class HelloPortlet extends GenericPortlet
{
             public void doView(RenderRequest req, RenderResponse res) throws
PortletException, IOException
             {
                          res.setContentType("text/html");
                          PrintWriter out = res.getWriter();
                          out.println("<h1>Hello World</h1>");
             }
}
```

The file needs portlet-api-1.0.jar in its class path to compile. Get it from tomcat/shared/lib if InfoGlue is installed, from the Pluto distributions or from any other official site.

Next we create a portlet.xml:

```
<portlet-app
    xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
    version="1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd
                        http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">

  <portlet>
    <description>Hello Portlet</description>
    <portlet-name>HelloPortlet</portlet-name>
    <display-name>Hello Portlet</display-name>
    <portlet-class>org.infoglue.portlets.HelloPortlet</portlet-class>

    <expiration-cache>-1</expiration-cache>

    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>VIEW</portlet-mode>
    </supports>

    <portlet-info>
      <title>Hello Portlet</title>
      <short-title>Hello Portlet</short-title>
      <keywords>Hello,Portlet</keywords>
    </portlet-info>
  </portlet>
</portlet-app>
```

Next we create a web.xml for this portlet:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
                         "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
    <display-name>Hello Portlet</display-name>
    <servlet>
        <servlet-name>HelloPortlet</servlet-name>
        <display-name>HelloPortlet Wrapper</display-name>
        <description>Hello Portlet Wrapper</description>
        <servlet-class>org.apache.pluto.core.PortletServlet</servlet-class>
        <init-param>
            <param-name>portlet-guid</param-name>
            <param-value>Hello.HelloPortlet</param-value>
        </init-param>
        <init-param>
            <param-name>portlet-class</param-name>
            <param-value>org.infoglue.portlets.HelloPortlet</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloPortlet</servlet-name>
        <url-pattern>/HelloPortlet/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Step 2 – Package the portlet as a WAR-file.

I just create a war-file with the following contents.

HelloPortlet.war

-- web.xml

-- portlet.xml

-- WEB-INF

-- classes

-- org

-- infoglue

-- portlets

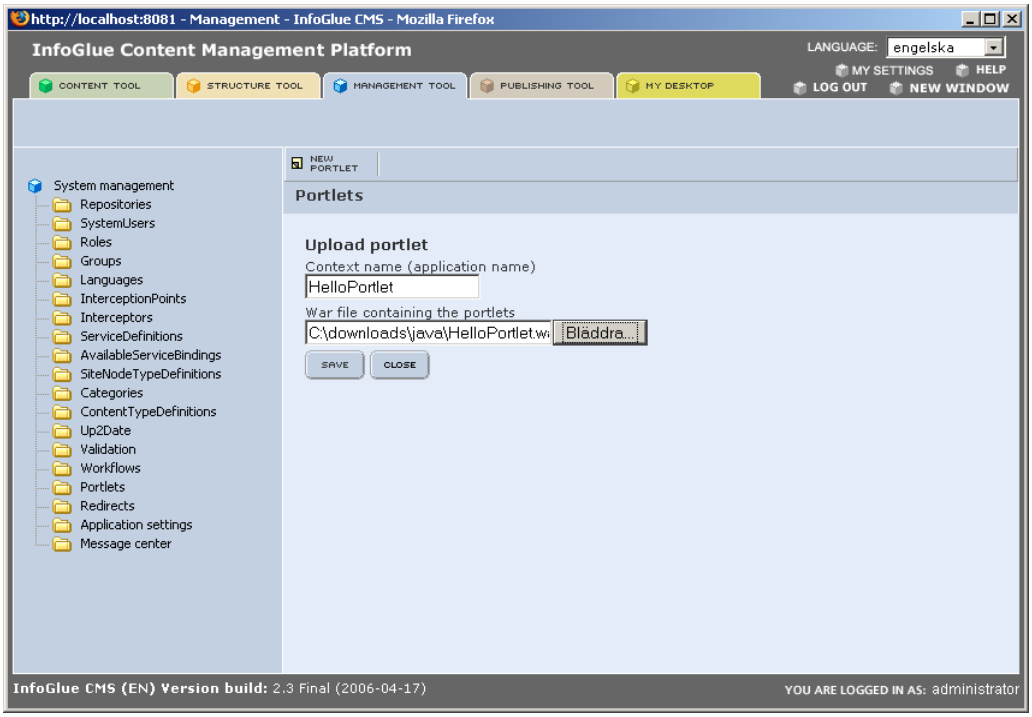
-- HelloPortlet.class

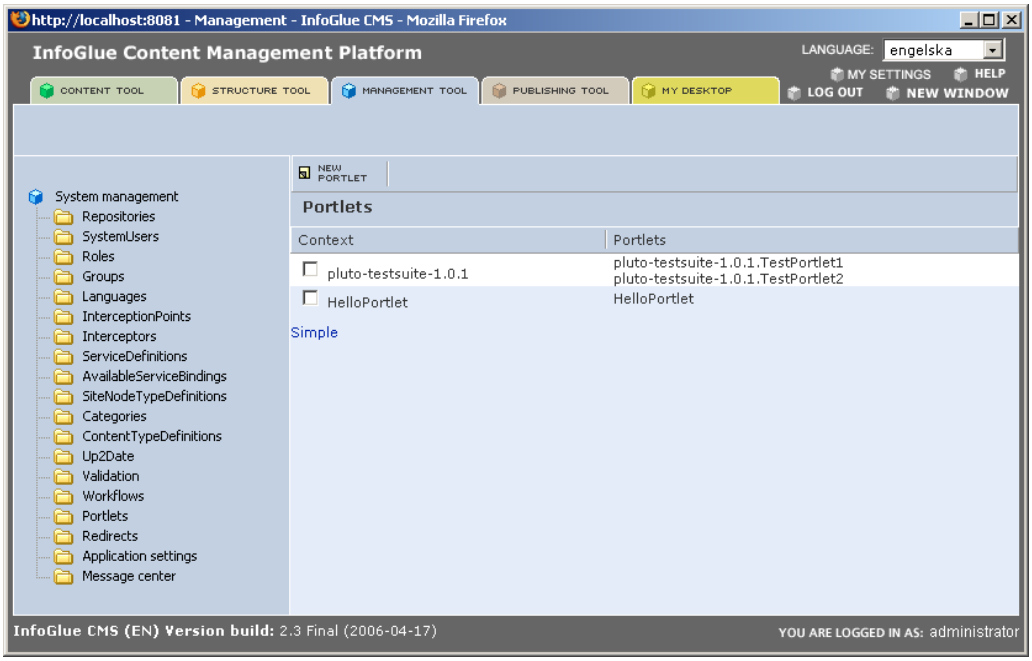
That is it really.

You can download the example war from http://www.infoglue.org/downloads.

Step 3 – Deploy the portlet in InfoGlue.

Now we deploy the portlet in InfoGlue. Open up the management tool and press the ”New Portlet”-button. Then state HelloPortlet and browse to the war-file. Finish by pressing ”Save”.





The result should be that the new portlet is listed.

Step 4 – Use the portlet on a page though a component.

Before you can use the new portlet often you have to restart tomcat after this stage. Also remove the deployed war-file from the catalina webapps directory as the webapp should now unpacked and the war is just interfering. **Do this before you continue.**

Create a component/template with the following code:

```
#set($portletName = "HelloPortlet.HelloPortlet")
#set($calendarPortlet = $portalLogic.getPortletWindow($portletName,
"p$templateLogic.componentLogic.infoGlueComponent.id"))
$calendarPortlet.setAttribute("componentId",
$templateLogic.componentLogic.infoGlueComponent.id)

$calendarPortlet.setAttribute("languageCode", $templateLogic.locale.language)
$calendarPortlet.setAttribute("siteNodeId", $templateLogic.siteNodeId)

$calendarPortlet.render()
```

Now just add this component to a page and you should see ”Hello World” in large letters.

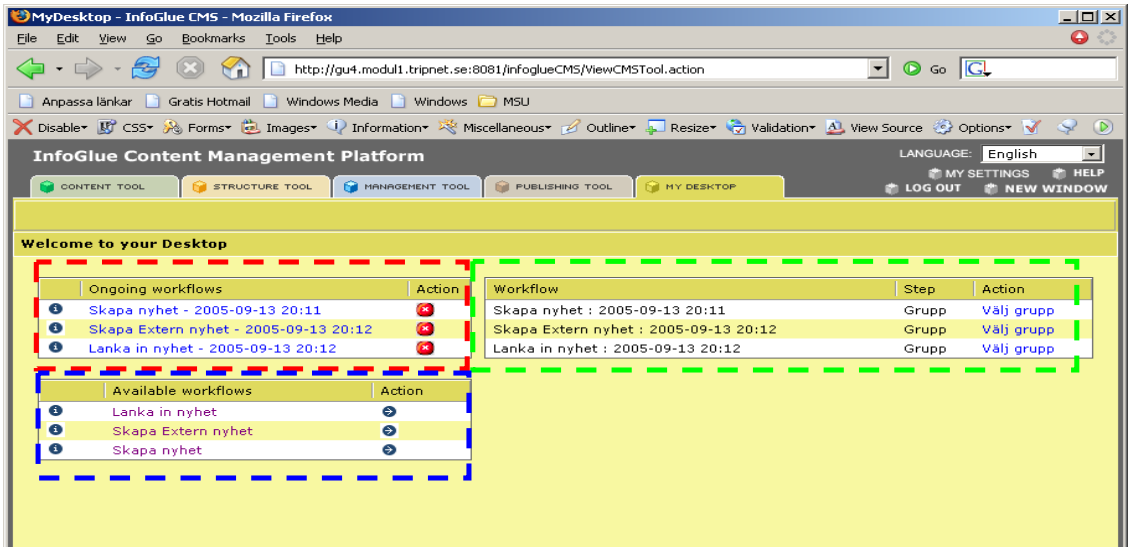
This was the crash course to creating your very first portlet. Now you probably have to read a portlet-book to know what to do next. My guess is that a lot of portlets will be developed for InfoGlue ahead.

Workflows in InfoGlue

This chapter tries to give an introduction to how one creates new workflows.

Introducing the My Desktop Tool

In InfoGlue there is a tool called My Desktop which foremost is the center for workflow activity. The interface looks like this:



The areas are:

Ongoing workflows - Marked with red border. Shows the workflows that are currently active. In 2.1 of InfoGlue one can set up if only to show workflows belonging to one self or all active workflows. It is possible to from the workflows set the name shown there, default is simply the workflow name. Next to the name are buttons representing functions available. For example stopping and aborting the workflow is done there. Left to the name is an information button that let's you see how far the workflow has come.

Available workflows – Marked with blue border. Shows all available workflow definitions that can be started. Left to the workflow name is an information button describing the steps in the workflow.

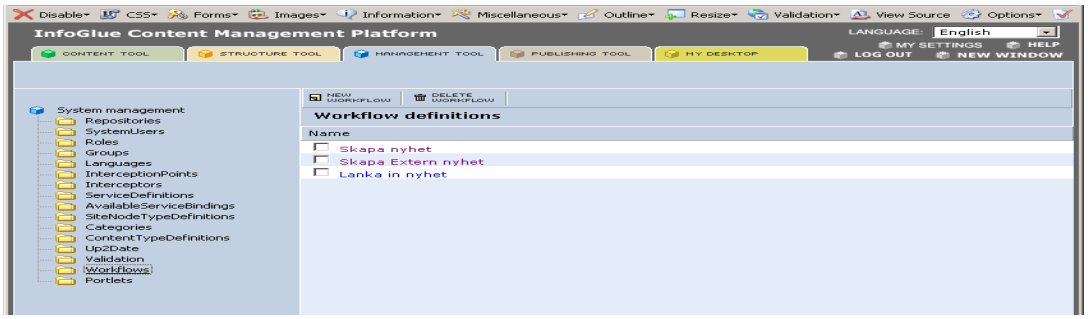
Workflow – the area marked with green. Shows workflow name followed by the step and finally the action waiting to be carried out. Can be filtered in 2.1 so you only see steps you own.

OSWorkflow - The Workflow Engine

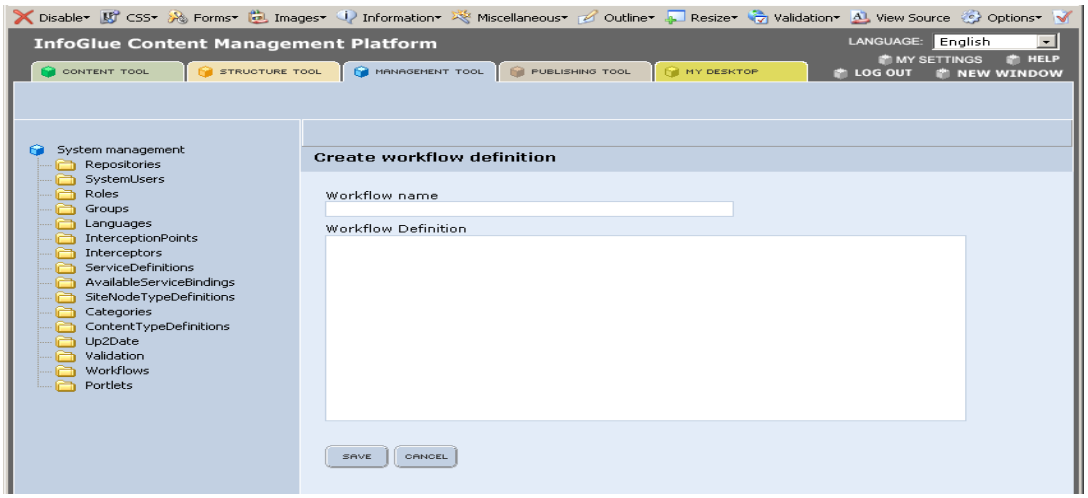
InfoGlue uses the OSWorkflow framework for all workflow handling. You should read the documentation on http://www.opensymphony.com/osworkflow/ before beginning to build workflows.

Managing workflow definitions in InfoGlue

InfoGlue 2.0 introduces a new menu option in the Management tool. It is called Workflows and is where you enter all workflows that should be available in the My Desktop view. It looks like this:



To create a new workflow you press the "New Workflow"-button and end up in the editing view:



Here you enter a workflow definition name – it will be the one visible under the available workflows-list. Below the name in the "Workflow definition" you enter the xml-definition which is the OSWorkflow-instructions really. Just copy/paste it from your favourite XML-tool where you edit it and validated it before.

Creating workflows

Views

Views are where you put the interfaces for the workflow. On each action that should have interaction you put the view attribute which can be a full URL or a relative URL. An example is:

```
<action id="20" name="Editera" view="workflows/gu_link_external_news/main.jsp">
```

Now here is a big design decision – either you can build your views as regular JSP-pages and put it as part of the CMS application. This is the most straight forward way and you get login, session and all InfoGlue API:s for free including a lot of ready made taglibs. Another solution is to build the views as pages in the delivery engine. This approach has its advantages if you have a big need to use texts and images entered in the tools. That is – you get your views content managed. On the other hand that way is not so tested by other projects.

The Property Set

This is by far one of the most important aspects of Workflow authoring. As workflow by definition does not hold state in memory over any longer period of time and those workflows often have a long duration one has limited options on data storing if OSWorkflow is going to keep track of it. OSWorkflow uses something called a Property Set which is basically a Map where you can persist objects with a string as the key identifying the object. The objects must be serializable and large objects are not encouraged to store that way.

The limitations this puts on us makes things like error handling, long transactions and similar issues more difficult and demands for a small framework helping us with all that. InfoGlue has some infrastructure for this.

Actions

As you will read in the OSWorkflow documentation there are prefunctions and results etc. to an action. From the workflow tool the My Desktop will send the user to the view first. When the view actually invokes the action upon user interaction (through the URL given to it) the prefunctions will get called first and then results-parts are evaluated and any post-functions get called last.

Let's look at a small step+action in detail:

```
.
.
<step id="1" name="Grupp">
<actions>
 <action id="11" name="Välj grupp" view="workflows/gu_common/groupSelector.jsp">
  <pre-functions>
    <function type="class">
      <arg name="class.name">com.opensymphony.workflow.util.Caller</arg>
    </function>
    <function type="class">
      <arg name="class.name">
        org.infoglue.cms.applications.workflowtool.function.LanguageProvider
      </arg>
      <arg name="code">sv</arg>
    </function>
    <function type="class">
      <arg name="class.name">
        se.gu.infoglue.cms.applications.workflowtool.function.GroupProvider
      </arg>
    </function>
  </pre-functions>
  <results>
    <result old-status="Finished" status="Queued" step="2" owner="${caller}">
      <conditions>
        <condition type="class">
          <arg name="class.name">
            org.infoglue.cms.applications.workflowtool.condition.HasStatus
          </arg>
          <arg name="status">status.group.ok</arg>
        </condition>
      </conditions>
    </result>
   <unconditional-result old-status="Finished" status="Queued" step="1" owner="${caller}"/>
  </results>
 </action>
</actions>
</step>
.
.
```

Taken a bit from context we can guess by the name of this action that it will present the user with a list of groups he can choose from. We can also see that this action has id 11 and the step has Id 1 and that the action begins with a few pre-functions. We have made them all as classes. The incoming data from the interface will be a group selected if the user has chosen.

The first pre-function is an OSWorkflow standard function which sets the caller as owner of the action – this is standard and should always be used.

The second function is an InfoGlue function which populates the transient vars with a Language-object fetched from InfoGlue's language list with the language code "sv". To be used later on.

The second function is an InfoGlue function which takes the incoming request and checks for the group-selection parameter in it and if found populates the transient vars and also the property set with this information as well as a status.group.ok-status. The status will be status.group.nok if the user forgot to select a group.

The status is then used in the condition which says that if a group was selected we continue to a step with Id 2, otherwise we return to the same step (id 1) and ask the user to pick a group this time.

Function providers, populators and other helper classes

Even though it is possible as pointed out by OSWorkflow to have scripting inside your very XML-definition it soon bloats down your xml making it hard to read. A popular way to structure your workflow then is instead to have small utility functions which does small tasks and puts needed data in the property set or transient vars. InfoGlue 2.1 contains a whole bunch of such functions which are pretty reusable but you can also write your own function which does exactly what you want.

It's pretty easy to write a function and to make sure you benefit from the InfoGlue architecture you extend the InfoglueFunction which if defined by the workflow will give you automatic transaction handling, functions and other nice to have features.

Let's look at a small function which in this case puts a content object in transient vars for the view to use later. This is for example used when a workflow in a previous step creates a working content which will be filled during the workflow but as content objects cannot be stored over time in the workflow itself we only store the id in the property set and this function re-fetches it when we need it.

```
package org.infoglue.cms.applications.workflowtool.function;

import java.util.Map;

import org.infoglue.cms.controllers.kernel.impl.simple.ContentController;
import org.infoglue.cms.entities.content.ContentVO;

import com.opensymphony.module.propertyset.PropertySet;
import com.opensymphony.workflow.WorkflowException;

public class ContentProvider extends InfoglueFunction
{

  public static final String RESULTSET_CONTENT_ID =
ContentPopulator.PROPERTYSET_CONTENT_PREFIX + "contentID";


  protected void doExecute(final Map transientVars, final Map args, final PropertySet ps)
throws WorkflowException {
    populate(transientVars, ps);
  }


  private void populate(final Map transientVars, final PropertySet ps) throws
WorkflowException {
    if(ps.exists(RESULTSET_CONTENT_ID))
      try {
        final Integer contentID = new Integer(ps.getString(RESULTSET_CONTENT_ID));
        final ContentVO content =
ContentController.getContentController().getContentVOWithId(contentID, getDatabase());
        transientVars.put(ContentFunction.CONTENT_PARAMETER, content);
      } catch(Exception e) {
        getLogger().warn("Non-existing contentId found; removing from the resultset.");
        ps.remove(RESULTSET_CONTENT_ID);
      }
  }
}
```

As you see the key to get the wanted contentId from the property set is pretty complex and that is because this function is working together with the function contentPopulator which has previously stored the content and it does not want any other function to overwrite the information by mistake by using the same key. In your functions you will do well in using a similar logic.

Example workflow definition

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE workflow PUBLIC "-//OpenSymphony Group//DTD OSWorkflow 2.7//EN"
"http://www.opensymphony.com/osworkflow/workflow_2_7.dtd">
<workflow>
  <meta name="org.infoglue.title">workflow_title</meta>
  <meta name="org.infoglue.database">db</meta>

  <!-- ================================================================== -->
  <initial-actions>
    <action id="0" name="initial">
      <pre-functions>
        <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.title.DateTitlePopulator
</arg></function>
        <function type="class"><arg
name="class.name">com.opensymphony.workflow.util.Caller</arg></function>
        <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.LanguageProvider</arg><a
rg name="code">sv</arg></function>
        <function type="class"><arg
name="class.name">se.gu.infoglue.cms.applications.workflowtool.function.GroupProvider</arg></f
unction>
        <function type="class">
          <arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.defaultvalue.StringPopul
ator</arg>
          <arg name="name">contentversion_TopNews</arg>
          <arg name="value">0</arg>
        </function>
        <function type="class">
          <arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.defaultvalue.PublishDate
Populator</arg>
        </function>
        <function type="class">
          <arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.defaultvalue.ExpireDateP
opulator</arg>
          <arg name="yearsAhead">1</arg>
        </function>
      </pre-functions>
      <results>
        <result old-status="Finished" status="Queued" step="2" owner="${caller}">
          <conditions>
            <condition type="class">
              <arg
name="class.name">org.infoglue.cms.applications.workflowtool.condition.HasStatus</arg>
              <arg name="status">status.group.ok</arg>
            </condition>
          </conditions>
        </result>
        <unconditional-result old-status="Finished" status="Queued" step="1"
owner="${caller}"/>
      </results>
    </action>
  </initial-actions>
```

```
  <!-- ========================================================================= -->
  <global-actions>
    <action id="1001" name="Radera">
      <meta name="icon">images/mydesktop/stop.gif</meta>
      <meta name="altKey">Radera</meta>
      <results>
        <unconditional-result old-status="Finished" status="Finished" step="5"/>
      </results>
    </action>
  </global-actions>



  <!-- ========================================================================= -->
  <steps>
    <!-- ======================================================================= -->
    <step id="1" name="Grupp">
      <actions>
        <action id="11" name="Välj grupp" view="workflows/gu_common/groupSelector.jsp">
          <pre-functions>
            <function type="class"><arg
name="class.name">com.opensymphony.workflow.util.Caller</arg></function>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.LanguageProvider</arg><a
rg name="code">sv</arg></function>
            <function type="class"><arg
name="class.name">se.gu.infoglue.cms.applications.workflowtool.function.GroupProvider</arg></f
unction>
          </pre-functions>
          <results>
            <result old-status="Finished" status="Queued" step="2" owner="${caller}">
              <conditions>
                <condition type="class">
                  <arg
name="class.name">org.infoglue.cms.applications.workflowtool.condition.HasStatus</arg>
                  <arg name="status">status.group.ok</arg>
                </condition>
              </conditions>
            </result>
            <unconditional-result old-status="Finished" status="Queued" step="1"
owner="${caller}"/>
          </results>
        </action>
      </actions>
    </step>
```

```
    <!-- ================================================================== -->
    <step id="2" name="Editera">
      <actions>
        <action id="20" name="Editera" view="workflows/gu_link_external_news/main.jsp">
          <pre-functions>
            <function type="class"><arg
name="class.name">com.opensymphony.workflow.util.Caller</arg></function>
          </pre-functions>
          <results>
            <unconditional-result old-status="Finished" status="Queued" step="2"
owner="${caller}"/>
          </results>
          <post-functions>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentTypeDefinitionPro
vider</arg><arg name="contentTypeDefinitionName">GUExternNyhet</arg></function>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentPopulator</arg></
function>
          </post-functions>
        </action>
        <action id="21" name="Publicera">
          <pre-functions>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentTypeDefinitionPro
vider</arg><arg name="contentTypeDefinitionName">GUExternNyhet</arg></function>
            <function type="class"><arg
name="class.name">com.opensymphony.workflow.util.Caller</arg></function>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.LanguageProvider</arg><a
rg name="code">sv</arg></function>
            <function type="class"><arg
name="class.name">se.gu.infoglue.cms.applications.workflowtool.function.GroupProvider</arg></f
unction>
            <function type="class"><arg
name="class.name">se.gu.infoglue.cms.applications.workflowtool.function.FolderProvider</arg></
function>
            <function type="class"><arg
name="class.name">se.gu.infoglue.cms.applications.workflowtool.function.GroupCategoryProvider<
/arg></function>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentPopulator</arg></
function>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentProvider</arg></f
unction>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentErrorPopulator</a
rg></function>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentCreator</arg></fu
nction>
            <function type="class">
              <arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.title.ContentVersionTitl
ePopulator</arg>
              <arg name="attributeName">URL</arg>
            </function>
          </pre-functions>
          <results>
            <result old-status="Finished" status="Queued" step="2" owner="${caller}">
              <conditions type="OR">
                <condition type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.condition.HasErrors</arg></condit
ion>
                <condition type="class" negate="true">
                  <arg
name="class.name">org.infoglue.cms.applications.workflowtool.condition.HasStatus</arg>
                  <arg name="status">status.content.ok</arg>
                </condition>
              </conditions>
            </result>
            <unconditional-result old-status="Finished" status="Queued" step="4"
owner="${caller}"/>
          </results>
        </action>
      </actions>
```

```
    </step>



    <!-- ==================================================================== -->
    <step id="4" name="Publicera">
      <actions>
        <action id="40" name="Publicera" auto="true">
          <pre-functions>
            <function type="class"><arg
name="class.name">com.opensymphony.workflow.util.Caller</arg></function>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.LanguageProvider</arg><a
rg name="code">sv</arg></function>
            <function type="class"><arg
name="class.name">se.gu.infoglue.cms.applications.workflowtool.function.GroupProvider</arg></f
unction>
            <function type="class"><arg
name="class.name">se.gu.infoglue.cms.applications.workflowtool.function.FolderProvider</arg></
function>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentProvider</arg></f
unction>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentMover</arg></func
tion>
            <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.ContentPublisher</arg></
function>
          </pre-functions>
          <results>
            <result old-status="Finished" status="Queued" step="2" owner="${caller}">
              <conditions>
                <condition type="class">
                  <arg
name="class.name">org.infoglue.cms.applications.workflowtool.condition.HasStatus</arg>
                  <arg name="status">status.publish.nok</arg>
                </condition>
              </conditions>
            </result>
            <unconditional-result old-status="Finished" status="Queued" step="5"
owner="${caller}"/>
          </results>
        </action>
      </actions>
    </step>



    <!-- ==================================================================== -->
    <step id="5" name="Avsluta">
      <pre-functions>
        <function type="class"><arg
name="class.name">org.infoglue.cms.applications.workflowtool.function.PropertysetCleaner</arg>
</function>
      </pre-functions>
    </step>
  </steps>


</workflow>
```

Example View

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<%@ taglib prefix="c"  uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="iw" uri="infoglue-workflow" %>

<html xmlns="http://www.w3.org/1999/xhtml" lang="sv">
  <head>
    <title>-</title>
    <meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
    <style type="text/css" media="screen">@import url(../gu_common/css/portlet.css);</style>
    <script type="text/javascript" src="../script/workflow.js"></script>
    <script type="text/javascript" src="../../script/listview.js"></script>
    <script type="text/javascript" src="../../script/calendar1.js"></script>
  </head>
  <body onload="resize();">
    <div id="outer">
      <div id="header">L&auml;nka in extern nyhet f&ouml;r <iw:property
key="group_publicGroupName"/></div>
      <div id="main">
        <iw:form>
          <div>
            <iw:hidden idAttr="content_Name" name="content_Name"/>
          </div>
          <script type="text/javascript">setContentName("Extern", "content_Name");</script>

          <table>
            <tr>
              <td><p>URL: *</p></td>
              <td>
                <iw:textfield idAttr="contentversion_URL" name="contentversion_URL"
cssClass="longtextfield"/>
                <iw:property id="error_contentversion_URL" key="error_contentversion_URL"/>
                <c:if test="${error_contentversion_URL != null}"><p class="error"><c:out
value="${error_contentversion_URL}"/></p></c:if>
              </td>
            </tr>
            <tr>
              <td><p>Titel: *</p></td>
              <td>
                <iw:textfield idAttr="contentversion_Title" name="contentversion_Title"
cssClass="longtextfield"/>
                <iw:property id="error_contentversion_Title"
key="error_contentversion_Title"/>
                <c:if test="${error_contentversion_Title != null}"><p class="error"><c:out
value="${error_contentversion_Title}"/></p></c:if>
              </td>
            </tr>
            <tr>
              <td><p>L&auml;nktitel: *</p></td>
              <td>
                <iw:textfield idAttr="contentversion_LinkTitle"
name="contentversion_LinkTitle" cssClass="longtextfield"/>
                <iw:property id="error_contentversion_LinkTitle"
key="error_contentversion_LinkTitle"/>
                <c:if test="${error_contentversion_LinkTitle != null}"><p class="error"><c:out
value="${error_contentversion_LinkTitle}"/></p></c:if>
              </td>
            </tr>
            <tr>
              <td><p>Egen pufftext:</p></td>
              <td>
                <iw:textarea idAttr="contentversion_Leadin" name="contentversion_Leadin"
cssClass="smalltextarea" rows="10" columns="80"/>
                <iw:property id="error_contentversion_Leadin"
key="error_contentversion_Leadin"/>
                <c:if test="${error_contentversion_Leadin != null}"><p class="error"><c:out
value="${error_contentversion_Leadin}"/></p></c:if>
              </td>
            </tr>
            <tr>
              <td><p>Fr&aring;n datum:</p></td>
              <td>
                <iw:textfield idAttr="content_PublishDateTime" name="content_PublishDateTime"
cssClass="dateField" readonly="readonly"/>
```

```
                <a name="calendar" onclick="showCalendar(event, '<%= request.getContextPath()
%>', 'content_PublishDateTime')"><img src="../../images/calendar.gif" class="calendar"
alt="kalender"/></a>
                <iw:property id="error_content_publishDateTime"
key="error_content_publishDateTime"/>
                <c:if test="${error_content_publishDateTime != null}"><p class="error"><c:out
value="${error_content_publishDateTime}"/></p></c:if>
              </td>
            </tr>
            <tr>
              <td><p>Till datum:</p></td>
              <td>
                <iw:textfield idAttr="content_ExpireDateTime" name="content_ExpireDateTime"
cssClass="dateField" readonly="readonly"/>
                <a name="calendar" onclick="showCalendar(event, '<%= request.getContextPath()
%>', 'content_ExpireDateTime')"><img src="../../images/calendar.gif" class="calendar"
alt="kalender"/></a>
                <iw:property id="error_content_expireDateTime"
key="error_content_expireDateTime"/>
                <c:if test="${error_content_expireDateTime != null}"><p class="error"><c:out
value="${error_content_expireDateTime}"/></p></c:if>
              </td>
            </tr>
            <tr>
              <td><p>Toppnyhet:</p></td>
              <td>
                <iw:radio idAttr="contentversion_TopNews1" name="contentversion_TopNews"
value="1"/> Ja
                <iw:radio idAttr="contentversion_TopNews2" name="contentversion_TopNews"
value="0"/> Nej
                <iw:property id="error_contentversion_TopNews"
key="error_contentversion_TopNews"/>
                <c:if test="${error_contentversion_TopNews != null}"><p class="error"><c:out
value="${error_contentversion_TopNews}"/></p></c:if>
              </td>
            </tr>
          </table>
          <p>
            <iw:submit value="Publicera" actionID="21"/>
            <iw:submit value="Radera"    actionID="1001"/>
          </p>
        </iw:form>
      </div>
    </div>
  </body>
</html>
```

As you can see there are quite a lot of taglibs which can be used in the view. They offer if used correctly validation, html-generation, form handling and property set access among others. The same thing is probably true in the deliver applications as well but that is less tested.

Common patterns

This chapter is meant to show off some of the common things developers want to do and how they can be implemented in InfoGlue. Remember that this is just examples and that there are probably other solutions to the same problem.

We will not explain the basics of InfoGlue for each example so it's sometimes important you have read the two public tutorials first or somehow know the background anyway. The chapter only shows how to deal with new Component based sites.

Building a basic page

To start building a site the first thing you need is a basic page component which sets the basic html for the site. This base template is most often general enough so it can be reused all over the site even if some pages needs to have a different layout than others. Here below is a small example of a JSP-based one with just three areas – a header, footer and a middle area.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="infoglue-content" prefix="content" %>
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<%@ page contentType="text/html; charset=UTF-8" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Example </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>

<body>
<table width="100%" border="0" cellspacing="2">
  <tr>
    <td colspan="2">Header</td>
  </tr>
  <tr>
    <td valign="top">Middle</td>
  </tr>
  <tr>
    <td height="24" colspan="2">Footer</td>
  </tr>
</table>
</body>
</html>
```

Create a new HTMLTemplate with this and assign it as the base template on a page to see the wonder in action.

How to manage style (CSS)

Managing style is a very important issue when building a site. Most sites currently moves away from using tables and invisible pixels-images which dominated previously built sites and goes toward using Cascading Style Sheets as the new mean of positioning elements on a webpage.

Which way you want to go is your decision but no matter what you will probably want to use CSS one way or another and if you do you can choose between at least two options:

- Inline the CSS-classes in the components.
  This approach is simple but has it's drawbacks as much more data is sent to the client each time and that reuse of CSS-information between templates is harder to achieve. We will not describe how you do this as it not involves any special InfoGlue-operations.

- Link to the CSS-resource.
  Very nice solution as it helps the browser perform better and also separates the CSS nicely from the templates enabling reuse of it from different places.

The rest of this section will outline how you achieve the second solution. It assumes you have a base component containing the html-elements needed to link a css in a standard HTML-fashion.

1. Create a new content of type HTMLTemplate which will be the CSS-template.

2. Create a new site node called "CSS page" or something and bind the new CSS-component as its base component/template. Preview the site node to see that the page delivers valid css-code.

3. Change the new site node's content type to text/css – important for Mozilla-browsers. (Change to Page Cover for this in structure)

4. Now create a property called CSS-page on the CSS-component which binds to a site node (look at the properties section). Assign this binding to the new CSS-page in the structure tool.

5. Write the html for the link in the html. Reference the CSS-link-property.

Here is a simple example of the basic html component with a css link.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<%@ page contentType="text/html; charset=UTF-8" %>

<structure:boundPage id="cssPage" propertyName="CSSPage"/>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Example </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css" media="screen, print">@import url(<c:out value="${cssPage.url}"
escapeXml="false"/>);</style>
</head>

<body>
<table width="100%" border="0" cellspacing="2">
  <tr>
    <td colspan="2">Header</td>
  </tr>
  <tr>
    <td valign="top">Middle</td>
  </tr>
  <tr>
    <td height="24" colspan="2">Footer</td>
  </tr>
</table>
</body>

And the property would be

<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <property name="CSSPage" type="binding" entity="SiteNode" multiple="false"/>
</properties>
```

How to manage JavaScript's

JavaScript are a must in today's more and more dynamic sites. If you want to use JavaScript's one way or another you can choose between at least two options:

- Inline the JavaScript code in the ordinary templates.
  This approach is simple but has it's drawbacks as much more data is sent to the client each time and that reuse of JavaScript-functions between templates is impossible. We will not describe how you do this as it not involves any special InfoGlue-operations.

- Link to the JavaScript-resource
  Very nice solution as it helps the browser perform and also separates the JavaScript functions nicely from the templates enabling reuse of it from different places.

The rest of this section will outline how you achieve the second solution.

1. Create the JavaScript code as an HTMLTemplate.
2. Create a new site node called "JavaScript page" or something and bind the new JavaScript template as its base component/template. Preview the site node to see that the page delivers valid JavaScript-code.
3. Change the new site node's content type to text/JavaScript – important for Mozilla-browsers.
4. Now create a property or an available service binding called Javascript-page which binds to a site node. Assign this binding to the new Javascript-page.
5. Write the html for the link in the html. Reference the JavaScript-link-property.

Here is a simple example of a basic html component with a JavaScript link.

```
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<%@ page contentType="text/html; charset=UTF-8" %>

<structure:boundPage id="javascriptPage" propertyName="javascriptPage"/>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Example </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript" src="<c:out value="${javascriptPage.url}" escapeXml="false"/>"></script>
</head>

<body>
<table width="100%" border="0" cellspacing="2">
  <tr>
    <td colspan="2">Header</td>
  </tr>
  <tr>
    <td valign="top">Middle</td>
  </tr>
  <tr>
    <td height="24" colspan="2">Footer</td>
  </tr>
</table>
</body>

And the property would be

<?xml version="1.0" encoding="UTF-8"?>
<properties>
  <property name="JavascriptPage" type="binding" entity="SiteNode" multiple="false"/>
</properties>
```

Adding slots for dynamic parts

Now we might want to add some more dynamic parts on the page. Perhaps we want a middle right area to be dynamic as well as adding a navigation column and an article surface.

We use the slot-technique and complete the html a bit:

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="infoglue-content" prefix="content" %>
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<%@ page contentType="text/html; charset=UTF-8" %>

<page:pageContext id="pc"/>

<structure:boundPage id="cssPage" propertyName="CSS page"/>
<structure:boundPage id="javascriptPage" propertyName="Javascript page"/>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title> Example </title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript" src="<c:out value="${javascriptPage.url}"
escapeXml="false"/>"></script>
<style type="text/css" media="screen, print">@import url(<c:out value="${cssPage.url}"
escapeXml="false"/>);</style>
</head>

<body>
<table width="100%" border="1" cellspacing="2">
  <tr>
    <td colspan="3">Header</td>
  </tr>
  <tr>
    <td valign="top">Meny column</td>
    <td valign="top">Article column</td>
    <td valign="top">AA<ig:slot id="right"></ig:slot></td>
  </tr>
  <tr>
    <td colspan="3">Footer</td>
  </tr>
</table>
</body>
</html>
```

Now the right side column should have the text “Right click to add component” so you can add components there. Feel free to make some example component and insert it there. We will add things there later on.

Building a basic content presentation component

When it comes to presenting content on the site you typically have the content taglibs to your aid if you use JSP. Check them out in the appendix.

JSP style

```
<%@ taglib uri="infoglue-content" prefix="content" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<content:contentAttribute id="title" propertyName="Article" attributeName="Title"/>
<content:contentAttribute id="fullText" propertyName="Article" attributeName="FullText"/>

<h1><c:out value="${title}"/></h1>
<p><c:out value="${fullText}"/></p>

And the properties would for this simple article be:

<?xml version='1.0' encoding='UTF-8'?><properties><property name='Article' type='binding'
entity='Content' multiple='false' allowedContentTypeDefinitionNames='Article' description='The
article this component should show'/></properties>
```

Velocity style

```
#set($cl = $templateLogic.componentLogic)
#set($title = $cl.getContentAttribute("Article", "Title"))
#set($fullText = $cl.getParsedContentAttribute("Article", "FullText"))

<h1>$title</h1>
<p>$fullText</p>

And the properties would for this simple article be:

<?xml version='1.0' encoding='UTF-8'?><properties><property name='Article' type='binding'
entity='Content' multiple='false' allowedContentTypeDefinitionNames='Article' description='The
article this component should show'/></properties>
```

Building a basic navigational component

When it comes to navigational components the JSP coder has a couple of nice tags to his/her disposal in the structure-taglib. Look at the appendix last in this paper.

JSP style

```
<%@ taglib uri="infoglue-content" prefix="content" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<structure:childPages id="childPages" propertyName="StartPage"/>
<table border="0" cellpadding="0" cellspacing="0" class="verticalNavigationTable">
<c:forEach var="childPage" items="${childPages}" varStatus="count">
<tr>
  <td class="verticalNavigationTD">
     <p>
       <a class="verticalNavigationLink" href="$page.url"><c:out
value="${page.navigationTitle}"/></a>
     </p>
  </td>
</tr>
</c:forEach>
</table>

And the properties would for this simple article be:

<?xml version='1.0' encoding='UTF-8'?>
  <properties><property name='StartPage' type='binding' entity='SiteNode' multiple='false'/>
</properties>
```

Velocity style

```
<table border="0" cellpadding="0" cellspacing="0">
#foreach($webpage in $templateLogic.componentLogic.getChildPages("StartPage"))
<tr>
  <td class="verticalNavigationTD">
    #foreach($childpage in $templateLogic.getChildPages($webpage.siteNodeId,
"publishDateTime", "desc"))
     <p>
       <a class="verticalNavigationLink"
href="$childpage.url">$childpage.navigationTitle</a><br>
       $templateLogic.getContentAttribute($childpage.metaInfoContentId, "Description")
     </p>
     #end
  </td>
</tr>
#end
</table>

And the properties would for this simple navigation be:

<?xml version='1.0' encoding='UTF-8'?>
  <properties><property name='StartPage' type='binding' entity='SiteNode' multiple='false'/>
</properties>
```

Performance tips

This is a hard area to give advice on but in general there are a couple of ground rules.

- Design you website so that dynamic pages are kept to a minimum. They are of course needed and we don't advice against them but don't get used to always turning page cache off as it severely reduces scalability. If you can get by with creating a separate page with your dynamic logic instead of putting everything in one super dynamic component which sends around a lot of parameters that is often justifiable for a high load site.

- InfoGlue 2.8.0 introduces a possibility to set a page cache timeout (in seconds) which is a great middle way for some pages. You can still have pretty high scalability but can have the page updated in regular intervals.

- Use JSP instead of velocity or freemarker. Faster and consumes less memory.

- Never use recursive macros in Velocity. Under load macros in velocity does not seem to be thread safe. Could be better since 2.4.6 which includes Velocity 1.5.

- Make sure what component takes the most time and analyse what in it is expensive. Perhaps it's a badly written component or perhaps we could optimize InfoGlue. Let us know.

- Make sure to use the new component cache available from InfoGlue 2.0 and forward. It let's you cache parts of a page and leave other parts dynamic.

- Check the database so the indexes in InfoGlue are ok. If one index are broken it can affect performance big.

- Set up a load balanced set of live servers or cms-servers. Also make sure you separate the live database and the cms-database.

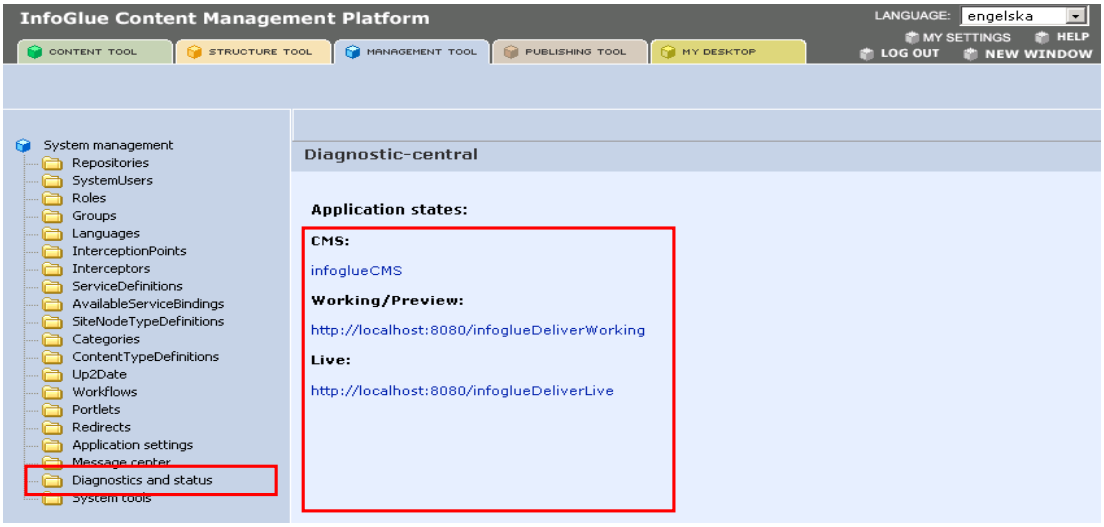
- Use the ViewApplicationState.action view to analyze which components are slow. Focus on those when optimizing. More info in the development manual on that topic.

Quality and Optimization tips

When you are in the last stages of your project you should always assert the quality of the site and the components you have developed. There is a valuable tool which aid you in this work. Part of the information below exists in more a more elaborate version in the administrator manual.

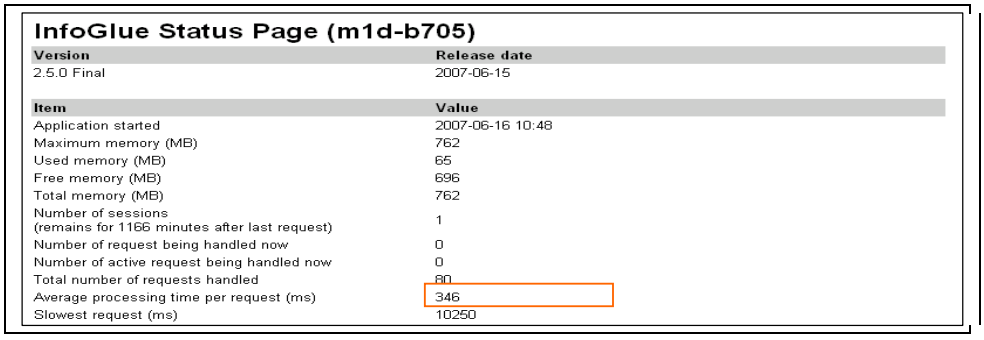
ViewApplicationState

The ViewApplicationState.action can be reached through the management tool and each application has one view.



Each application then has a status view and the sections are described below (opened in new window here):

The first section (shown below) describes some release information, memory information and rough statistics on how the site is performing and how the load is at the moment. This is all live information so reloading this view is often quite useful. Pay close attention to the memory state when doing load tests for example or when you are developing new heavy components. The average processing time is also of great interest during load testing or during production. A low value here but slow site performance for users would for example indicate that there is an issue with your infrastructure or bandwidth.



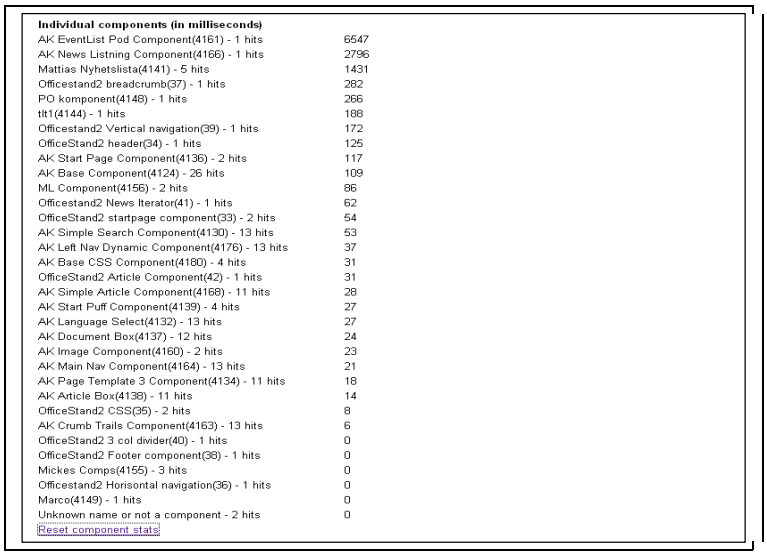
InfoGlue Status Page (m1d-b705)

Version	Release date
2.5.0 Final	2007-06-15

Item	Value
Application started	2007-06-16 10:48
Maximum memory (MB)	762
Used memory (MB)	65
Free memory (MB)	696
Total memory (MB)	762
Number of sessions (remains for 1166 minutes after last request)	1
Number of request being handled now	0
Number of active request being handled now	0
Total number of requests handled	80
Average processing time per request (ms)	346
Slowest request (ms)	10250

The next section (exemplified below) shows a full list of the components rendered since the application was started. It also shows the average processing time for each component and this is really great if you want to optimize your site. In this view it's quite clear that most components are quite fast except a couple in the top which should be

considered first when optimizing. Also consider optimizing components with a lot of hits before seldom used components if they both have slow times.



Individual components (in milliseconds)	
AK EventList Pod Component(4161) - 1 hits	6547
AK News Listning Component(4166) - 1 hits	2796
Mattias Nyhetslista(4141) - 5 hits	1431
Officestand2 breadcrumb(37) - 1 hits	282
PO komponent(4148) - 1 hits	266
tlt1(4144) - 1 hits	188
Officestand2 Vertical navigation(39) - 1 hits	172
OfficeStand2 header(34) - 1 hits	125
AK Start Page Component(4136) - 2 hits	117
AK Base Component(4124) - 26 hits	109
ML Component(4156) - 2 hits	86
Officestand2 News Iterator(41) - 1 hits	62
OfficeStand2 startpage component(33) - 2 hits	54
AK Simple Search Component(4130) - 13 hits	53
AK Left Nav Dynamic Component(4176) - 13 hits	37
AK Base CSS Component(4180) - 4 hits	31
OfficeStand2 Article Component(42) - 1 hits	31
AK Simple Article Component(4168) - 11 hits	28
AK Start Puff Component(4139) - 4 hits	27
AK Language Select(4132) - 13 hits	27
AK Document Box(4137) - 12 hits	24
AK Image Component(4160) - 2 hits	23
AK Main Nav Component(4164) - 13 hits	21
AK Page Template 3 Component(4134) - 11 hits	18
AK Article Box(4138) - 11 hits	14
OfficeStand2 CSS(35) - 2 hits	8
AK Crumb Trails Component(4163) - 13 hits	6
OfficeStand2 3 col divider(40) - 1 hits	0
OfficeStand2 Footer component(38) - 1 hits	0
Mickes Comps(4155) - 3 hits	0
Officestand2 Horisontal navigation(36) - 1 hits	0
Marco(4149) - 1 hits	0
Unknown name or not a component - 2 hits	0

Reset component stats

Mail notifications

In InfoGlue 2.4.6 we have added a feature that senses if the system is getting in trouble or if some pages takes to long. This is a very, very important debugging feature and quite often the best way to start debugging any production instabilities. The concept is quite simple. You set up the warning email receiver address in application settings and InfoGlue will then send emails with full thread information when pages takes to long or memory issues arises. Simplifies debugging and error tracking hugely. USE IT!!!

Included 3:rd party taglibs

InfoGlue comes with a huge set of JSP-tags itself and they are described in full in the appendix last in this document. InfoGlue also comes with some 3:rd party taglibs and those are:

Webwork 1:
http://www.opensymphony.com/webwork_old/build/result/web/docs/api/webwork/view/taglib/

They are imported like this:
<%@ taglib uri="webwork" prefix="ww" %>

Jakarta mailer 1.1
http://jakarta.apache.org/taglibs/doc/mailer-doc/intro.html

They are imported like this:
<%@ taglib uri="http://jakarta.apache.org/taglibs/mailer-1.1" prefix="mailer" %>

OpenSymphony OSCache
http://www.opensymphony.com/oscache/wiki/JSP%20Tags.html

They are imported like this:
<%@ taglib uri="http://www.opensymphony.com/oscache" prefix="cache" %>

Yahoo User Interface (YUI)
Since InfoGlue 2.8.0 we have included this wonderful UI-framework. The javascript framework is located under /script/yui and can be used under both deliver and cms application. Hopefully people will share examples on how this is used. Check out this great framework at http://developer.yahoo.com/yui/

JQuery
Since InfoGlue 2.8.0 we have included the JQuery javascript framework as well as some of the plugins it offers. The javascript framework is located under /script/jquery and can be used under both deliver and cms application. Hopefully people will share examples on how this is used. Check it out at http://www.jquery.com

Appendix A – Taglib reference

For the InfoGlue platform

Page Tags

Description
These tags are made to give you access to context information for the page currently rendered.

Location
Located in infoglue-page

Available tags and short description

Tag	Description
pageContext	This tag will get you the TemplateController-object ($templateLogic in velocity) in a nice way. That object has a lot of usefull API:s not offered by the tags.
deliveryContext	This tag lets you get information about the current rendering process but also let's you set some parameters affecting the rendering process.
clearCache	This tag lets you clear deliver caches – experimental for now – use on own risk.
htmlHeadItem	This tag is great for adding HTML-head sections to the resulting page.
pageAttribute	Great way to share values between components.
httpHeader	This tag is great for letting developers add response headers in a cache-neutral way.
editOnSightMenu	A tag which allows you to expose many great features as a popup menu based on a button or link etc.

pageContext

Description

This tag will get you the TemplateController-object ($templateLogic in velocity). That object has a lot of usefull API:s not offered by the tags. Check out the org.infoglue.deliver.controllers.kernel.impl.simple.BasicTemplateController-class for more information.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.

ID-result variable contains:
An org.infoglue.deliver.controllers.kernel.impl.simple.BasicTemplateController-object.

Examples
The following example gets the BasicTemplateController-object(same as $templateLogic) and puts it in the variable "pc". Then we can call any method it has. Look at the javadoc for that class for more information on what methods are available.

```
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:pageContext id="pc"/>

<c:set var="availableLanguages" value="${pc.availableLanguages}"/>
The variable "availableLanguages" now contains a list of allowed languages for the current page.

<content:contentAttribute id="responsibleName"
contentId="${pc.metaInformationContentId}"
attributeName="responsibleName"/>

This shows another call which gets the meta info content id for the page so we can extract non standard attributes from it if we added our own.
```

deliveryContext

Description

This tag lets you get information about the current rendering process but also let's you set some parameters affecting the rendering process.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
useFullUrl	false	false	Object/String	Sets if the rendering engine should generate full urls or not. If true it will contain http://myserver.. etc and not relative urls.
disablePageCache	false	false	Object/String	The property allows you to disable page cache all together so the result are not cached.
disableNiceUri	true	false	Object/String	Allows you to disable Nice URI:s in the generation process.
trimResponse	false	false	boolean	Makes it possible to dictate for the delivery engine from a component that a page should be trimmed before sent to the client. This means components returning xml as a result don't have to be written on one line to avoid invalid xml any more.
evaluateFullPage	false	true	boolean	Makes it possible to dictate for the delivery engine from a component that a page should not be evaluated a final time by velocity after all components has been evaluated. Improves performance but can lead to different behaviour in your sites compared to now.
contentType	false		String	Makes it possible to dictate for the delivery engine from a component what content-type the page response header should report to the browser.

ID-result variable contains:

The org.infoglue.deliver.applications.databeans.DeliveryContext-object.

Examples

The following example gets the deliveryContext and in the process disables Nice URI:s. Then we can query the object for information and you can look at the DeliveryContext-object in the API for more information.

```
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:deliveryContext id="deliveryContext" disableNiceUri="true"/>

This page has the siteNodeId <c:out value="${deliveryContext.siteNodeId}"/> and the contentType <c:out value="${deliveryContext.contentType}"/> and the pagePath <c:out value="${deliveryContext.pagePath}"/>
```

clearCache

Description

This tag lets you get information about the current rendering process but also let's you set some parameters affecting the rendering process.

Parameters

Name	Required	Default	Type	Description
entity	true		Object/String	The entity name to clear.
entityId	true		Object/String	The entityId to clear.

Examples

The following example clears all caches affected by editing of the content entity with id 54.

```
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:clearCache entity="
org.infoglue.cms.entities.content.impl.simple.ContentImpl" entityId="54"/>
```

httpHeader

Description

This tag lets you add response headers in a controlled and cache-safe way.

Parameters

Name	Required	Default	Type	Description
name	true		Object/String	The name of the header to set.
value	true		Object/String	The value of the header to set.

Examples

The following example sets a response header "Cache-control".

```
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:httpHeader name="Cache-control" value="no-cache"/>
```

pageAttribute

Description

This tag let's you share and use shared data between components globally on a page. Data can be anything but keep it simple. This is an essential part of the new preprocessing mechanism. If no value is given the tag assumes you wish to read the value of that attribute and returns it. Otherwise it save the value.

Parameters

Name	Required	Default	Type	Description
name	true		Object/String	The name of the variable/attribute.
value	false		Object/String	The value to save in the attribute.

Examples

The following example shows how we first store a value and then reads it again. If you read the description of the new preprocessing mechanism you can share it between templates. Otherwise it's very dependent on rendering order.

```
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:pageAttribute name="dummyVar" value="Testing"/>

The variable is <page:pageAttribute name="dummyVar”/>
```

htmlHeadItem

Description

This tag is great for adding HTML-head sections to the resulting page. The problem with components are that they normally had no way of putting html-tags belonging in the head-part but this one does. A very common example is a component which has it's own css or javascript references but if you just print it in the component itself the HTML will not validate in the W3C-validator as some tags are only allowed in the head-block.

Parameters

Name	Required	Default	Type	Description
value	true		Object/String	The string to insert in the page HTML head-section.

Examples

The following example shows a situation where we have a RSS-page we want to reference by a link-tag. The code is perhaps part of a component which list news but also wants to channel it as RSS. So to be able to keep the RSS-link-generation in the right component we use the new tag for this. The page validates and the component is kept in charge of what happens.

```
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<structure:pageUrl id="rssPageUrl" propertyName="RSSPage"/>

<c:set var="rssLinkTag"><link rel="alternate" type="application/rss+xml"
title="My RSS" escapeXml="false"/>" href="<c:out
value="${rssPageUrl}"/>"/></c:set>

<page:htmlHeadItem value="${rssLinkTag}"/>
```

editOnSightMenu

Description

A very flexible way of adding a customized edit on sight menu to a component. Very userfriendly and the interfaces for these items are new with the new upcoming design.

Parameters

Name	Required	Default	Type	Description
id	false		String	The id of the attribute to save the menu under if you don't want to write the markup right away to the page.
html	false		String/Object	The HTML you want should represent the clickable item which then reveals the menu.
showInPublishedMode	false		boolean	Should the menu be shown in published mode as well (default false).
contentId	false		String/Object	States the id of the content the menu should concern when it comes to inline editing etc.
showEditMetaData	false		boolean	Should the menu show edit page meta data item.
showCreateSubpage	false		boolean	Should the menu show the create subpage item.
showEditInline	false		boolean	Should the menu show the edit inline item.
showEditContent	false		boolean	Should the menu show the edit content item.
showCategorizeContent	false		boolean	Should the menu show the categorize content item.
showPublishPage	false		boolean	Should the menu show the publish page item.
showNotifyUserOfPage	false		boolean	Should the menu show the user notification item.
showPageNotifications	false		boolean	Should the menu show the page notifications item.
showContentNotifications	false		boolean	Should the menu show the content notifications item.
showTranslateArticle	false		boolean	Should the menu show the translate article item.
showCreateNewsFromContent	false		boolean	Should the menu show the create new from content item.
showMySettings	false		boolean	Should the menu show the my settings item.

Examples

The following example shows a simple situation where we want to show a button below the article printout which shows a button for the menu.

```
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
….
<h1><c:out value="${title}" escapeXml="false"/></h1>

<c:out value="${fullText}" escapeXml="false"/>

<c:if test="${pc.isInPageComponentMode}">
   <page:editOnSightMenu contentId="${articleContent.id}"/>
</c:if>
```

Content Tags

Description

These tags are made to make life easier for the template developer who wants to extract content one way or another. We think we have covered most cases as we have built numerous sites with these.

Location

Located in infoglue-content

Available tags and short description

Tag	Description
contentAttribute	This tag will get you the value of a named attribute in a content, either bound though a property or referenced by a content id.
assetUrl	This tag will get you the asset url taken from a content, either bound though a property or referenced by a content id.
assetUrls	This tag will get you the asset urls taken from one or many contents bound by a property.
assetUrlFromString	This tag will get you an asset url by storing the string you supply as a file in the asset directory and returning an url to it.
assetThumbnailUrl	This tag will get you the asset url taken from a content, either bound though a property or referenced by a content id. Then it will scale it according to your specs so you get the thumbnail and not the full image.
contentTypeDefinition	Gets a content type definition object by it's name. This can then be used for generating forms etc.
contentDetailPage	A tag capable of finding the url to the most probable endpage for a certain content.
relatedContents	This tag lets you get a contents related contents.
contentVersion	This tag fetches a content version so you can get the properties of that object or get related objects in the object model.
content	This tag fetches a content so you can get the properties of that object or get related objects in the object model.
assignedCategories	This tag fetches the full path of a category. Important tag in followup searches based on assigned categories for example.
boundContents	This tag will get you a list of bound contents bound though a property on a component.
childContents	This tag will get you a list of child contents beneath a content, either bound though a property or referenced by a content id.

Tag	Description
matchingContents	This tag fetches contents based on categorisation etc. That is – it is more of a query-api than a firm binding to contents.
contentSort, sortContentProperty, sortContentVersionProperty, sortContentVersionAttribute	These tags will allow you to sort a collection of contents in a number of ways.
editOnSight	A tag which lets you make a visible representation of the right-click editing found in the structure tool. Easier for the users and can be used on live sites as well.
assignPropertyBinding	A tag which lets you make a visible representation of the assign property editing found in the structure tool on component property bindings. Easier for the users and can be used on live sites as well.
remoteContentService, contentParameter, contentVersionParameter, contentVersionAttributeParameter, digitalAssetParameter, categoryParameter, deleteContent, updateContent, deleteContentVersion, updateContentVersion, deleteDigitalAsset	A collection of tags aimed at performing some CRUD operations on contents in InfoGlue from the deliver engine. Uses a web service API to work against the CMS.
contentExportUrl	A tag capable of exporting a content fully into an importable xml file. Same result as the content export feature found in the content tool but available from templates. We use this in InfoGlue.org to automatically export the latest version of templates on requests.

contentAttribute

Description

This tag will get you the value of a named attribute in a content, either bound though a property or referenced by a content id.

Parameters

Name	Req.	Def.	Type	Description
id	true		String	The result is stored in this variable.
contentId	false		Object/String	The id of the content you wish to get the attribute value from.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the attribute value from.
languageId	false		Object/String	The id of the language you wish to get the attribute in. Default the language the user is browsing the pages in is used.
contentVersion	false		Object/String	The ContentVersion object you want to get the attribute from if you have that available directly.
attributeName	true		Object/String	The name of the attribute you wish to get.
disableEditOnSight	false	false	Object/String	If set to true the component editor will not decorate the attribute for rightclick actions. A must if the attribute is to be inserted in a html attribute.
useAttributeLanguageFallback	false	false	Object/String	If set to true InfoGlue will fallback to the master language version if no attribute was found in the current language.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above. useStructureInheritance overrides the structural inheritance but not the local.
useRepositoryInheritance	false	true	boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
parse	false	false	Object/String	If set to true the article will be parsed and any code in the text will also be executed.
fullBaseUrl	false	false	Object/String	If set to true any references to either assets or pages will be given as full URL:s and not as relative ones.
mapKeyName	false		Object/String	If set the tag assumes that the attribute pointed out by the attribute name is a name-value parameter map and that the attribute we want is found in there.

ID-result variable contains:

The string found in the sought attribute.

Examples

The following example gets the content attribute "Title" from the content bound by the component property "Article".

```
<content:contentAttribute id="title" propertyName="Article" attributeName="Title"/>
<h1><c:out value="${title}"/></h1>
```

The next example does the same but turns off the javascript decoration on it.

```
<content:contentAttribute id="title" propertyName="Article" attributeName="Title"
disableEditOnSight="true"/>
<a href="http://www.infoglue.org" title="<c:out value="${title}"/>"><c:out
value="${title}"/></a>
```

assetUrl

Description

This tag will get you the asset url taken from a content, either bound though a property or referenced by a content id.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
digitalAssetId	false		Object/String	The id of the digitalAsset you wish to get the url to.
contentId	false		Object/String	The id of the content you wish to get the asset url from.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the asset url from.
assetKey	true		Object/String	The key of the asset you wish to get the url to.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
useDownloadAction	false	false	boolean	Sets if the component should use the new download action which can control that the user has access to the asset.

ID-result variable contains:
An string containing the url to the asset.

Examples

The following example gets the url to the asset with asset key "Image" from the content bound by the component property "Article".

```
<content:assetUrl id="imageUrl" propertyName="Article" assetKey="Image"/>

<img src="<c:out value="${imageUrl}"/>"/>
```

assetUrls

Description

This tag will get you the asset urls taken from one or many contents bound by a property.

Parameters

Name	Required	Default	Type	Description
Id	true		String	The result is stored in this variable.
propertyName	true		Object/String	The property name the component used to bind the content by in which you wish to get the asset urls from.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

An Collection containing the urls to the assets.

Examples

The following example gets all assets bound to the multiple asset binding property "Images". The loop just prints the urls and does nothing with them. This is of course just an example – you have to decide what to do with them.

```
<content:assetUrls id="imageUrls" propertyName="Images"
useInheritance="false"/>

<c:forEach var="imageUrl" items="${imageUrls}" varStatus="status">
   <c:out value="${imageUrl}"/>
</c:forEach>
```

assetUrlFromString

Description

This tag will get you an asset url by storing the string you supply as a file in the asset directory and returning an url to it.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
data	true		Object/String	The string you want to get written to file.
fileNamePrefix	true		Object/String	The start of the filename.
fileNameSuffix	true		Object/String	The file suffix.

ID-result variable contains:
An string containing the url to the asset/file.

Examples
The following example writes the string to an asset with the prefix "MyTest" and suffix "txt".

```
<content:assetUrlFromString id="myAssetUrl" data="I just want to test this feature" filePrefix="MyTest" fileSuffix="txt"/>

<img src="<c:out value="${myAssetUrl}"/>"/>
```

assetThumbnailUrl

Description

This tag will get you a url to a thumbnail of a asset if it's an image taken from a content, either bound though a property or referenced by a content id.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
digitalAssetId	false		Object/String	The id of the digitalAsset you wish to get the url to.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the asset thumbnail from.
contentId	false		Object/String	The id of the content by in which you wish to get the asset thumbnail from.
assetKey	true		Object/String	The key of the asset you wish to get the url to.
width	false		Object/String	The width of the thumbnail
height	false		Object/String	The height of the thumbnail
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

An string containing the url to the thumbnail asset.

Examples

The following example gets the url to the asset with asset key "Image" from the content bound by the component property "Article".

```
<content:assetThumbnailUrl id="imageThumbnailUrl" propertyName="Article"
assetKey="Image" width="100" height="100"/>

<img src="<c:out value="${imageThumbnailUrl}"/>"/>
```

assets

Description

This tag will get you the assets available on a content in the current language version, either bound though a property or referenced by a content id.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	false		Object/String	The id of the content you wish to get the assets from.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the assets from.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

An list of DigitalAssetVO-objects.

Examples

The following example gets the assets from the content bound by the component property "Article" and shows the image as well as the asset size.

```
<content:assets id="assets" propertyName="Article"/>

<c:forEach var=”asset” items=”${assets}”>

  <content:assetUrl id="imageUrl" digitalAssetId="${asset.id}"/>

  Asset: <img src="<c:out value="${imageUrl}"/>"/> <c:out
value="${asset.assetFileSize}"/>

</c:forEach>
```

contentTypeDefinition

Description

Gets a content type definition object by it's name. This can then be used for generating forms etc.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	false		Object/String	The id of the content you wish to get the content type of.
contentTypeDefinitionName	false		Object/String	The name of the content type you wish to get.

ID-result variable contains:

An org.infoglue.cms.entities.management.ContentTypeDefinitionVO-object.

Examples

The following example the content type definition with the name "Article". That object (ContentTypeDefinitionVO) contains the schema for that content type. The schema can be parsed with the other tags shown as well.

```
<content:contentTypeDefinition id="ctd" contentTypeDefinitionName="Article"/>

<management:contentTypeDefinitionAttributes
id="contentTypeDefinitionAttributes" schemaValue="${ctd.schemaValue}"/>

<management:contentTypeDefinitionAssets id="contentTypeDefinitionAssets"
schemaValue="${ctd.schemaValue}"/>
```

relatedContents

Description

If you have used the content relation editor on a content attribute and added related contents, to for example an article, this tag lets you get those related contents.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	true		Object/String	The id of the content you wish to get the related contents from.
attributeName	true		Object/String	The name of the attribute containing the related contents.
onlyFirst	false		Object/String	Sets if the tag should only return the first related content. Otherwise a list will allways be returned.

ID-result variable contains:

An list of org.infoglue.cms.entities.content.ContentVO-objects.

Examples

The following example gets a list of contents related to the articleContent on attribute with name RelatedArticles.

```
<content:relatedContents id="relatedArticles"
contentId="${articleContent.contentId}" attributeName="RelatedArticles"/>
```

content

Description

This tag fetches a content object so you can get the properties of that object or get related objects in the object model.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The page to search for the property name on. Very useful to get contents on other pages.
contentId	false		Object/String	If set the tag returns the content object with this contentId.
propertyName	false		Object/String	If set the tag returns the content object bound on this property name.
useInheritance	false		boolean	Sets if the tag should look at inherited properties if the given propertyName was not assigned on the current component.
useRepositoryInheritance	false	true	Boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

A org.infoglue.cms.entities.content.ContentVO-object.

Examples

The following example gets the content object bound to the property "Article". It then gets the original creator and presents his/her name.

```
<content:content id="articleContent" propertyName="Article"
useInheritance="true"/>

<management:principal id="articleCreator"
userName="${articleContent.creatorName}"/>

The article was created by <c:out value="${articleCreator.firstName}"/> <c:out
value="${articleCreator.lastName}"/>
```

contentVersion

Description

This tag fetches the latest content version object so you can get the properties of that object.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentVersionId	false		Object/String	The id of the contentVersion we wish to get. Not common to use.
content	false		Object/String	The content on which to get the last version in the current mode.
languageId	false		Object/String	If set the tag returns looks for a last version matching that laguage. Defaults to the users current site language.
useLanguageFallback	false		Object/String	Sets if the tag should fallback to the master language if it does not find any version in the given language.

ID-result variable contains:

A org.infoglue.cms.entities.content.ContentVersionVO-object.

Examples

The following example gets the latest content version object based on the content object previously fetched. It then gets the latest editor and presents his/her name.

```
<content:contentVersion id="articleContentVersion" content="${articleContent}"/>

<management:principal id="articleModifyer"
contentVersion="${articleContentVersion}"/>

The article was last changed by <c:out value="${articleModifyer.firstName}"/>
<c:out value="${articleModifyer.lastName}"/>
```

contentVersions

Description

This tag fetches the content versions on a content. Not common to use.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
content	false		Object/String	The content on which to get the versions.
contentId	false		Object/String	The id of the content on which to get the versions we wish to get.
languageId	false		Object/String	If set the tag returns looks for a last versions matching that laguage.
includeAllLanguages	false	false	Object/String	Sets if the tag should get all language version or only the versions in the current language.

ID-result variable contains:

A list of org.infoglue.cms.entities.content.ContentVersionVO-object.

Examples

The following example gets the latest content versions based on the content object previously fetched. It then gets the latest editor and presents his/her name.

```
<content:contentVersions id="articleContentVersions"
content="${articleContent}"/>
```

assignedCategories

Description

This tag fetches the assigned categories for a certain content version.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	false		Object/String	The contentId on which to get the last version in the current mode.
propertyName	false		Object/String	If set the tag returns the content object bound on this property name.
languageId	false		Object/String	If set the tag returns looks for a last version matching that laguage. Defaults to the users current site language.
categoryKey	false		Object/String	What category key should be checked.
useAttributeLanguageFallback	false		Object/String	Sets if the tag should fallback to the master language if it does not find any assigned categories in the given language event though a content version exists.
useLanguageFallback	false	true	boolean	Sets if the tag should fallback to the master language if it does not find any version in the given language.
useRepositoryInheritance	false	true	Boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

A org.infoglue.cms.entities.content.ContentCategory-object.

Examples

The following example gets a list of ContentCategory-objects assigned to the latest content version in the current language on the content bound by the property Article. It also prints the list and shows off another tag which gives you the full path.

```
<content:assignedCategories id="categories" propertyName="Article"
categoryKey="Area"/>

<c:forEach var="category" items="${categories}" varStatus="count">
   <c:out value="${category.id}"/>
   <management:categoryPath id="path" categoryId="${category.id}"/>
   path: <c:out value="${path}"/><br>
</c:forEach>
```

boundContents

Description

This tag fetches a list of content objects bound to a certain property.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
propertyName	false		Object/String	If set the tag returns the content object bound on this property name.
useInheritance	false	true	boolean	Sets if the tag should look at inherited properties if the given propertyName was not assigned on the current component.
useRepositoryInheritance	false	true	boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

A org.infoglue.cms.entities.content.ContentVO-object.

Examples

The following example gets the content object bound to the property "Article". It then gets the original creator and presents his/her name.

```
<content:boundContents id="articleContents" propertyName="Article"
useInheritance="true"/>

<c:forEach var="article" items="${articleContents}" varStatus="count">

<content:contentAttribute id="Title" contentId="${article.contentId}"
attributeName="Title"/>

<c:out value="${Title}"/><br/>

</c:forEach>
```

childContents

Description

This tag will get you a list of child contents beneath a content, either bound though a property or referenced by a content id.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	false		Object/String	The id of the content you wish to get the attribute value from.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the attribute value from.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	Boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
searchRecursive	false	false	Object/String	If set to yes up to three levels of recursion will take place so InfoGlue will go though folders as well and add items in them to the list.
sortAttribute	false	NavigationTitle	Object/String	Will decide what attribute to sort the collection of contents on. Any attribute name is ok as well as name, publishDateTime and expireDateTime.
sortOrder	false	ascending	Object/String	'asc' or 'desc' are valid arguments. Just like a sql-orderby.
includeFolders	false	false	Object/String	If set to true the list will also contain folder contents and not just plain contents.

ID-result variable contains:

A list of org.infoglue.cms.entities.content.ContentVO-objects.

Examples

The following example gets all the children to the folder content bound by the component property "ArticleFolder" and prints out the attribute Title on each one of them.

```
<content:childContents id="articles" propertyName="ArticleFolder"/>

<c:forEach var="article" items="${articles}" varStatus="count">

<content:contentAttribute id="Title" contentId="${article.contentId}"
attributeName="Title"/>

<c:out value="${Title}"/><br/>

</c:forEach>
```

matchingContents

Description

This tag fetches contents based on categorisation etc. That is – it is more of a query-api than a firm binding to contents.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentTypeDefinitionNames	false		Object/String	If stated the tag will only return contents of that content type. Comma separated string with names.
categoryCondition	false		Object/String	Here you can state very complex category expressions, including boolean expressions etc. Look at the examples.
freeText	false		Object/String	Here you can if you want a free text search as well.
freeTextAttributeNames	false		Object/String	If you state free tezt search you should also state what attributes to search in.
fromDate	false		Object/Date	This lets you limit the search to only contents with a publishDateTime after this java.util.Date.
toDate	false		Object/Date	This lets you limit the search to only contents with a publishDateTime before this java.util.Date.
repositoryIds	false		Object/String	This lets you limit the search to only contents in certain repositoryId:s (commaseperated id-list).
cacheResult	false	True	Object/Date	This lets you state if you want the matching result to be cached or not.
cacheInterval	false	1800	Object/Date	This lets you set the cache period in seconds.
cacheName	false		Object/Date	This lets you override the cache name.
cacheKey	false		Object/Date	This lets you override the cache key.

ID-result variable contains:

A list of org.infoglue.cms.entities.content.ContentVO-objects.

Examples

The following example gets all articles and news which has the category attribute "Area" categorized as "Medicine".

```
<content:matchingContents id="articles" contentTypeDefinitionNames="Article,News"
categoryCondition="Area=/Areas/Medicine"/>

<c:forEach var="article" items="${articles}" varStatus="count">
  <content:contentAttribute id="Title" contentId="${article.contentId}" attributeName="Title"/>
<c:out value="${Title}"/><br/>

</c:forEach>
```

The following example gets all articles which has the category attribute "Area" categorized as "Medicine" and the category attribute "Department" categorized as "Finance". If you need "OR" - use [] around the categoryCondition instead of {} as here.

```
<content:matchingContents id="articles" contentTypeDefinitionNames="Article,News"
categoryCondition="{Area=/Areas/Medicine,Department=/Departments/Finance}"/>

<c:forEach var="article" items="${articles}" varStatus="count">
  <content:contentAttribute id="Title" contentId="${article.contentId}" attributeName="Title"/>
  <c:out value="${Title}"/><br/>
</c:forEach>
```

contentSort

Description

This tag will allow you to sort a collection of contents in any number of ways.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
input	true		String/Object	The collection to sort
comparatorClass	false		String/Object	Can be used to state a custom comparator-class

ID-result variable contains:

A sorted list of org.infoglue.cms.entities.content.ContentVO-objects.

Examples

The following example gets all the child content to a folder bound by the component property "ArticleFolder" and sorts them first on the Title attribute found in each version and then secondly on the publishDateTime so the come in date order if they have the same Title.

```
<content:childContents id="articles" propertyName="ArticleFolder"/>

<content:contentSort id="sortedArticles" input="${articles}">
  <content:sortContentVersionAttribute name="Title" className="java.lang.String"
ascending="false"/>
  <content:sortContentProperty name="publishDateTime" ascending="false"/>
</content:contentSort>
```

editOnSight

Description

A tag which lets you make a visible representation of the right-click editing found in the structure tool. Easier for the users and can be used on live sites as well.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	true		String	The content the edit on sight should support edit
languageId	true		String	The language the edit on sight should edit
attributeName	true		String	The attribute the edit on sight should mark.
html	true		String	The html you want the user to click on to get the editing form.
showInPublishedMode	false	false	String	This property lets you show the edit-link even on the published site.

ID-result variable contains:

The html-string needed to perform edit on sight.

Examples

The following example creates a link which leads to the editing screen for the content.

```
<content:editOnSight id="editOnSightHTML" contentId="${content.id}"
attributeName="FullText" html="Edit text"/>

<c:out value="${editOnSightHTML}" escapeXml="false"/>
```

assignPropertyBinding

Description

A tag which lets you make a visible representation of the assign property editing found in the structure tool on component property bindings. Easier for the users and can be used on live sites as well.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
propertyName	true		String	The property name we want to assign by the link.
createNew	true		String	Sets if the link should invoke the create wizard first and then assigns the binding.
html	true		String	The html you want the user to click on to get the editing form.
showInPublishedMode	false	false	String	This property lets you show the edit-link even on the published site.

ID-result variable contains:

The html-string needed to perform the action.

Examples

The following example creates two links. One that opens the binding dialog for the property "Article". The second opens the content creation wizard so the user can create a new content which is then assigned to the property "Article".

```
<content:assignPropertyBinding id="assignPropertyBindingHTML"
propertyName="Article" html="Choose article"/>

<content:assignPropertyBinding id="assignPropertyBindingNewHTML"
propertyName="Article" createNew="true" html="Create new article"/>

<c:out value="${assignPropertyBindingHTML}" escapeXml="false"/> |       <c:out
value="${assignPropertyBindingNewHTML}" escapeXml="false"/>
```

remoteContentService

Description

A collection of tags aimed at performing some CRUD operations on contents in InfoGlue from the deliver engine. Uses a web service API to work against the CMS.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
targetEndpointAddress	false			The webservice Url
operationName				What operation should we call - look at the API
principal				What principal is doing the operation – defaults to the current user (anonymous or extranet).

ID-result variable contains:

The result from the webservice – mostly a string with ok / nok.

Examples

This example creates a new Article-content in the CMS under the folder with content id 21(use alternative method like binding perhaps to point out where). It also creates a version with two attributes – Title and FullText in the language the user is currently browsing the site. You can of course make this example much more dynamic but this is a small example at least.

```
<page:deliveryContext id="dc" useFullUrl="true" disableNiceUri="false"/>

<content:contentTypeDefinition id="ctd" contentTypeDefinitionName="Article"/>

<content:remoteContentService id="rcs" operationName="createContents">

  <content:contentParameter name="InfoGlue test" parentContentId="21"
contentTypeDefinitionId="${ctd.id}" repositoryId="1">

    <content:contentVersionParameter languageId="${dc.languageId}">

      <content:contentVersionAttributeParameter name="Title" value="InfoGlue
test"/>
      <content:contentVersionAttributeParameter name="FullText" value="InfoGlue
has now created a content from deliver"/>

      <content:categoryParameter categoryKey="Service categories"
fullCategoryName="/Service areas/Hosting"/>

    </content:contentVersionParameter>

  </content:contentParameter>

</content:remoteContentService>
```

remoteContentVersion

Description

A tag capable to fetch the latest active content version directly from the CMS no matter what state. It's very practical when it comes to editing contents from the live sites in a replicated environment or when showing info from a non-published version. Uses a web service API to work against the CMS.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	true		String	The content id to fetch from
languageId	true		String	The language to fetch with
targetEndpointAddress	false			The webservice Url
operationName	false			What operation should we call - look at the API

ID-result variable contains:

An org.infoglue.cms.entities.content.ContentVersionVO object.

Examples

Gets the latest version from a content with the id 54 in the language with id 1.

```
<content:remoteContentVersion id="contentVersion" contentId="54"
languageId="1"/>

Title: <content:contentAttribute contentVersion="${contentVersion}"
attributeName="Title"/>
```

contentExportUrl

Description

A tag capable of exporting a content fully into an importable xml file. Same result as the content export feature found in the content tool but available from templates. We use this in InfoGlue.org to automatically export the latest version of templates on requests.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentIdList	false		String	State this list if you wish to export multiple contents. A list of Integers.
contentId	false		String	State this if you only wish to export one content.
fileNamePrefix	true		String	The prefix of the file.
includeContentTypeDefinitions	false		boolean	State if you want the export to include the content type.
includeCategories	false		boolean	State if you want the export to include the categories in the system.

ID-result variable contains:

An url to the exported file.

Examples

Gets an export of the template-content itself (exporting the component actually).

```
<content:contentExportUrl id="exportUrl"
contentId="${pc.componentLogic.infoGlueComponent.contentId}"
fileNamePrefix="MyTestExport" includeCategories="false"
includeContentTypeDefinitions="false"/>

<a href="<c:out value="${exportUrl}"/>">Download content</a>
```

contentDetailPage

Description

A tag capable of finding the url to the most probable endpage for a certain content. The logic is that the system finds all pages on which the content is bound directly to a component on that page. If it's more than one the user using a special property can choose which is the master. If not defined the first is picked.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		String	State the detail siteNodeId hard.
contentId	false		String	State which content to find the detail page for.
propertyName	false		String	The property name of the binding to the content to find the detail page to.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	Boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
escapeHTML	false		boolean	
hideUnauthorizedPages	false		boolean	
disableValidateBindingOnPage	false		boolean	
disableFallback	false		boolean	

ID-result variable contains:

An url to the exported file.

Examples

Gets an export of the template-content itself (exporting the component actually).

```
<content:contentDetailPage id="detail" propertyName="ContentPageBindingX"/>

detailUrl: <a href="<c:out value="${detail.url}"/>"><c:out
value="${detail.navigationTitle}"/></a>
```

Structure Tags

Description

These tags are made to make life easier for the template developer who wants to with pages or references to pages one way or another. As with content tags most functionality are there.

Location

Located in infoglue-structure

Available tags and short description

Tag	Description
currentPageUrl	This tag returns the current url
pageUrl	Returns a url to an internal infoglue page
pageUrlAfterLanguageChange	Returns a url to the current infoglue page but with another language parameter.
relatedPages	Returns a collection of related pages created in a content attribute through the content relation editor.
boundPage	The tag returns a bound page as a WebPage-object (see API).
boundPages	Returns a collection of bound pages (WebPage-objects).
childPages	Returns a collection of pages (WebPage-objects) which are located directly below the given node.
isCurrentSiteNode	This tag returns if the given siteNode is the same as the user are currenly looking at – nice for navigation.
isSiteNodeParentToCurrentSiteNode	This tag returns if the given siteNode is a parent to the one the user is currenly looking at – nice for navigation.
componentPropertyValue	This tag returns a non-binding property value defined in the component. Mostly used for textfields, textareas etc.
componentPropertyValues	This tag will get you String array consisting of component property values if it is a textfield, textarea or other non binding property. Useful if you for example use the new checkbox property with several options.
hasDefinedProperty	Returns if a component property value was set or not.
childComponents	Returns a list of components contained in the current one.
siteNode	Returns a siteNode object for further operations.
sortPages	This tag is very useful as it can sort a collection of pages in many ways.
hasPageAccess	This tag gets if the user (logged in or default annymous) has access to a certain siteNode.
componentProperties	Returns all the properties all components on the page with the same property name.
siteNodeLanguages	This tag returns all languages available for the current repository minus any disables languages for this page.
pageAsDigitalAssetUrlTag	This method calls an page and stores it as an digitalAsset - that way one can avoid having to serve javascript-files and css-files through InfoGlue. Not suitable for use if you have very dynamic css:es or scripts which includes logic depending on user info etc.. mostly usable if you have a static css or controls it on the pageCache parameters.
siteNodesFromWebPages	Allows easy conversion between a list of WebPage-beans and SiteNodeVO:s which are more suitable for advanced sorting etc.

currentPageUrl

Description
This tag will get you the current page url.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.

ID-result variable contains:
A string representing the url.

Examples
The following example gets the current page url.

```
<structure:currentPageUrl id="currentUrl"/>

The current url is <c:out value="${currentPageUrl}"/>
```

pageUrlAfterLanguageChange

Description

This tag will get you a url to the current infoglue page but with a new language parameter.

Parameters

Name	Required	Default	Type	Description
Id	true		String	The result is stored in this variable.
languageCode	true		Object/String	The languageCode of the language you wish to get a url to.

ID-result variable contains:

A string representing the url.

Examples

The following example gets an url to the current page but in the language with language code "sv" for Swedish and prints out the url in the href-tag. The success of this depends of course on that Swedish is enabled on that site/page.

```
<structure:pageUrlAfterLanguageChange id="languageUrl" languageCode="sv"/>

<a href="<c:out value="${languageUrl}"/>">Read page in swedish</a>
```

pageUrl

Description

This tag will get you a url to a infoglue page.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the attribute value from.
useInheritance	false	true	Boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
siteNodeId	false		Object/String	If stated the siteNodeId will be used when creating the url instead of the page bound on the propertyName.
contentId	false		Object/String	If given the url will include the argument.
languageId	false		Object/String	If given the url will include the language as an argument – thereby overriding the current user language.

ID-result variable contains:

A string representing the url.

Examples

The following example gets an url to the page bound by the component property "Detail page" and prints out the url in the href-tag.

```
<structure:pageUrl id="detailUrl" propertyName="Detail page"/>

<a href="<c:out value="${detailUrl}"/>">Read more</a>
```

relatedPages

Description

If you have used the structure relation editor on a content attribute and added related pages, to for example an article, this tag lets you get those related pages as an collection.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	false		Object/String	The id of the content you wish to get the related contents from.
propertyName	false		Object/String	The name of the property that binds to the content you wish to get the related contents from.
attributeName	true		Object/String	The name of the attribute containing the related contents.

ID-result variable contains:

A list of org.infoglue.deliver.applications.databeans.WebPage-objects.

Examples

The following example gets a list of contents related to the articleContent on attribute with name RelatedArticles.

```
<structure:relatedPages id="relatedPages"
contentId="${articleContent.contentId}" attributeName="RelatedPages"/>
```

boundPage

Description

This tag will get you a webpage object describing a bound infoglue page.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the attribute value from.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

A org.infoglue.deliver.applications.databeans.WebPage-object.

Examples

The following example gets the webpage-object representing a page bound by the component property "Detail page" and prints out the url in the href-tag and some other properties that object can offer.

```
<structure:boundPage id="detailPage" propertyName="Detail page"/>

The page id is: <c:out value="${detailPage.siteNodeId}"/><br/>

<a href="<c:out value="${detailPage.url}"/>"><c:out
value="${detailPage.navigationTitle}"/></a>
```

boundPages

Description

This tag will get you a collection of webpage object describing a bound infoglue page.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the attribute value from.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
escapeHTML	false	false	Object/String	Sets if the tag should escape any i18n-chars to HTML-entities.
hideUnauthorizedPages	false	false	Object/String	Sets if protected pages which the user does not have access to should be shown.

ID-result variable contains:

A list of org.infoglue.deliver.applications.databeans.WebPage-objects.

Examples

The following example gets the list of webpage-objects representing pages bound by the component property "Pages" and prints out the url in the href-tag and some other properties that object can offer.

```
<structure:boundPages id="pages" propertyName="Pages"/>

<c:forEach var="page" items="${pages}" varStatus="count">
  The page id is: <c:out value="${page.siteNodeId}"/><br/>

  <a href="<c:out value="${page.url}"/>"><c:out
value="${page.navigationTitle}"/></a>

</c:forEach>
```

childPages

Description

This tag will get you a list of child pages beneath a page, either bound though a property or referenced by a sitenode id.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the siteNode you wish to get the child pages below.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the child pages below.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
escapeHTML	false	false	Object/String	If set to yes up any attributes fetched will be HTML-enoded so no non-ascii chars are left.
hideUnauthorizedPages	false	false	Object/String	If set to true the pages the user have no access to will not be shown.

ID-result variable contains:

A list of org.infoglue.deliver.applications.databeans.WebPage-objects.

Examples

The following example gets all the children to the page bound by the component property "BasePage" and prints out the NavigationTitle on each one of them.

```
<structure:childPages id="childPages" propertyName="BasePage"/>

<c:forEach var="childPage" items="${childPages}" varStatus="count">

  The page id is: <c:out value="${page.siteNodeId}"/><br/>

  <a href="<c:out value="${page.url}"/>"><c:out
value="${page.navigationTitle}"/></a>

</c:forEach>
```

isCurrentSiteNode

Description

This tag returns if the given siteNode is the same as the user are currenly looking at – nice for navigation.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the siteNode to check.

ID-result variable contains:

True or false.

Examples

The following prints out if the given site node is the one the user is looking at now.

```
<structure:isCurrentSiteNode id="isCurrentSiteNode"
siteNodeId="${mySiteNodeId}"/>

Is the siteNode sent in the same one as the user is currently rendering: <c:out
value="${isCurrentSiteNode}"/>.
```

isSiteNodeParentToCurrentSiteNode

Description

This tag returns if the given siteNode is a parent to the one the user is currenly looking at – nice for navigation.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the siteNode to check.

ID-result variable contains:

True or false.

Examples

```
<structure:isSiteNodeParentToCurrentSiteNode
id="isSiteNodeParentToCurrentSiteNode" siteNodeId="${mySiteNodeId}"/>

Is the siteNode sent in the same one as the user is currently rendering: <c:out
value="${isSiteNodeParentToCurrentSiteNode}"/>.
```

componentPropertyValue

Description

This tag will get you a component property value if it is a textfield, textarea or other non binding property.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the siteNode on which the property is located if not on the current page.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the child pages below.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

A string containing the value.

Examples

The following example gets the component property "Headline" and prints it out.

```
<structure:componentPropertyValue id="headline" propertyName="Headline"/>

<h1><c:out value="${headline}"/></h1>
```

componentPropertyValues

Description

This tag will get you String array consisting of component property values if it is a textfield, textarea or other non binding property. Useful if you for example use the new checkbox property with several options.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the siteNode on which the property is located if not on the current page.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the child pages below.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

A string containing the value.

Examples

The following example gets the component property values of the property named "markets" and prints it out. Use with your own imagination.

```
<structure:componentPropertyValues id="markets" propertyName="Headline"/>

<h1>Markets used by this components (could be)</h1>

<c:forEach var="market" items="${markets}">

<p><c:out value="${market}"/></p>

</c:forEach>
```

hasDefinedProperty

Description

Returns if a component property value was set or not.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the siteNode to check.
languageId	false		Object/String	The id of the language to check for a property.
propertyName	false		Object/String	The property name the component on which to look for a property.
useInheritance	false	true	Object/String	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.

ID-result variable contains:

True or false.

Examples

```
<structure:hasDefinedProperty id="hasDefinedProperty"
siteNodeId="${mySiteNodeId}" propertyName="Article"/>

Is article defined: <c:out value="${hasDefinedProperty}"/>.
```

childComponents

Description

Returns a list of components contained in the current one.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
slotId	false		Object/String	The id of slot to investigate for components.

ID-result variable contains:

A list of org.infoglue.deliver.applications.actions.InfoGlueComponent.

Examples

This example gets how many components are in the center slot.

```
<structure:childComponents id="childComponents" slotId="center"/>

<common:size id="size" list="${childComponents}"/>

Slot center contains <c:out value="${size}"/> components.
```

siteNode

Description

Returns a siteNode object for further operations.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
targetSiteNodeId	false		Object/String	The id of a site node you wish to investigate for the property if you don't want to look for it on the current page.
siteNodeId	false		Object/String	The id of the site node we wish to load – must state either this or propertyName.
propertyName	false		Object/String	The property binding the site node we wish to load – must state either this or siteNodeId.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

A org.infoglue.cms.entities.structure.SiteNodeVO-object.

Examples

This example gets the site node object bound on property "Article".

```
<structure:siteNode id="siteNode" propertyName="Article"/>

SiteNode was <c:out value="${siteNode.name}"/>.
```

sortPages

Description

This tag will sort a collection of pages.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
input	true		Object/String	The list of pages that are to be sorted.
sortProperty	false	NaviigationTitle	Object/String	The attribute name on the page properties to sort the pages on. Can be any of the defined ones you have on your meta info content type as well as name and publishDateTime.
sortOrder	false	asc	Object/String	Set to asc to sort in ascending order and desc to sort in descending order.
numberOrder	false	false	Object/String	Set to true if you know the values are numbers and want them sorted like numbers. Normally they are sorted like strings.
type	false	asc	Object/String	Not used for now
namesInOrderString	false	asc	Object/String	If you supply a hardcoded commaseparated string with the names of the pages that should be in what order that overrides the other sorting strategy for those items.

ID-result variable contains:

A sorted list of org.infoglue.deliver.applications.databeans.WebPage-objects.

Examples

The following example sorts a collection of pages on the NavigationTitle-attribute on the pages meta information.

```
<structure:childPages id="childPages" siteNodeId="${menuBasePageSiteNodeId}"/>
<structure:sortPages id="childPages" input="${childPages}"/>

<c:forEach var="page" items="${childPages}">
  The page id is: <c:out value="${page.siteNodeId}"/><br/>
  <a href="<c:out value="${page.url}"/>"><c:out value="${page.navigationTitle}"/></a>
</c:forEach>
```

The following example is a more advanced one.

```
<structure:childPages id="childPages" siteNodeId="${menuBasePageSiteNodeId}"/>
<structure:sortPages id="childPages" input="${childPages}" sortProperty="SortOrder" sortOrder="asc"
namesInOrderString="Products ,News,Feedback"/>

<c:forEach var="page" items="${childPages}">
  The page id is: <c:out value="${page.siteNodeId}"/><br/>
  <a href="<c:out value="${page.url}"/>"><c:out value="${page.navigationTitle}"/></a>
</c:forEach>
```

hasPageAccess

Description

This tag gets if the user (logged in or default anonymous) has access to the site node sent in.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the site node we to check.
interceptionPointName	false		Object/String	Sets which interception point name to check for. Default SiteNodeVersion.Read.

ID-result variable contains:
True or false.

Examples

This example gets if the user has access to read the child page.

```
<structure:hasPageAccess id="hasPageAccess"
siteNodeId="${childPage.siteNodeId}"/>

Had page access: <c:out value="${hasPageAccess}"/>.
```

pageAccessRights

Description

Fetches the access rights for a certain entity in the system. Very useful for some specific situations. Probably mostly usable for experts in unusual situations.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
interceptionPointName	false		String/Object	The interception point you wish to check for access rights.
siteNodeId	false		String/Object	State which site node to check.

ID-result variable contains:

A list of AccessRightVO.

Examples

This example get's all access rights for the current page and print out the roles/groups/users for it.

```
<structure:pageAccessRights id="accessRightsVOList"/>

<h1>Access rights</h1>

<c:forEach var="accessRightsVO" items="${accessRightsVOList}">
  <c:out value="${accessRightsVO}"/>

  <c:forEach var="role" items="${accessRightsVO.roles}">
    role: <c:out value="${role.roleName}"/><br/>
  </c:forEach>

  <c:forEach var="group" items="${accessRightsVO.groups}">
    group: <c:out value="${group.groupName}"/><br/>
  </c:forEach>

  <c:forEach var="user" items="${accessRightsVO.users}">
    user: <c:out value="${user.userName}"/><br/>
  </c:forEach>

</c:forEach>
```

componentProperties

Description

Returns all the properties all components on the page with the same property name.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the site node we are to check for properties.
propertyName	true		Object/String	Sets which property to check for bindings.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.

ID-result variable contains:

A list of Map-objects which contains information as elements in them like name, path, type and bindings.

Examples

This example gets all articles on a page and then loops them and prints info on each bound content.

```
<structure:componentProperties id="articleProperties"
siteNodeId="${param.originalSiteNodeId}" propertyName="Article"
useInheritance="false"/>

<c:forEach var="articleProperty" items="${articleProperties}" varStatus="count">
   <c:forEach var="articleBinding" items="${articleProperty.bindings}"
varStatus="countBindings">
   <content:content id="articleContent" contentId="${articleBinding}"/>
   <content:contentAttribute id="title" contentId="${articleContent.id}"
attributeName="Titel"/>
   <p><c:out value="${title}" escapeXml="false"/></p>
  </c:forEach>
 </c:if>
</c:forEach>
```

siteNodeLanguages

Description

This tag returns all languages available for the current repository minus any disabled languages for this page.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
siteNodeId	false		Object/String	The id of the siteNode you wish to get the available languages on.

ID-result variable contains:

A list of org org.infoglue.cms.entities.management.LanguageVO-objects.

Examples

The following example gets all the available languages for the current page and prints them. Perhaps using flags would be better later.

```
<structure:siteNodeLanguages id="languages"/>

Allowed languages:

<c:forEach var="language" items="${languages}" varStatus="count">

  <c:out value="${language.name}"/><br/>

</c:forEach>
```

pageAsDigitalAssetUrlTag

Description

This method calls an page and stores it as an digitalAsset - that way one can avoid having to serve javascript-files and css-files through InfoGlue. Not suitable for use if you have very dynamic css:es or scripts which includes logic depending on user info etc.. mostly usable if you have a static css or controls it on the pageCache parameters..

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
propertyName	false		Object/String	The property name the component used to bind the content by in which you wish to get the attribute value from.
useInheritance	false	true	boolean	Sets if the component should look for other articles on other components with the same propertyname, either on the same page or on pages above.
useRepositoryInheritance	false	true	boolean	Sets if the component should use the repository inheritance or not when looking up entities using inheritance.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
siteNodeId	false		Object/String	If stated the siteNodeId will be used when creating the url instead of the page bound on the propertyName.
contentId	false		Object/String	If given the url will include the argument.
languageId	false		Object/String	If given the url will include the language as an argument – thereby overriding the current user language.
fileSuffix	False		Object/String	If given the created asset will get this file-suffix.

ID-result variable contains:
A string representing the url.

Examples

The following example gets an url to the page bound by the component property "CSS page", makes an internal call with the same headers as the user and stores the result as a digital asset which is then returned as a url to the template and we then reference it as usual.

```
<structure:pageAsDigitalAssetUrl id="cssUrl" propertyName="Css page"
fileSuffix="css"/>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Cascading Style Sheets, level 1</TITLE>
<LINK rel="stylesheet" type="text/css" media="screen" href="<c:out
value="${cssUrl}"/>">
</HEAD>
<BODY>
....
....
</BODY>
</HTML>
```

siteNodesFromWebPages

Description

Allows easy conversion between a list of WebPage-beans and SiteNodeVO:s which are more suitable for advanced sorting etc.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
webPages	true		Object/String	A collection of WebPage-objects.

ID-result variable contains:

A Collection of SiteNodeVO-objects usable in other tags etc.

Examples

The following example gets the bound pages.

```
<structure:childPages id="childPages" propertyName="BasePage"/>

<structure: siteNodesFromWebPages id="childSiteNodes"
webPages="${childPages}"/>
```

contentAttribute

Description

This tag will get you the value of a named attribute in a content, either bound though a property or referenced by a content id. The benefits compared to contentAttribute which can also be used for this is that it is less code and that it can be used for inheritance in structure for attributes that are optional.

Parameters

Name	Req.	Def.	Type	Description
Id	false		String	The result is stored in this variable. If not set the result is printed directly to the page.
siteNodeId	false		Object/String	The id of the page your want to get meta info content attribute from.
languageId	false		Object/String	The id of the language you wish to get the attribute in. Default the language the user is browsing the pages in is used.
attributeName	true		Object/String	The name of the attribute you wish to get.
disableEditOnSight	false	false	Object/String	If set to true the component editor will not decorate the attribute for rightclick actions. A must if the attribute is to be inserted in a html attribute.
useAttributeLanguageFallback	false	false	Object/String	If set to true InfoGlue will fallback to the master language version if no attribute was found in the current language.
useRepositoryInheritance	false	true	boolean	Sets if the component should use repository inheritance when looking for inherited items.
useStructureInheritance	false	true	boolean	Sets if the component should use inherit properties from it's parent site nodes.
parse	false	false	Object/String	If set to true the article will be parsed and any code in the text will also be executed.
fullBaseUrl	false	false	Object/String	If set to true any references to either assets or pages will be given as full URL:s and not as relative ones.
mapKeyName	false		Object/String	If set the tag assumes that the attribute pointed out by the attribute name is a name-value parameter map and that the attribute we want is found in there.

ID-result variable contains:

The string fetched if id is given.

Examples

The following example gets the content attribute "Title" from the current page and uses it as the page title.

```
<structure:pageAttribute id="title" attributeName="Title"/>

<html><head><title><c:out value="${title}"/></title></head>
<body>…</body>
</html>
```

common Tags

Description

These tags are made to make life easier for the template developer in many common tasks not really bound to InfoGlue.

Location

Located in infoglue-common

Available tags and short description

Tag	Description
Sublist	This tag returns a sublist of a collection
Size	Returns the size of a collection
Slots	Returns a slotted collection – useful for google lists with x items on each page.
URLEncode	UrlEncodes a string sent in.
Encrypt	Encrypts a string according to the DES cipher mode.
Decrypt	Decrypts a string according to the DES cipher mode.
urlBuilder, parameter	A valuable tag for creating urls to almost anything, with or without parameters.
remoteWorkflowService	Let's a user invoke a workflow remote.
Include	A tag which lets you include a template inside this component without it being added dynamically. Very good for more strict layouts.
Import	A tag which lets you import an url just like c:import but allows you to specify timeouts.
cropText	Crops a string and can add a suffix etc.
setCookie	Sets a cookie
getCookie	Gets a cookie
transformText	This tag transforms a text in a number of ways.
rssFeed, rssFeedEntry	Creates a rss-feed in almost any format there is.
textRender	This tag creates and return the url of a image rendered with a string. Pretty advanced and good for menues etc.
parseMultipart	This tag parses a file upload.
Diff	This tag compares two texts.
XSLTransform	Transforms xml with xslt just like x:transform but with saxon.
XSLTransformParameter	Parameter tag to the XSLTransform-tag.
documentConverter	Experimental document conversion tag which operates together with JODConverter (Open Office serverside). It can take a worddocument and convert it on the fly to pdf, odf and html. The tag also manages to create table of contents separately so you can have navigation separate.
mail	This is a simple mail-tag. For now it only support simple mails and it gives direct feedback on any server connection problems which the Jakarta mailer tag does not.

subList

Description

This taglib lets you get a subset of the given list. Useful for showing parts of a list. The list could contain almost anything as long as it's a collection.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
list	true		Object/String	The list we want to limit.
startIndex	false	0	Object/String	If you wish to exclude the initial x elements – state this number.
count	false	true	Object/String	How many elements should be included from the startIndex.

ID-result variable contains:
A list containg a sublist of the original list.

Examples
This will result in that the variable "sublist" will contain a list which contains the first 5 elements of the original list.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="infoglue-content" prefix="content" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:pageContext id="pc"/>

<structure:childPages id="childPages" siteNodeId="${pc.siteNodeId}"/>

<common:sublist id="sublist" list="${childPages}" count="5"/>
```

You can also add a starting index like this below:

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="infoglue-content" prefix="content" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:pageContext id="pc"/>

<structure:childPages id="childPages" siteNodeId="${pc.siteNodeId}"/>

<common:sublist id="sublist" list="${childPages}" startIndex="1" count="3"/>
```

This will result in that the variable "sublist" will contain a list which contains the elements 2-4.

size

Description

This taglib lets you get size of the given list. JSP is rather limited in that respect strangely enough.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
list	true		Object/String	The list we want to limit.

ID-result variable contains:

An integer.

Examples

This will result in that the variable "count" will contain the size of the childPages.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="infoglue-content" prefix="content" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:pageContext id="pc"/>

<structure:childPages id="childPages" siteNodeId="${pc.siteNodeId}"/>

<common:size id="size" list="${childPages}"/>

Size: <c:out value="${size}"/><br/>
```

slots

Description

This taglib lets you divide the list into slots. Useful for having a archive or a search list divided in smaller pages.

Parameters

Name	Required	Default	Type	Description
visibleElementsId	true		String	This variable will contain a list of elements that are to be shown.
visibleSlotsId	true		Object/String	This variable will contain a list of slots to show.
lastSlotId	true		Object/String	The variable will contain which slot is the last one in the result.
elements	true		Object/String	The list we want to divide in slots.
maxSlots	false		Object/String	At what number of slots should we stop.
currentSlot	false		Object/String	The variable the current slot will be stored in.
slotSize	true		Object/String	How many items should be in each slot.
slotCount	true		Object/String	How many slots should be visible.

ID-result variable contains:

A org.infoglue.deliver.util.Slots-object.

Examples

This will result in a number of variables being set with information and sublists.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="infoglue-content" prefix="content" %>
<%@ taglib uri="infoglue-structure" prefix="structure" %>
<%@ taglib uri="infoglue-page" prefix="page" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<page:pageContext id="pc"/>

<content:matchingContents id="unsortedNewsItems"
contentTypeDefinitionNames="Article"/>

<c:set var="currentSlot" value="${param.currentSlot}"/>
<c:if test="${currentSlot == null}">
        <c:set var="currentSlot" value="1"/>
</c:if>

<common:slots visibleElementsId="newsItems" visibleSlotsId="indices"
lastSlotId="lastSlot" elements="${unsortedNewsItems}"
currentSlot="${currentSlot}" slotSize="10" slotCount="10"/>
```

URLEncode

Description

This taglib lets encode a string according to the URLEncoding-scheme. Useful for sending parameters in url:s which contain special characters etc.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
value	true		Object/String	The string we want to encode.
encoding	false	UTF-8	Object/String	The encoding you wish to encode it by.

ID-result variable contains:

A string-object.

Examples

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:URLEncode id="encodedString" value="This is a string we want to
encode" encoding="UTF-8"/>

Encoded string: <c:out value="${encodedString}"/><br/>
```

This will result in that the variable "encodedString" will contain an URLEncoded string ("This+is+a+string+we+want+to+encode").

encrypt / decrypt

Description

These tags lets you encrypt / decrypt a string. Very useful for hiding information from the user and still being able to transfer it between pages etc. We use it heavily in contact forms so email addresses are hidden.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
value	true		Object/String	The string we want to encode.

ID-result variable contains:
A String-object.

Examples

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:encrypt id="encryptedString" value="This is a string we wish to
encrypt"/>

Encrypted string: <c:out value="${encryptedString}"/><br/>
```

This will result in that the variable "encryptedString" will contain an MD5-encrypted string.

Below is an example of how to decrypt it. The server keeps the encryption key to itself.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:decrypt id="decryptedString" value="${encryptedString}"/>

Decrypted string: <c:out value="${decryptedString}"/><br/>
```

This will result in that the variable "decryptedString" will contain the cleartext string again.

urlBuilder

Description

This tag creates urls to pages in InfoGlue or outside.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
baseURL	false		Object/String	If stated you can add your own base to the url. Http://mysite... not commonly used.
query	false		Object/String	Used if you wish to add your own query string
excludedQueryStringParameters	false		Object/String	Used to exclude query string parameters, comma separated string
fullBaseUrl	false		Object/String	State this if you wish the url to be absolute.
disableNiceURI	false		Object/String	State this if you wish to disable NiceUrl for this url.

ID-result variable contains:

A String-object.

Examples

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:urlBuilder id="currentUrl" fullBaseUrl="true"/>
```

This results in the current full url.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:urlBuilder id="url" excludedQueryStringParameters="apa,bepa">
        <common:parameter name="extraparameter1" value="foo"/>
        <common:parameter name="extraparameter2" value="bar"/>
</common:urlBuilder>
```

This results in the current full url without the parameters apa and bepa but with two new parameters.

include

Description

This tag makes it possible to include a template in another template, that way modularizing a page without making it dynamic. Good in some cases where users don't want so much flexibility.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
contentId	false		Object/String	The content id in which the related templates lies.
relationAttributeName	false		Object/String	What attribute contains the related contents.
contentName	false		Object/String	In the collection of related template contents – what is the name of the content you wish to include.
useAttributeLanguageFallback	false	true	Object/String	States if language fallback should be used.

ID-result variable contains:

The rendered result as a String-object.

Examples

This results in that the template / content called Article related to the current component on relation attribute "RelatedComponents" is included on the page as a template.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:include relationAttributeName="RelatedComponents"
contentName="Article"/>
```

import

Description

This tag is almost identical to c:import but allows you to specify a timeout.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
url	false		Object/String	The url to import.
charEncoding	false		Object/String	Char encoding of the input data.
timeout	false		Object/String	Sets after how long (milliseconds) the tag should return with error.

ID-result variable contains:

The rendered result as a String-object.

Examples

Read the description on c:import – mostly no need for more examples. The only addition we have made is that it is possible to set request the request timeout.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:import id="result" url="http://www.google.com/search">
  <common:parameter name="q" value="infoglue"/>
</common:import>
```

cropText

This tag will crop a text sent in – very useful for cropping a long texts into short introductions showing only part of the text.

Parameters

Name	Required	Default	Type	Description
id	false		String	if specified the result is stored in this variable – otherwise the cropped text will be written to the page directly.
text	true		Object/String	The text you wish to crop.
startIndex	false	0	Object/String	If stated the tag will remove all chars before this index.
maxLength	false		Object/String	If stated the tag will remove all chars after the specified number of chars.
suffix	false	"..."	Object/String	Will add the suffix after the cropped text to indicate that there are more text which was removed.
adjustForEntities	false	true	Object/String	If the text contains HTML-entities it will affect the string length – set to false if you do not wish the tag to account for this (not recommended).

ID-result variable contains:

A String-object.

Examples

The following example crops the text and outputs it to the page.

```
<common:cropText id="croppedText" text="This is a text which we want to crop"
maxLength="10"/>
```

The result would be "This is a text which..." (without the qutoes). Below is an example of a more customized example where we use an existing variable as input instead and also state our own suffix.

```
<common:cropText id="croppedText" text="${myTextVar}" maxLength="20"
suffix="&raquo;"/>
```

The result would be "This is a text which »" (without the qutoes).

setCookie / getCookie

These taglibs lets you set / get a cookie.

Parameters

Name	Required	Default	Type	Description
id	false		String	if specified the result is stored in this variable – otherwise the cropped text will be written to the page directly.
text	true		Object/String	The text you wish to crop.
startIndex	false	0	Object/String	If stated the tag will remove all chars before this index.
maxLength	false		Object/String	If stated the tag will remove all chars after the specified number of chars.
suffix	false	"..."	Object/String	Will add the suffix after the cropped text to indicate that there are more text which was removed.
adjustForEntities	false	true	Object/String	If the text contains HTML-entities it will affect the string length – set to false if you do not wish the tag to account for this (not recommended).

ID-result variable contains:

A String-object.

Examples

This below will result that the cookie "foo" will get the value "bar" but it expires when you close the browser.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:setCookie name="foo" value="bar" domain=".infoglue.com" path="/"
maxAge="-1"/>
```

This below will result that the cookie "foo" will get the value "bar" which expires on the date you give it.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:setCookie name="foo" value="bar" domain=".infoglue.com" path="/"
maxAge="${expireDateInMilliSeconds}"/>
```

This will result in a new context value being set with the value of the cookie "foo".

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:getCookie id="fooCookieValue" name="foo"/>
```

transformText

These taglibs lets you set / get a cookie.

Parameters

Name	Req.	Type	Description
Id	true	String	The id of the variable to store the result in.
text	true	Object/String	The text to transform
htmlEncode	false	Object/String	Should the text be HTMLEncoded – all non ascii chars are replaced by html-entities.
replaceLineBreaks	false	Object/String	Should the tag replace linebreak chars?
lineBreakReplacer	false	Object/String	What string should replace the linebreak chars?
replaceString	false	Object/String	State this if you wish to replace a string in the text with another string. Can be a regular expression.
replaceWithString	false	Object/String	The new string
prefix	false	Object/String	Set this attribute if you wish to insert a string before the text.
addPrefixIfTextMatches	false	Object/String	This is an optional attribute to the prefix attribute. Let's you state that the prefix only be added if the text contains/matches a regular expression you state here.
addPrefixIfTextNotMatches	false	Object/String	This is an optional attribute to the prefix attribute. Let's you state that the prefix only be added if the text does not contains/matches a regular expression you state.
suffix	false	Object/String	Set this attribute if you wish to insert a string after the text.
addSuffixIfTextMatches	false	Object/String	This is an optional attribute to the suffix attribute. Let's you state that the suffix only be added if the text contains/matches a regular expression you state here.
addSuffixIfTextNotMatches	false	Object/String	This is an optional attribute to the suffix attribute. Let's you state that the suffix only be added if the text does not contains/matches a regular expression you state.

ID-result variable contains:
A String-object.

Examples

This example will output "Testing replacing all spaces" into the document.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:transformText id="transformedText" text="Testing replacing all spaces" replaceString="\\s"
replaceWithString="&amp;nbsp;"/>
<c:out value="${transformedText}"/>
```

This example will output the string html encoded and all line breaks would be replaced with
-tags into the document.

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:transformText id="transformedText" text="Testing replacing & i18N like ääö ÅÄÖ"
replaceString="\\s" replaceWithString="&amp;nbsp;" htmlEncode="true" replaceLineBreaks="true"
lineBreakReplacer="<br/>"/>
<c:out value="${transformedText}"/>
```

rssFeed

This is a nice taglib which lets you give users RSS-feeds from your site. We use Rome backend to output the format you wish to use.

Parameters

Name	Required	Default	Type	Description
id	false		String	The id of the variable to store the feed in.
feedType	true		Object/String	What kind of feed - rss_0.9, rss_0.91, rss_0.92, rss_0.93, rss_0.94, rss_1.0, rss_2.0 atom_0.3
title	false	0	Object/String	The title of the feed
link	false		Object/String	The url to the feed
description	false	"..."	Object/String	A description of the feed.

ID-result variable contains:
A String-object containing the RSS-XML.

Examples

This example will return an rss-xml. **OBS - Unfortunaltely you have to remove all linebreaks and spaces in this example so the whole thing is one one line otherwise the xml will be faulty. In later versions of IG you can avoid this by stating the trimResponse-attribute in the deliverContext tag. The categories below are just fakes to show how you can add atom-categories.**

```
<%@ taglib uri="infoglue-common" prefix="common" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<common:rssFeed id="rss" title="MyFeed" link="http://www.mycomp.com/rss"
description="The best feed" feedType="atom_1.0">
        <content:childContents id="news" propertyName="News folder"
sortAttribute="publishDateTime" sortOrder="desc"/>
        <c:forEach var="newsItem" items="${news}" varStatus="count">
                <content:contentAttribute id="navigationTitle"
contentId="${newsItem.id}" attributeName="${titleName}"/>
                <content:contentAttribute id="leadIn"
contentId="${newsItem.id}" attributeName="${descriptionName}"/>
                <structure:pageUrl id="url" contentId="${newsItem.id}"
propertyName="News detail page"/>
                <common:rssFeedEntry title="${navigationTitle}"
link="${url}" description="${leadIn}"
publishedDate="${newsItem.publishDateTime}">
                  <common:rssFeedEntryCategory taxonomyUri="infoglue"
name="Sports"/>
                  <common:rssFeedEntryCategory taxonomyUri="infoglue"
name="News"/>
                </common:rssFeedEntry>
        </c:forEach>
</common:rssFeed>

<c:out value="${rss}" escapeXml="false"/>
```

textRender

This tag creates a image from a string. I can cope with a lot of parameters.

Parameters

Name	Required	Default	Type	Description
id	true		String	The variable the url to the generated image will be stored in.
propertyName	false		Object/String	If stated the tag looks at this binding for a configuration content which defines all the parameters below in attributes. Nice if you wish to reuse a few rendering profile but you don't want to hard code it in the templates.
contentId	false		Object/String	If stated the tag looks at this content as a configuration content which defines all the parameters below in attributes. Nice if you wish to reuse a few rendering profile but you don't want to hard code it in the templates.
text	true		Object/String	The text we want to render.
imageType	false	BufferedImage.TYPE_4BYTE_ABGR	Object/String	What kind of image do we want – look at BufferedImage.TYPE_XXX for the available types.
fontName	false	Dialog	Object/String	The name of the font to use
fontStyle	false	Font.PLAIN	Object/String	Which style to use
fontSize	false	18	Object/String	The size of the font
fgColor	false	Black	Object/String	What color should the text have – colon separated RGBA-color
bgColor	false	White	Object/String	What color should the background have – colon separated RGBA-color
renderWidth	false	200	Object/String	How wide is the render canvas in pixels
align	false	left	Object/String	How should the text align: left, right or center
padTop	false	4	Object/String	If stated the text will have x pixels of padding above it
padBottom	false	4	Object/String	If stated the text will have x pixels of padding below it
padLeft	false	4	Object/String	If stated the text will have x pixels of padding left of it
padRight	false	4	Object/String	If stated the text will have x pixels of padding right of it
pad	false		Object/String	If stated the text will have x pixels of padding around it
maxRows	false	20	Object/String	State the maximum numbers of rows the tag can extend the rendering to.
trimEdges	false	0	Object/String	Should the edges be trimmed or not. 0 = notrim, 1 = left, 2 = right, 3 = left and right
tileBackgroundImage	false	0	Object/String	State if the background image should be tiled. 0 = no, 1 = horizontal, 2 = vertical, 3 = both
backgroundImageUrl	false		Object/String	The url of the background image to be used.

ID-result variable contains:

A String-object representing the image-url.

Examples

This very simple example just renders a simple text with all the default settings except that we trim both sides and we then show it in a image-tag.

```
<common:textRender id="titleImageUrl" text="InfoGlue is cool" trimEdges="3"/>
<img src="<c:out value="${titleImageUrl}"/>"/>
```

parseMultipart

This tag can handle file upload posts. That is – if such a post is done this tag is needed if you wish to extract both normal parameters and files – the normal http-request object will not do that for you.

Parameters

Name	Required	Default	Type	Description
id	false		String	The result is stored in this variable.
maxSize	true		Object/String	Sets how large file are allowed.
allowedContentTypes	false	0	Object/String	Sets which content types are allowed. Comma seperated string.
ignoreEmpty	false		Object/String	Sets if the file can be left out.

ID-result variable contains:
A java.util.Map-object containing all post information.

Examples

This example parses a form-post including files but only allows files are no bigger than 100.000 bytes and of content type gif, jpg or png. It then gets a form parameter we know are sent in and lists all files posted.

```
<common:parseMultipart id="formParameters" maxSize="100000"
allowedContentTypes="image/gif,image/jpg,image/png" ignoreEmpty="true"/>

The form contained the field company name and the value filled in was: <c:out
value="${formParameters.companyName}"/>

<c:forEach var="fileItem" items="${formParameters.files}">
          Uploaded file: <c:out value="${fileItem.name}"/> had content type
<c:out value="${fileItem.contentType}"/>

</c:forEach>
```

diff

This tag can compare two texts and produce a compare-result.

Parameters

Name	Required	Default	Type	Description
id	false		String	The result is stored in this variable.
originalText	true		Object/String	The original text.
modifiedText	true		Object/String	The modified text.

ID-result variable contains:

A diff-string.

Examples

This example compares two texts and prints the result on the webpage.

```
<c:set var="org" value="The original text"/>
<c:set var="new" value="The new text"/>

<common:diff id="diffString" originalText="${org}" modifiedText="${new}"/>

Result was: <br/>

<c:out value="${diffString}" escapeXml="false"/><br/>
```

documentConverter

Experimental document conversion tag which operates together with JODConverter (Open Office serverside). It can take a worddocument and convert it on the fly to pdf, odf and html. The tag also manages to create table of contents separately so you can have navigation separate. Remember that this tag currently requires open office remote service installed. Read more on http://www.artofsolving.com/opensource/jodconverter.

Parameters

Name	Required	Default	Type	Description
id	false		String	The result is stored in this variable.
docUrl	false		Object/String	Url to the word-document to convert.
docFilePath	false		Object/String	Path to the word document to convert.
title	true		Object/String	Title of the index.
menuTextLength	true		Object/String	Lengths of the index items.
cssList	true		Object/String	The list of css:s to use during conversion.
rewrite	false		Object/String	Allows you to force a rewrite of the document.

ID-result variable contains:

A ConvertedDocumentBean which contains url:s to the pdf/html /odf-versions etc.

XSLTransform

This tag replaces the x:transform tag and use SAXON XSLT 2.0 processor instead which is more efficient, less memory intense and faster. Not to mention 2.0 compliant. The tag incorporates caching of stylesheets etc.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
xmlFile	false		Object/String	The full xml file path.
xmlString	false		Object/String	The xml as a string.
source	false		Object/String	The xml as a source.
styleFile	false		Object/String	The full style file path.
styleString	false		Object/String	The style as a string.
cacheStyle	false	true	boolean	State if the xslt should be cached. Consumes memory so keep down the number of cached templates.
outputFormat	false	string	Object/String	The output format. (document, tinyDocument, string)

ID-result variable contains:

A string (or other if specified) with the result of the transformation.

Examples

This example transforms a xml with the xsl and prints the result on the webpage.

```
<common:XSLTransform id="resultString" xml="${xmlString}"
styleString="${xslString}"/>

Result was: <br/>

<c:out value="${resultString}" escapeXml="false"/><br/>
```

XSLTransformParameter

This tag lets you set parameters to the transformation process if you use the XSLTransform-tag. Just add the parameters as subtags to the XSLTransform-tag.

Parameters

Name	Required	Default	Type	Description
name	true		Object/String	The parameter name.
value	true		Object/String	The parameter value.

ID-result variable contains:
A string (or other if specified) with the result of the transformation.

Examples

This example transforms a xml with the xsl using two parameters in the process and prints the result on the webpage.

```
<common:XSLTransform id="resultString" xml="${xmlString}"
styleString="${xslString}">
  <common:XSLTransformParameter name=”param1” value=”foo”/>
  <common:XSLTransformParameter name=”param2” value=”bar”/>
</common:XSLTransform>

Result was: <br/>

<c:out value="${resultString}" escapeXml="false"/><br/>
```

mail

This is a simple mail-tag. For now it only support simple mails and it gives direct feedback on any server connection problems which the Jakarta mailer tag does not.

Parameters

Name	Required	Default	Type	Description
id	true		Object/String	The result variable name.
from	true		Object/String	The from address.
to	true		Object/String	The to address.
recipients	true		Object/String	The recipients you want to send the mail to as bcc.
subject	true		Object/String	The subject of the mail.
type	false		Object/String	If you want to send it as html or plain.
charset	false		Object/String	The charset you want to send the mail in.
message	true		Object/String	The message to send.

ID-result variable contains:

A boolean containing if the mail was sent successfully or not. If the mail failed to send the extra attribute "commonMailTagException" is set containing the exception.

Examples

This example sends a mail to a dummy person.

```
<common:mail id="success" from="no-reply@infoglue.org" to="no-reply@infoglue.org" recipients="dummy@mycomp.com;dummy2@mycomp.com"/>

Result was: <br/>

<c:out value="${success}"/><br/>
```

Management Tags

Description

These tags are made to retrieve system information from InfoGlue.

Location

Located in infoglue-management

Available tags and short description

Tag	Description
principal	This tag returns a principal (user more or less).
principalProperty	Returns a user property
language	Returns the language object.
contentTypeDefinitionAttributes	Gets the attributes of a content type. Useful for creating forms etc.
contentTypeDefinitionAssets	Gets the defined asset keys of a content type. Useful for creating forms etc.
categoryPath	This tag fetches the full path of a category. Important tag in followup searches based on assigned categories for example.
remoteUserPropertiesService, userPropertiesAttributeParameter	Lets you manage user properties from deliver engine.
categoryPath	This tag fetches the full path of a category. Important tag in followup searches based on assigned categories for example.
accessRights	This tag fetches a list of access rights for a certain entity.

principal

Description

This taglib gets you a principal either by userName or through the contentVersion.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
userName	false		Object/String	The userName of the principal we want to load.
contentVersion	false		Object/String	You can load the principal that modified this version last.

ID-result variable contains:

A org.infoglue.cms.security.InfoGluePrincipal-object.

Examples

The following example gets the latest content version object based on the content object previously fetched. It then gets the latest editor and presents his/her name.

```
<content:contentVersion id="articleContentVersion" content="${articleContent}"/>

<management:principal id="articleModifyer"
contentVersion="${articleContentVersion}"/>

The article was last changed by <c:out value="${articleModifyer.firstName}"/>
<c:out value="${articleModifyer.lastName}"/>
```

principalProperty

Description

This taglib returns a principal property.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
userName	false		Object/String	The userName of the principal on which we wish to extract a property.
principal	false		Object/String	The principal on which we wish to extract a property.
attibuteName	false		Object/String	The property to fetch.

ID-result variable contains:

A String-object.

Examples

This will result in that the alias of the current user is printed if it's set in the users properties.

```
<management:principalProperty id="alias" attributeName="Alias"/>

The user alias is: <c:out value="${alias}"/>
```

language

Description

This taglib gets a language object.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
languageId	false		Object/String	The id of the language we wish to fetch.
languageCode	false		Object/String	The code of the language we wish to fetch.

ID-result variable contains:

A org.infoglue.cms.entities.management.LanguageVO-object.

Examples

This will result in that the variable "language" will contain the language object.

```
<management:language id="currentLanguage" languageId="${pc.languageId}"/>

Current language as a localized representation is: <c:out
value="${currentLanguage.localizedDisplayLanguage}"/>
```

contentTypeDefinitionAttributes

Description

This tag lets you get a list of all attribute definitions on a content type.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
schemaValue	true		Object/String	The schema value of the content type you want to analyze

ID-result variable contains:

A list of org.infoglue.cms.entities.management.ContentTypeAttribute-objects.

Examples

```
<content:contentTypeDefinition id="ctd" contentTypeDefinitionName="Article"/>

<management:contentTypeDefinitionAttributes
id="contentTypeDefinitionAttributes" schemaValue="${ctd.schemaValue}"/>
```

categoryPath

Description

This tag fetches the full path of a category. Important tag in followup searches based on assigned categories for example.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
categoryId	true		Object/String	The categoryId you wish to get the full path to.

ID-result variable contains:

A String-object.

Examples

The following example gets a list of ContentCategory-objects assigned to the latest content version in the current language on the content bound by the property Article. It also prints the list and shows off another tag which gives you the full path.

```
<content:assignedCategories id="categories" propertyName="Article"
categoryKey="Area"/>
```

```
<c:forEach var="category" items="${categories}" varStatus="count">
   <c:out value="${category.id}"/>
   <management:categoryPath id="path" categoryId="${category.id}"/>
   path: <c:out value="${path}"/><br>
</c:forEach>
```

contentTypeDefinitionAssets

Description

This tag lets you get a list of all asset key definitions on a content type.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
schemaValue	true		Object/String	The schema value of the content type you want to analyze

ID-result variable contains:

A list of org.infoglue.cms.applications.databeans.AssetKeyDefinition-objects.

Examples

```
<content:contentTypeDefinition id="ctd" contentTypeDefinitionName="Article"/>

<management:contentTypeDefinitionAssets id="contentTypeDefinitionAssets"
schemaValue="${ctd.schemaValue}"/>
```

remoteUserPropertiesService

Description

Allows you to for example update the user properties.

Parameters

Name	Required Default	Type	Description
id	true	String	The result is stored in this variable.
targetEndpointAddress	false	String/Object	The cms-endpoint adress to call with the SOAP-request.
operationName	true	String/Object	The operation you wish to perform on the user properties.
principal	false	String/Object	State this if you wish to override which user to update.
languageId	false	String/Object	In which language do you want to edit the property. Defaults to the current site language.
contentTypeDefinitionId	false	String/Object	Sets which content type to use when creating a new user property object if no existing is available.

ID-result variable contains:

The status of the webservice call as a string.

Examples

This example updates the users Alias-attribute.

```
<page:deliveryContext id="dc" useFullUrl="true" disableNiceUri="false"/>

<content:contentTypeDefinition id="ctd" contentTypeDefinitionName="User
properties"/>

<management:remoteUserPropertiesService id="rups"
operationName="updateUserProperties" languageId="${dc.languageId}"
contentTypeDefinitionId="${ctd.id}">

        <management:userPropertiesAttributeParameter name="Alias"
value="Blade"/>

</management:remoteUserPropertiesService>
```

accessRights

Description

Fetches the access rights for a certain entity in the system. Very useful for some specific situations. Probably mostly usable for experts in unusual situations.

Parameters

Name	Required	Default	Type	Description
id	true		String	The result is stored in this variable.
interceptionPointName	true		String/Object	The interception point you wish to check for access rights.
parameters	true		String/Object	Depending on the interception point you will need to send in contentId, siteNodeId or similar.

ID-result variable contains:

A list of AccessRightVO.

Examples

This example get's all access rights and print out the roles/groups/users for it. In this case the access rights are fetched for the content with id 543.

```
<management:accessRights id="accessRightsVOList"
interceptionPointName="Content.Read" parameters="543"/>

<h1>Access rights</h1>

<c:forEach var="accessRightsVO" items="${accessRightsVOList}">
  <c:out value="${accessRightsVO}"/>

  <c:forEach var="role" items="${accessRightsVO.roles}">
    role: <c:out value="${role.roleName}"/><br/>
  </c:forEach>

  <c:forEach var="group" items="${accessRightsVO.groups}">
    group: <c:out value="${group.groupName}"/><br/>
  </c:forEach>

  <c:forEach var="user" items="${accessRightsVO.users}">
    user: <c:out value="${user.userName}"/><br/>
  </c:forEach>

</c:forEach>
```

Appendix A GNU Free Documentation License

GNU Free Documentation License
Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly

within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a

section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that

edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers

or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section
may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements"
or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or
appendices that qualify as Secondary Sections and contain no material
copied from the Document, you may at your option designate some or all
of these sections as invariant. To do this, add their titles to the
list of Invariant Sections in the Modified Version's license notice.
These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains
nothing but endorsements of your Modified Version by various
parties--for example, statements of peer review or that the text has
been approved by an organization as the authoritative definition of a
standard.

You may add a passage of up to five words as a Front-Cover Text, and a
passage of up to 25 words as a Back-Cover Text, to the end of the list
of Cover Texts in the Modified Version. Only one passage of
Front-Cover Text and one of Back-Cover Text may be added by (or
through arrangements made by) any one entity. If the Document already
includes a cover text for the same cover, previously added by you or
by arrangement made by the same entity you are acting on behalf of,
you may not add another; but you may replace the old one, on explicit
permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License
give permission to use their names for publicity for or to assert or
imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this
License, under the terms defined in section 4 above for modified
versions, provided that you include in the combination all of the
Invariant Sections of all of the original documents, unmodified, and
list them all as Invariant Sections of your combined work in its
license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and
multiple identical Invariant Sections may be replaced with a single
copy. If there are multiple Invariant Sections with the same name but
different contents, make the title of each such section unique by
adding at the end of it, in parentheses, the name of the original
author or publisher of that section if known, or else a unique number.
Make the same adjustment to the section titles in the list of
Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History"
in the various original documents, forming one section Entitled
"History"; likewise combine any sections Entitled "Acknowledgements",
and any sections Entitled "Dedications". You must delete all sections

Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4.
Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.