

* Created by [Tobias Mattsson](https://documentation.magnolia-cms.com/display/%7Etmattsson), last modified on [2016-04-29](https://documentation.magnolia-cms.com/pages/diffpagesbyversion.action?pageId=105697429&selectedPageVersions=39&selectedPageVersions=40)

[Go to start of metadata](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#page-metadata-start)

|  |  |
| --- | --- |
| Download | [magnolia-module-blossom.jar](https://nexus.magnolia-cms.com:443/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.1.3/magnolia-module-blossom-3.1.3.jar) |
| Edition | CE |
| License | [MLA](https://www.magnolia-cms.com/magnolia/pricing/license-faq.html), [GPL](http://www.gnu.org/licenses/gpl.html) |
| Issues | [BLOSSOM](http://jira.magnolia-cms.com/browse/BLOSSOM) |
| Maven site | [Blossom](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/) |
| Latest version | 3.1.3 |

The Blossom module makes the Spring Framework available for Magnolia. Blossom enables you to create editing components that display a high level of dynamic behavior. These components can be used by editors to create truly interactive web pages. For example, the Spring Framework application stack makes it easier to connect to business systems, so that they fetch the information that you want to present in your pages. As Blossom is built on the Spring Web MVC, familiarity with this framework will ensure a smooth experience working with the module.

**Key Blossom functionality**

* Annotation based API that builds on the Spring Web MVC. The Blossom API automatically detects annotated classes and registers them for use in Magnolia. To do this, simply add @Template to your controllers and they are ready to be used as building blocks by editors.
* Exposes controllers as templates and components, allowing you to use the controller for building a model. Useful if you need to call a web service for information that you need to present or if you need to read information from a database.
* Template based. Having templates (pages, areas and components) backed by an MVC framework has the benefit of providing natural web development business logic.
* Enable the re-use of previously created controllers.
* Allows you to create dialogs with code rather than configuration. This has many benefits:
  + you can populate the dialog at runtime with options detected at runtime:
  + you can obtain dialogs from the repository and place into your source control (CVS, SVN, GIT).
* Components can be executed in front of Magnolia. This means that it can choose to do a redirect and skip page rendering. This is very useful, for instance, if you have a form that on post should either present an error message or do a redirect.

**Versions and compatibility**

With Magnolia 5.4 use Blossom 3.1  
With Magnolia 5.1 to 5.3 use Blossom 3.0, its documentation is available [here](https://documentation.magnolia-cms.com/display/DOCS53/Blossom+module)   
With Magnolia 4.5 use Blossom 2.0, its documentation is available [here](https://documentation.magnolia-cms.com/display/DOCS45/Blossom+module)   
With Magnolia 4.1 to 4.4 use Blossom 1.2, its documentation is available [here](https://documentation.magnolia-cms.com/display/DOCS45/Blossom+1.2+module)

* [Key Blossom functionality](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-KeyBlossomfunctionality)
* [Versions and compatibility](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Versionsandcompatibility)
* [Minimum Requirements](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-MinimumRequirements)
* [Installing](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Installing)
* [Getting Started](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-GettingStarted)
  + [Programming model](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Programmingmodel)
  + [Running the sample project](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Runningthesampleproject)
  + [Creating your own project](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Creatingyourownproject)
* [What's new in Blossom 3](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-What%27snewinBlossom3)
* [What's new in Blossom 2](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-What%27snewinBlossom2)
* [Reference](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Reference)
  + [Templates](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Templates)
    - [Controlling where a template can be used](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Controllingwhereatemplatecanbeused)
  + [Areas](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Areas)
    - [Controlling which components can be added to an area](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Controllingwhichcomponentscanbeaddedtoanarea)
    - [Area inheritance](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Areainheritance)
    - [Maximum number of components in area](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Maximumnumberofcomponentsinarea)
    - [Reusing areas between templates through class inheritance](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Reusingareasbetweentemplatesthroughclassinheritance)
    - [Auto generation](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Autogeneration)
  + [Components](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Components)
  + [Accessing content in controllers](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Accessingcontentincontrollers)
  + [Accessing Magnolia context objects in controllers](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-AccessingMagnoliacontextobjectsincontrollers)
  + [Template parameters](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Templateparameters)
  + [View rendering](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Viewrendering)
    - [Content redirects](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Contentredirects)
    - [Thymeleaf views](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Thymeleafviews)
    - [Substituting Blossom view resolvers with custom resolvers](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-SubstitutingBlossomviewresolverswithcustomresolvers)
  + [Dialogs](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Dialogs)
    - [Fluent builder style API](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-FluentbuilderstyleAPI)
    - [Dialog factories](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Dialogfactories)
    - [Templates create their own dialog](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Templatescreatetheirowndialog)
    - [Templates can contain dialog factories](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Templatescancontaindialogfactories)
    - [Dialogs and the class hierarchy](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Dialogsandtheclasshierarchy)
    - [Post create callbacks for dialogs](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Postcreatecallbacksfordialogs)
    - [Custom dialog actions](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Customdialogactions)
    - [Validating dialog input](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Validatingdialoginput)
    - [Using DAM fields](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-UsingDAMfields)
  + [Post processing callbacks prior to registration of templates, areas, components and dialogs](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Postprocessingcallbackspriortoregistrationoftemplates,areas,componentsanddialogs)
  + [Pre-execution of components](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Pre-executionofcomponents)
  + [Handler Mapping](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-HandlerMapping)
  + [Localization](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Localization)
    - [Spring i18n interoperability](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Springi18ninteroperability)
    - [Content translation](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Contenttranslation)
  + [Sites and the template prototype](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Sitesandthetemplateprototype)
  + [Multipart Requests](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-MultipartRequests)
  + [Virtual URI Mappings](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-VirtualURIMappings)
  + [Dependencies on Magnolia components](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-DependenciesonMagnoliacomponents)
  + [Configuring beans in the repository](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Configuringbeansintherepository)
  + [Autowiring RenderingModel classes](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-AutowiringRenderingModelclasses)
  + [Interactions with <mvc:annotation-driven/>](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Interactionswith%3Cmvc:annotation-driven/%3E)
  + [Additional DispatcherServlets and AJAX calls](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-AdditionalDispatcherServletsandAJAXcalls)
* [Examples](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Examples)
  + [Using pre-execution](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Usingpre-execution)
  + [Using Spring WebFlow inside Magnolia](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-UsingSpringWebFlowinsideMagnolia)
* [Version Repository and Release Notes](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-VersionRepositoryandReleaseNotes)
  + [Version 3.1.3](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.1.3)
  + [Version 3.1.2](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.1.2)
  + [Version 3.1.1](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.1.1)
  + [Version 3.0.7](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.0.7)
  + [Version 3.0.6](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.0.6)
  + [Version 3.0.5](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.0.5)
  + [Version 3.0.4](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.0.4)
  + [Version 3.0.3](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.0.3)
  + [Version 3.0.2](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.0.2)
  + [Version 3.0.1](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.0.1)
  + [Version 3.0](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version3.0)
  + [Version 2.0.8](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0.8)
  + [Version 2.0.7](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0.7)
  + [Version 2.0.6](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0.6)
  + [Version 2.0.5](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0.5)
  + [Version 2.0.4](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0.4)
  + [Version 2.0.3](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0.3)
  + [Version 2.0.2](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0.2)
  + [Version 2.0.1](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0.1)
  + [Version 2.0](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version2.0)
  + [Version 1.2.4](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version1.2.4)
  + [Version 1.2.3](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version1.2.3)
  + [Version 1.2.2](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version1.2.2)
  + [Version 1.2.1](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version1.2.1)
  + [Version 1.2](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version1.2)
  + [Version 1.1.1](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version1.1.1)
  + [Version 1.1](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version1.1)
  + [Version 1.0](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version1.0)
  + [Version 0.5](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Version0.5)
* [Resources](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Resources)

**Minimum Requirements**

* Blossom requires Magnolia 5.4 or later.
* Needs at least Java 7.
* Supports Spring Framework 3.2 or later.

**Installing**

Maven is the easiest way to install the module. Add the following dependency to your [bundle](https://documentation.magnolia-cms.com/display/DOCS/Bundles+and+webapps):

<dependency>

<groupId>info.magnolia.blossom</groupId>

<artifactId>magnolia-module-blossom</artifactId>

<version>3.1.3</version>

</dependency>

Pre-built jars are also available for download. See [Installing a module](https://documentation.magnolia-cms.com/display/DOCS/Installing+a+module) for help.

* [magnolia-module-blossom.jar](https://nexus.magnolia-cms.com:443/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.1.3/magnolia-module-blossom-3.1.3.jar)

**Getting Started**

**Programming model**

Blossom is an annotation based API that's used in combination with Spring MVC. This snippet gives an overview of how the API is used.

|  |
| --- |
| @Template(id = "myModule:components/text", title = "Text")  @Controller  public class TextComponent {        @TabFactory("Content")      public void contentTab(UiConfig cfg, TabBuilder tab) {          tab.fields(                  cfg.fields.text("heading").label("Heading"),                  cfg.fields.richText("body").label("Text body").required()          );      }        @TabFactory("Margins")      public void marginsTab(TabBuilder tab) {      }        @RequestMapping("/text")      public String render() {          return "components/text.jsp";      }  }    @Template(id = "myModule:components/main", title = "Main")  @Controller  public class MainTemplate {        @DialogFactory("frontpage-properties")      public void frontPageProperties(UiConfig cfg, DialogBuilder dialog) {          dialog.form().tabs(                  cfg.forms.tab("Properties").fields(                          cfg.fields.text("headline").label("Headline").description("The text to use as a headline")                  )          );      }        @RequestMapping("/main")      public String render() {          return "pages/mainTemplate.ftl";      }        @Area("Content")      @AvailableComponentClasses({TextComponent.class})      public static class ContentArea {            @RequestMapping("/main/content")          public String render() {              return "areas/contentArea.jsp";          }      }  }    @DialogFactory("main-properties")  public class MainDialogFactory {        @TabFactory("Properties")      public void propertiesTab(UiConfig cfg, TabBuilder tab) {      }  } |

**Running the sample project**

This sample is a complete web application based on Magnolia 4.5 and requires nothing more than a Maven installation.

1. Clone the [git repository](http://git.magnolia-cms.com/gitweb/?p=modules/blossom/samples.git), switch to the right branch, build the project and then run magnolia-blossom-sample-webapp using mvn jetty:run-war.
2. When maven has downloaded the necessary artifacts and started the application point your browser to [localhost:8080](http://localhost:8080).
3. Magnolia will now be in install mode and display a list of modules for install.  
   Complete the installation wizard and then log in using superuser for both username and password.

|  |
| --- |
| git clone http://git.magnolia-cms.com/git/modules/blossom/samples.git  cd samples  mvn install  cd magnolia-blossom-sample-webapp  export MAVEN\_OPTS="-XX:+CMSClassUnloadingEnabled -XX:PermSize=256M -XX:MaxPermSize=512M -Xmx512m"  mvn jetty:run-war |

The sample web app uses JSP EL 2.2 bundled with Servlet API 3.0, use Tomcat 7+ or another servlet container of the same standard compliance level.

**Creating your own project**

We recommend using the archetypes to create your own project. For more details see this [Getting started with Blossom](https://documentation.magnolia-cms.com/display/DOCS/Getting+started+with+Blossom).

**What's new in Blossom 3**

Major update for Magnolia 5 style dialogs, compatible with Magnolia 5.1 and later. In version 5 Magnolia has a brand new user interface and with it comes a new API for dialogs. This version of Blossom is an update for this API and also contains a few other improvements.

**Fluent builder style API for dialogs**

Building dialogs is now done using a fluent builder style API with a very compact syntax. The builders allow for controlling every detail of how a field should appear and behave. Where in the old API you had to resort to configure the controls using key and value pairs the new API has chainable setters for every property.

**Meta-annotations for component availability**

Components can be categorised using custom meta-annotations, these can then be specified when setting the available components on areas. All components having this component category will be available in the area.

**Support for autowiring in RenderingModels**

AbstractAutowiredRenderingModel is an abstract class for RenderingModels that need be to be autowired using Spring.

**Support for Spring MVC 3.1 MVC handler methods**

In Spring MVC 3.1 the implementation of annotated MVC was reimplemented adding new handler adapters and handler mappings, starting with Blossom 3.0.2 handlers detected using the new implementation are discovered.

**Post processing callbacks prior to registration of templates, areas, components and dialogs**

Blossom 3.0.2 introduced a new annotation @PreRegister to be used on methods that should be called prior to registration with Magnolia.

**Auto generation of components in areas**

The process of auto generation gives areas the opportunity to pre-populate areas with components to save editors some work.

**What's new in Blossom 2**

Major update for Magnolia 4.5 and its new rendering engine, page editor and templating concepts. The term paragraph is replaced with component. Components and templates have been streamlined and are internally identical, the only real difference is that they have different id formats which in turn controls where they're applicable.

**New id conventions for templates**

In Magnolia 4.5 template IDs are in the format of <moduleName>:<path>. (See the [Templates](https://documentation.magnolia-cms.com/display/DOCS/Templates) documentation.) As a consequence the Blossom convention was updated. You now need to specify the ID explicitly on your controller: @Template(id="myModule:pages/news", title="News").

For help on how to migrate your templates to the new naming convention, see [Migrating content when upgrading to Blossom 2](http://wiki.magnolia-cms.com/display/WIKI/Migrating+content+when+upgrading+to+Blossom+2) .

**Areas**

With Magnolia 4.5 areas are explicit entities rather than implicit as node collections. They're declared as nested classes (public static) directly within the page templates they belong to.

**Reference**

This is the reference documentation for Blossom. The sample code in this section uses Spring 2.5+ annotated controllers.

**Templates**

To make a controller function as a template, simply add the @Template annotation. For example:

|  |
| --- |
| @Template(id = "myModule:pages/section", title = "Section Template")  @Controller  public class SectionController {        @RequestMapping("/section")      public ModelAndView render() {          ...      }  } |

This controller will be exposed in Magnolia, showing up as 'Section Template'. Note that the 'id' follows a convention used in Magnolia: module name, followed by a path. For templates that you want to use for pages, the path needs to start with pages.

**Controlling where a template can be used**

Where a page template is available for editors to use can be customized in two ways. Either using the site module and configure in the site definition or by using the @Available annotation. If the @Available is used it will override any setting in the site definition. Use it on a method in your controller that returns boolean. It can receive as arguments an argument of type Node or Content, this is the page in the website for which to determine if the template is available. It can also accept an argument of type TemplateDefinition this is the internal definition used by Magnolia:

|  |
| --- |
| @Available  public boolean isAvailable(Node websiteNode) {      // Replace this with logic for your specific use case      return true;  } |

For more details see the documentation on the [site definition.](https://documentation.magnolia-cms.com/display/DOCS/Site+definition)

**Areas**

Areas are defined using nested classes (public static class) within a template or within an other area. Areas are standard Spring Web MVC controllers annotated with @Area.  In addition Areas have their own dialog.

|  |
| --- |
| @Template(title = "Main", id = "myModule:pages/main")  @Controller  public class MainTemplate {        @Area("Promos")      @Inherits      @AvailableComponentClasses({TextComponent.class})      @Controller      public static class PromosArea {            @RequestMapping("/mainTemplate/promos")          public String render() {              return "areas/promos.jsp";          }      }        // rest of the template excluded for brevity  } |

**Controlling which components can be added to an area**

To specify which components can be used in an area you can either list the ids of the components using the @AvailableComponents annotation, list their classes using the @AvailableComponentClasses annotation, or use meta-annotations to categorise your components. It's possible to use all three options on the same area. Meta-annotations are annotations that you create yourself and use on your components. They need to annotated with @ComponentCategory.

This annotation is a component category annotation.

|  |
| --- |
| @Retention(RetentionPolicy.RUNTIME)  @Target(ElementType.TYPE)  @ComponentCategory  public @interface Promo {  } |

Any controller annotated with it will be included in an area that specifies it.

|  |
| --- |
| @Controller  @Template(title = "Text", id = "myModule:components/text")  @Promo  public class TextComponent {      ...  } |

**Area inheritance**

A common requirement is to have every  page on a site or a whole section to have a consistent base look, sharing header, footer and banners. Area inheritance is a feature that makes an area include components from its parent pages.  To use this feature use the @Inherited annotation on an area.

|  |
| --- |
| @Inherits |

By default the annotation is set to filtered mode, where it will only include components explicitly marked to be inherited. This is done using a property named inheritable. To flag it the component needs to have a checkbox added to its dialog.

|  |
| --- |
| cfg.fields.checkbox("inheritable").label("Inherited") |

Filtering mode can be turned off causing all components to be inherited.

|  |
| --- |
| @Inherits(components = ComponentInheritanceMode.ALL) |

The annotation also allows you to control how properties are inherited. These are properties on the area itself. A property set on the area in the parent page is then available when the area is rendered in the child page.

|  |
| --- |
| @Inherits(properties = PropertyInheritanceMode.ALL) |

**Maximum number of components in area**

Sometimes its helpful to limit the number of components an editor can add into an area. On the area annotation you can specify the limit your design requires.

|  |
| --- |
| @Area(name = "Promos", maxComponents = 5) |

**Reusing areas between templates through class inheritance**

Its common that many templates in a project have the same set of areas by taking advantage of the class hierarchy you only need to define them once.

|  |
| --- |
| public abstract class AbstractTemplate {        @Controller      @Area("headerArea")      public static class HeaderArea {            @RequestMapping("/headerArea")          public String render() {              return "areas/headerArea";          }      }  }    @Controller  @Template(...)  public class ExampleTemplate extends AbstractTemplate {  } |

**Auto generation**

To reduce the effort needed by editors to create pages it can be useful to pre-populate pages by automatically adding (generating) components in areas. To use this feature add a method on your area and annotate it with @AutoGenerator. It will receive the Node for the area content.

|  |
| --- |
| @AutoGenerator  public void generate(Node node) {      // if node has no sub nodes add nodes for components  } |

**Components**

Components are controllers annotated with @Template and having an id in the format moduleName:components/\*.

|  |
| --- |
| @Template(id = "myModule:components/textAndImage", title="Text and Image")  @TemplateDescription("Adds a text section with an image")  @Controller  public class TextAndImageController {        @RequestMapping("/textAndImage")      public ModelAndView render() {          ...      }  } |

This controller will be visible as the Text and Image component. The description set using @TemplateDescription is shown when you choose a component to add in the page editor.

**Accessing content in controllers**

In controllers you can have the content object passed directly to your method. In templates this is the content object for the page, in components it's the content object for the component within the page. If you declare two arguments of type Content or Node the first one will be the content of the page and the second will be the content of the component. Other arguments supported are AggregationState, Context, WebContext, User and MgnlUser:

|  |
| --- |
| @RequestMapping("/book")  public String render(Node pageNode, Node componentNode) {      ...  } |

To do this, configure the BlossomHandlerMethodArgumentResolver on your RequestMappingHandlerAdapter.

**Accessing Magnolia context objects in controllers**

In addition to the content you can access a range of objects about the current rendering context by adding them as arguments to your @RequestMapping annotated methods.

* info.magnolia.rendering.template.TemplateDefinition The definition of the template as generated by Blossom based on the controllers annotations.
* info.magnolia.rendering.template.AreaDefinition When rendering an area this is the definition of the area generated by Blossom.
* info.magnolia.cms.core.AggregationState Aggregation of state about an operation in the CMS
* info.magnolia.context.Context Context regarding an operation in the CMS and methods for accessing the JCR.
* info.magnolia.context.WebContext Specialization of Context for web requests.
* info.magnolia.cms.security.User The current user.
* info.magnolia.cms.security.MgnlUser The current user as a instance of the default User implementation.
* info.magnolia.cms.core.Channel The current channel being rendered for in a multi-channel setup.
* info.magnolia.module.site.theme.Theme The current theme.
* info.magnolia.module.site.Site The current site.

**Template parameters**

On templates, areas and components you can declare parameters that can then be accessed in the view using the def.parameters.<parameter-name> notation.

Example:

|  |
| --- |
| @Controller  @Template(title = "Main", id = "blossomSampleModule:pages/main")  @TemplateParams({          @TemplateParam(name = "example", value="my-value")  }) |

To access the example parameter in a freemarker view use:

|  |
| --- |
| ${def.parameters.example!} |

**View rendering**

View rendering with Blossom is performed by Spring. Blossom comes with support for using the rendering chain in Magnolia both for JSPs and Freemarker. It works by having view resolvers that return views that call into Magnolia. As a result you'll have access to all the objects that Magnolia provides to template scripts.

**Content redirects**

Blossom also provides a convenient way of returning redirects to content. By adding the UuidRedirectViewResolver you can return the uuid to redirect to directly from your controller. This example returns a redirect to a page:

|  |
| --- |
| return "website:" + content.getUUID(); |

This example returns a redirect for a resource in the documents repository:

|  |
| --- |
| return "dms:" + content.getUUID(); |

Redirecting to the page currently being rendered is a very common case and Blossom has a shortcut syntax that does exactly this without requiring the uuid.

|  |
| --- |
| return "magnolia-redirect:main-content"; |

It's also possible to redirect to the current content being rendered when executing further down the content hierarchy.

|  |
| --- |
| return "magnolia-redirect:current-content"; |

**Thymeleaf views**

If you prefer using Thymeleaf there is a project on GitHub by Thomas Kratz that adds support for it to Blossom. It is fully functional and is in production use.

For more details see the [magnolia-thymeleaf-renderer project on GitHub](https://github.com/eiswind/magnolia-thymeleaf-renderer).

**Substituting Blossom view resolvers with custom resolvers**

If you substitute or combine the Blossom view resolvers with your own implementations, be aware that the GZip filter included in Magnolia wraps the response object and gives it some non-standard semantics. It will tell you that the response is not committed even though it has been written to. It will also allow you to do redirects even though the response has been written to.

Be aware that if you write controllers that exploit the GZip filter, you are creating a dependency and you will need to have the GZip filter active. Certain framework code, such as the JSP view rendering in Spring and WebFlow will test if the response is committed to detect if redirects are possible and sometimes to choose between doing a forward or an include.

**Dialogs**

Blossom allows you to create dialogs using code. One of the benefits of this is that when the controller fetches data from a web service you can populate your dialog with options retrieved from that web service. See also [Dialogs](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module).

For example, if the controller displays a list of the best selling books in a category of books you could call the web service to find out what categories are available and use them to populate a select control. Another benefit is that you get your dialogs out of the repository and into version control. You get to version your dialogs together with your controllers.

You can easily change your dialog code in your IDE, compile and hot swap the new code into an instance of Magnolia running in debug mode. This makes developing dialogs extremely fast.

**Fluent builder style API**

Dialogs are described using a model of definition classes. These define the structure of the dialog, its tabs and its fields. They describe both what the dialogs and its forms should look like and how they should behave. Blossom uses a fluent builder style API to construct these definitions. The API consists of config classes that produce builders, the builders produce definitions.

The central config class is UiConfig which is used to create tabs and all different kinds of fields provided by Magnolia.

Blossom provides UiConfig to all methods that create dialogs or populate dialogs. In addition it will also provide a TabBuilder or DialogBuilder, depending on the use case. These represent the tab or dialog that's being populated.

Note that is also possible to extend the dialog creation mechanism in Blossom to use custom config classes or replace UiConfig or TabBuilder with classes of your own for adding your custom or third-party fields. The article [Extending the Blossom TabBuilder](http://wiki.magnolia-cms.com/display/WIKI/Extending+the+Blossom+TabBuilder) on the wiki explains how this is done in detail.

**Dialog factories**

Dialog factories create dialogs. Although the dialogs are automatically exposed in Magnolia, you need to configure them as beans in Spring. The easiest way to do this is to use Spring 2.5+ component scan, however manual configuration in XML will work. For example:

|  |
| --- |
| @DialogFactory("front-page-dialog")  @TabOrder({"Content", "Margins"})  public class FrontPageDialog {        @TabFactory("Margins")      public void marginsTab(UiConfig cfg, TabBuilder tab) {          tab.fields(                  cfg.fields.staticField("static").description("Margins around the side of the front page"),                  cfg.fields.text("leftMargin").label("Left Margin").description("Left margin in pixels"),                  cfg.fields.text("rightMargin").label("Right Margin").description("Right margin in pixels");      }        @TabFactory("Content")      public void contentTab(UiConfig cfg, TabBuilder tab) {          tab.fields(                  cfg.fields.text("title").label("Title").description("The title of this page")          );      }  } |

This registers a dialog in Magnolia with the name front-page-dialog. It has two tabs, Content and Margins. Methods annotated with @TabFactory in superclasses will be called. If there are many dialogs that look the same you can create an abstract base class for them.

The order that the dialogs appear in has been set using the @TabOrder annotation. It sorts them based on their names. If you would rather sort them by their names you can change it with a property in your module descriptor.

|  |
| --- |
| <properties>    <property>      <name>magnolia.blossom.sortTabsByLabel</name>      <value>false</value>    </property>  </properties> |

**Templates create their own dialog**

A template always has a dialog so you don't have to create a dialog factory and link them with a name. Instead your controller will act as its own dialog factory. You can use the same annotations inside your controller that you can for dialog factories. (See above.) If you want your template to use a dialog that is created by a dialog factory or configured in the repository, it's possible to override this behavior and specify which dialog should be used:

|  |
| --- |
| @Template(dialog="my-dialog") |

**Templates can contain dialog factories**

Because templates are often designed for use with few dialogs, Blossom allows you to create these within your template class for ease of location. The concept is similar to dialog factories but here everything is done in a single method.

|  |
| --- |
| @Controller  @Template(id = "myModule:pages/productTemplate", title = "Product Template")  public class ProductTemplate {        @DialogFactory("product-dialog")      public void productDialog(UiConfig cfg, DialogBuilder dialog) {          dialog.form().tabs(                  cfg.forms.tab("Settings").fields(                          cfg.fields.text("title").label("Title")                  ),                  cfg.forms.tab("Properties").fields(                          cfg.fields.text("headline").label("Headline")                  )          );      }  } |

**Dialogs and the class hierarchy**

It's not uncommon to have several dialogs that are very similar. For instance you might want to have the same basic sets of tabs in a number of dialogs. Blossom makes this possible by allowing you to define tab factories in super classes. Methods are invoked in superclass-first order, methods within the same class have no defined order of invocation.

For instance a tab for meta data in an abstract base class extended by several templates.

|  |
| --- |
| public abstract class BasePageTemplate {        @TabFactory("Meta")      public void metaTab(UiConfig cfg, TabBuilder tab) {          tab.fields(              cfg.fields.text("metaAuthor").label("Author"),  cfg.fields.text("metaKeywords").label("Keywords"),              cfg.fields.text("metaDescription").label("Description")          );      }  } |

**Post create callbacks for dialogs**

Tab factories will always add a tab and then call your method to have it populated. If you wish to only add the tab in certain scenarios you can instead use a post create callback. You can then determine the scenario, for instance testing if the user has a certain role or if the page is in a certain part of the website tree. They can also be useful when you need to do changes to the dialog after all tab factories have been added. Methods are invoked in superclass-first order and always after tab factories in the same class, post create methods within the same class have no defined order of invocation.

In this example a field is added if a criteria is met.

|  |
| --- |
| @PostCreate  public void postCreate(DialogBuilder dialog, UiConfig cfg) {      if (some criteria) {          dialog.form().tab("Content").label("Content").fields(                  cfg.fields.text("header").label("header")          );      }  } |

**Custom dialog actions**

Blossom will add the standard save and cancel actions to the dialog before it is shown to the user unless any have been configured.

This example shows how to configure a custom save action using a post create callback.

|  |
| --- |
| @PostCreate  public void postCreate(UiConfig cfg, DialogBuilder dialog) {      // Add custom save action      MyCustomSaveDialogActionDefinition save = new MyCustomSaveDialogActionDefinition();      save.setName("commit");      save.setLabel("save");      dialog.addAction(save);          // Add the standard cancel action      CancelDialogActionDefinition cancel = new CancelDialogActionDefinition();      cancel.setName("cancel");      cancel.setLabel("cancel");      dialog.addAction(cancel);  } |

**Validating dialog input**

To ensure that the content is correctly structured you can add validators to the fields. In the example below the name field can not be empty and the email has to be valid.

|  |
| --- |
| @TabFactory  public void contentTab(UiConfig cfg, TabBuilder tab) {      tab.fields(         cfg.fields.text("name").label("Name").required(),         cfg.fields.text("email").label("Email").validator(cfg.validators.email())      );  } |

**Using DAM fields**

The DAM module has a config class of its own, DamConfig, Blossom also provides this automatically when added as an argument.

|  |
| --- |
| @TabFactory("Content")  public void contentTab(UiConfig cfg, DamConfig dam, TabBuilder tab) {      tab.fields(              cfg.fields.text("title").label("Title"),              cfg.fields.richText("body").label("Text"),              dam.fields.assetLink("photo").label("Photo")      );  } |

**Post processing callbacks prior to registration of templates, areas, components and dialogs**

Before Blossom registers templates, areas, components and dialogs with Magnolia at startup, effectively making them available for use, it will inspect its class for methods annotated with @PreRegister and invoke those methods. This can be used to customize the definition Blossom built based on the annotations present on the class.

**@PreRegister in template or component**

|  |
| --- |
| @PreRegister  public void register(BlossomTemplateDefinition templateDefinition) {  } |

**@PreRegister in area**

|  |
| --- |
| @PreRegister  public void register(BlossomAreaDefinition areaDefinition) {  } |

**@PreRegister in dialog**

|  |
| --- |
| @PreRegister  public void register(BlossomDialogDescription dialogDescription) {  } |

**Pre-execution of components**

Pre-execution allows you to write components that can take care of the entire page rendering. This is necessary if you want to do redirects. Another scenario is if you want to have your controller rendering a form into the page on GET requests and write an XML document or a PDF on POST requests.

Note that this feature is enabled by default from version 1.1 and disabled by default in 0.5. To enable it you need to change a flag in the repository. Navigate to config:/server/filters/cms/blossom in AdminCentral and change the property enabled from 'false to 'true'.

In order for pre-execution to work you need to use the BlossomHandlerMapping. See section on this page on Handler Mapping.

Pre-execution is performed by a filter that will intercept requests and look for a request parameter that contains the UUID of the component that is to be pre-executed. The request parameter is named \_pecid. Blossom provides a taglib to make it easier to include this request parameter.

The blossom taglib has two tags, <blossom:pecid-input /> that will output <input type="hidden" name="\_pecid" value="<component uuid>" />. This tag is also available under the alias <blossom:pecidInput />, this is necessary for use in Freemarker templates.

|  |
| --- |
| <%@ taglib uri="blossom-taglib" prefix="blossom" %>  <form action="?" method="POST">      <blossom:pecid-input />      <input type="text" name="q" />      <input type="submit" value="Search" />  </form> |

It is possible to use <blossom:pecid var="pecid" /> to set the UUID of the component as a variable, in this example named pecid. The var attribute is optional. If it's not specified the component UUID is written directly into the page.

|  |
| --- |
| <a href="/news/new-website-launched.html?\_pecid=<blossom:pecid />" |

When you have multiple forms on the same page it can be helpful to distinguish which one is supposed to handle a POST request by testing in a controller if it is being pre-executed.

|  |
| --- |
| if (request.getMethod().equals("POST") &amp;&amp; PreexecutionUtils.isPreexecuting() {      processFormSubmit(request);  } else {      showForm(request);  } |

At the end of this article you'll find a full example of how to use pre-execution.

**Handler Mapping**

Spring uses HandlerMappings to map a request to a handler (usually a controller). Blossom interrogates the HandlerMappings to find out how you have mapped your handlers. Your HandlerMapping needs to be a class inherited from AbstractUrlHandlerMapping. This is usually the case, both Spring 2.5+ annotated controllers and BeanNameUrlHandlerMapping inherit from AbstractUrlHandlerMapping.

If you want support for pre-execution (see below) you will need to have BlossomHandlerMapping delegate to your HandlerMapping. This is necessary because this is where pre-execution is taken care of.

Note, you have to explicitly declare your HandlerMapping in XML. Although Spring will supply you with defaults even if you do not use XML, it is not possible for Blossom ot get a reference to the defaults and therefore will not take part in the rendering.

**Localization**

Using @I18nBasename you can set the name of the resource bundle you want to use. This can be used on templates, areas and components and in dialog factories. Labels and descriptions set for instance using the @Template and @TabFactory annotations will then be localized using the specified resource bundle.

|  |
| --- |
| @Controller  @Template(title = "Text", id = "blossomSampleModule:components/text")  @I18nBasename("info.magnolia.blossom.sample.messages")  public class TextComponent {      @TabFactory("textComponent.contentTab.label")    public void contentTab(UiConfig cfg, TabBuilder tab) {      tab.fields(        cfg.fields.text("heading").label("textComponent.contentTab.heading")      );    }  } |

You can also leave labels and titles empty to have Magnolia resolve them automatically by generating a key based on the dialog's name, the tab's name, and the field's name. However by default Blossom will use the value given in @TabFactory both for name and label of the tab. In order to leave the label empty so automatic i18n takes effect you need to set a property in your module descriptor.

|  |
| --- |
| <properties>    <property>      <name>magnolia.blossom.setTabLabels</name>      <value>false</value>    </property>  </properties> |

**Spring i18n interoperability**

Blossom provides you with a LocaleResolver to enable Spring to pick up the locale that Magnolia has selected for the current request:

|  |
| --- |
| info.magnolia.module.blossom.context.MagnoliaLocaleResolver |

To bridge Magnolia i18n into Spring, use the following MessageSource implementation:

|  |
| --- |
| info.magnolia.module.blossom.context.MagnoliaMessageSource |

**Content translation**

If the content of a tab field should be translatable then it can be made i18n aware by using the i18n() method.

|  |
| --- |
| @TabFactory("textComponent.contentTab.label")   public void contentTab(UiConfig cfg, TabBuilder tab) {     tab.fields(       cfg.fields.text("heading").label("textComponent.contentTab.heading").i18n()     );   } |

**Sites and the template prototype**

In the site definition you can define a template prototype which is a template that other templates inherit from. Things configured on the prototype such as areas are copied onto other templates.

To use this feature with Blossom you need to use a special view renderer. Blossom provides a FactoryBean for creating this renderer, SiteAwareFreemarkerTemplateViewRendererFactoryBean.

The [Blossom sample](https://documentation.magnolia-cms.com/display/DOCS/Blossom+module#Blossommodule-Blossommodule-Runningthesampleproject) has example configuration for this renderer and the [getting started guide](https://documentation.magnolia-cms.com/display/DOCS/Getting+started+with+Blossom) uses a maven archetype that generates this configuration for you.

For more details see the documentation on the [site definition](https://documentation.magnolia-cms.com/display/DOCS/Site+definition).

**Multipart Requests**

Because Magnolia handles multipart requests, Blossom provides a MultipartResolver implementation. Add this snippet to your beans XML. In the example above this would be blossom-servlet.xml.

|  |
| --- |
| <bean id="multipartResolver" class="info.magnolia.module.blossom.multipart.BlossomMultipartResolver" /> |

**Virtual URI Mappings**

Any bean implementing VirtualUriMapping that is configured with Spring is automatically detected and exposed in Magnolia. This means that you don not need to configure them in the repository as you normally would.

In addition, Blossom supports writing virtual URI mappings by annotating a class with @VirtualURIMapper. Instances of this class, when configured as a bean in Spring, will be scanned for methods that can be used for URI mapping. Any method that return String or MappingResult and accepts as arguments a String that is the incoming URI or a HttpServletRequest is used.

The returned string from a Virtual URI mapping is the new URI that Magnolia will use to look up the page in the repository that is to be rendered. The returned URI can also be prefixed with "redirect:", "permanent:" or "forward:" to trigger either a temporary redirect, a permanent redirect or a forward respectively. For redirects the URI can be absolute or relative within the web application (the context path is added automatically).

|  |
| --- |
| @VirtualURIMapper  public class SampleURIMapper {        public String about(String uri, HttpServletRequest request) {          if (uri.equals("/about"))              return "/sections/about";          return null;      }        public String news(String uri, HttpServletRequest request) {          if (uri.equals("/news"))              return "forward:/dispatcher/news.do";          return null;      }  } |

For more details the reference documentation on  [Virtual URI mapping](https://documentation.magnolia-cms.com/display/DOCS/URI+mapping" \l "URImapping-VirtualURImapping) .

**Dependencies on Magnolia components**

Magnolia is not based on Spring, but rather uses a container internally that manages components. It is possible to pull these components in as beans in the ApplicationContext. Here they are made available for declaring dependencies on them and as candidates for autowiring. This is an example that exposes the ModuleManager as a bean with id 'moduleManager':

|  |
| --- |
| <blossom:component id="moduleManager" type="info.magnolia.module.ModuleManager" /> |

(Note the id is optional). For more details on Magnolia Components see the documentation on  [Configuration](https://documentation.magnolia-cms.com/display/DOCS/Configuration+management) .

**Configuring beans in the repository**

Besides for storing content for web pages, the JCR repository is also widely used for configuration. One benefit of this is that configuration is easy to change while the application is running.

To read this configuration Magnolia uses a mechanism called Node2Bean to transform the configuration into plain Java beans. The process is based on reflection. Blossom provides an extension to this mechanism which, during initialization of the bean, applies post processing on it using the Spring ApplicationContext.

This makes it possible to have dependencies in the bean that should be autowired by the ApplicationContext. Note that the bean will also receive all the standard life cycle callbacks that a bean managed by Spring can use. These include:

* InitializingBean,
* ServletContextAware,
* ApplicationContextAware
* and many more.  
  It can also use @PostConstruct and it will be subject for having AOP aspects applied.

The Node2Bean mechanism by default calls a method with the signature public void init() on the beans it creates if it exists. The extension provided by Blossom does not support this. Instead use @PostConstruct.

With Blossom it is possible to use this mechanism directly in Spring beans xml files. The bean is proxied and is transparently replaced if changes are made in the repository.

This is an example for a book store web site that has a discount service configured in the repository:

|  |
| --- |
| <blossom:configured-bean id="discountService" path="/modules/book-store-module/beans/discountService" /> |

This example is identical to the one above except that the configuration is observed for any changes being made. When it changes the bean is reloaded making it easy to change the discounts while the system is running:

|  |
| --- |
| <blossom:observed-bean id="discountService" path="/modules/book-store-module/beans/discountService" /> |

By default the proxy is created using JDK dynamic proxies, this proxies on an interface level. If you require proxying on the class you can specify proxy-target-class="true" and the proxy will instead be a cglib class proxy. This is analogous to the way Spring AOP uses proxies.

For more details on Node2Bean see the documentation on  [Configuration](https://documentation.magnolia-cms.com/display/DOCS/Configuration+management) .

**Autowiring RenderingModel classes**

When using standard Magnolia templates with RenderingModel classes do their backing logic you can get access to beans managed by Spring through autowiring. Blossom provides an abstract base class for this called AbstractAutowiredRenderingModel. It will do autowiring on itself using the root web application context. Note that Magnolia will still create the RenderingModel and since it supports the @Inject annotation you should use @Autowired for the dependencies you want to from Spring.

|  |
| --- |
| public class MyRenderingModel extends AbstractAutowiredRenderingModel<TemplateDefinition> {        @Autowired      private MyService service;        public MyRenderingModel(ServletContext servletContext, Node content, TemplateDefinition definition, RenderingModel<?> parent) {          super(content, definition, parent, servletContext);      }      ...  } |

In order to use this you will need to have annotation config enabled in your root web application context. Add the following to your applicationContext.xml.

|  |
| --- |
| <context:annotation-config/> |

**Interactions with <mvc:annotation-driven/>**

The <mvc:annotation-driven/> tag is essentially a macro. It adds a number of beans, such as message converters, handler adapters and so on. They're all configured with reasonable defaults that suits the majority of users. That is, until you actually need something that differs from the defaults. Then you need to remove it and explicitly configure instead.

In order for Blossom to work it customizes some of the same beans that <mvc:annotation-driven/> adds. Therefor some things will break when you add <mvc:annotation-driven/>. One thing that does break is pre-execution. In order for it to work all handler adapters need to be configured within a BlossomHandlerMapping bean.

A common reason for adding <mvc:annotation-driven/> is bean validation of submitted forms since it adds configuration for this by default. When using Blossom you need to set up configuration for bean validation explicitly. If you've used the archetypes to create your module this was done for you. Otherwise refer to this snippet that shows how its done:

|  |
| --- |
| <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">    <property name="customArgumentResolver">      <bean class="info.magnolia.module.blossom.web.BlossomWebArgumentResolver" />    </property>    <property name="webBindingInitializer">      <bean class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer">        <property name="validator">          <bean class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean" />        </property>      </bean>    </property>  </bean> |

**Additional DispatcherServlets and AJAX calls**

Controllers used for rendering content are not accessible for requests coming in to the servlet container. Requests that come in are handled my Magnolia and mapped to content that is only later rendered by these controllers. The BlossomDispatcherServlet that manages these controllers are started in your module class and the servlet container has no knowledge of this servlet. Since these controllers are non meaningful without content there's no reason to have them directly accessible.

Controllers that should be accessible directly and are not used to render content need to be managed by a different DispatcherServlet.

There's two ways to add a DispatcherServlet in a Magnolia project. You either simply add one to web.xml or you add to your module descriptor.

Adding it to the module descriptor is the better choice because then its started after your module has started and it will become a child of the root web application context and can access all beans within it. You also won't need to update web.xml which makes module install easier and upgrades of Magnolia easier.

The module descriptor is your **/src/main/resources/META-INF/magnolia/<module name>.xml** file.

Here's an example of what that looks like:

|  |
| --- |
| <servlets>    <servlet>      <name>dispatcher</name>      <class>org.springframework.web.servlet.DispatcherServlet</class>      <mappings>        <mapping>/ajax/\*</mapping>      </mappings>      <params>        <param>          <name>contextConfigLocation</name>          <value>classpath:/ajax-servlet.xml</value>        </param>      </params>    </servlet>  </servlets> |

Magnolia installs the servlet when the module is installed so after you've added it you need to reinstall your module. When it's installed you can see it in the configuration app at **/server/filters/servlets/dispatcher**.

In the example above the DispatcherServlet will get all requests with a path starting with **/ajax/** and it uses a configuration file**/src/main/resources/ajax-servlet.xml**.

You'll want to make sure that **ajax-servlet.xml** has only the controllers that should be accessible from the outside and not those used for rendering content. Do this by organizing them in different packages and use component scan to find the right ones.

Here's what a simple**ajax-servlet.xml** would look like:

|  |
| --- |
| <beans xmlns="[http://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans)         xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)         xmlns:context="[http://www.springframework.org/schema/context"](http://www.springframework.org/schema/context)         xsi:schemaLocation="<http://www.springframework.org/schema/beans> <http://www.springframework.org/schema/beans/spring-beans.xsd> <http://www.springframework.org/schema/context> [http://www.springframework.org/schema/context/spring-context.xsd"](http://www.springframework.org/schema/context/spring-context.xsd)>      <context:annotation-config/>      <context:component-scan base-package="my.module.ajax" />    </beans> |

It will component scan and find the controller below which becomes accessible at  **/ajax/hello** , note that the /ajax part is not specified in the controller.

|  |
| --- |
| package my.module.ajax;    import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.ResponseBody;    @Controller  public class AjaxController {        @ResponseBody      @RequestMapping("/hello")      public String render() {          return "Hello!";      }  } |

**Examples**

**Using pre-execution**

This is a simple controller that displays a form:

|  |
| --- |
| @Controller  @RequestMapping("/search")  @Template(id = "myModule:components/searchForm", title = "Search form")  @TemplateDescription("Adds a customizable search form")  public class SearchController {        @RequestMapping(method = RequestMethod.GET)      public ModelAndView form() {          return new ModelAndView("searchForm");      }        @RequestMapping(method = RequestMethod.POST)      public ModelAndView search(@RequestParam("q") String q, Node content) throws RepositoryException {          if (StringUtils.isBlank(q)) {              return new ModelAndView("searchForm", "errorMessage", content.getProperty("errorMessage").getString());          }          String searchUrl = content.getProperty("searchUrl").getString();          return new ModelAndView(new RedirectView(searchUrl + q));      }        @TabFactory("Settings")      public void settingsTab(UiConfig cfg, TabBuilder tab) {          tab.fields(                  cfg.fields.text("title").label("Title").required().requiredErrorMessage("You need to enter a title!"),                  cfg.fields.text("bodyText").label("Text"),                  cfg.fields.text("searchUrl").label("Search engine URL").description("For instance: [http://www.google.com/search?q="](http://www.google.com/search?q=)),                  cfg.fields.text("errorMessage").label("Error message").description("For instance: You need to enter a query")          );      }  } |

The JSP for this component uses the blossom taglib to add the request parameter.

|  |
| --- |
| <%@ taglib uri="[http://java.sun.com/jsp/jstl/core"](http://java.sun.com/jsp/jstl/core) prefix="c" %>  <%@ taglib uri="blossom-taglib" prefix="blossom" %>  <h1>${content.title}</h2.  <p>${content.bodyText}</p>  <c:if test="${not empty errorMessage}">  <p style="color:red;">${errorMessage}</p>  </c:if>  <form action="?" method="POST">      <blossom:pecid-input />      <input type="text" name="q" />      <input type="submit" value="Search" />  </form> |

**Using Spring WebFlow inside Magnolia**

It is possible to include a webflow as a component and embed it into a page. The example below is a controller that exposes a booking flow:

|  |
| --- |
| @Controller  @Template(id="myModule:components/bookingFlow", title = "Booking Flow")  public class BookingFlowController extends FlowController {        @Override      @Autowired      public void setFlowExecutor(FlowExecutor flowExecutor) {          super.setFlowExecutor(flowExecutor);      }        @Override      @RequestMapping("/booking")      public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) throws Exception {          return super.handleRequest(request, response);      }  } |

Note that you need to turn off the redirectOnPause behavior since it's not usable when the flow is embedded in a page. Also, you might want to customize the handling of NoSuchFlowException.

**Version Repository and Release Notes**

**Version 3.1.3**

Adds support for declaring template parameters with @TemplateParam and for specifying on a template whether it can be  
changed, moved and deleted.

Improvements:

* BLOSSOM-231 Blossom seems to have no location in metadata
* BLOSSOM-233 Duplicate registration of templates with multiple @RequestMapping methods
* BLOSSOM-234 Upgrade sample to Magnolia 5.4.6 and Spring 4.2.5

Added features:

* BLOSSOM-235 Support for setting writable, movable and deletable with @Template
* BLOSSOM-232 Annotations for declaring template parameters

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.1.3/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.1.3/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.1.3/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.1.3)

**Version 3.1.2**

Fixes a problem with using certain arguments in @RequestMapping annotated methods and introduces two properties for  
controlling whether labels should are set on tabs and whether to sort tabs by label or name. These properties makes it  
easier to adopt the new I18n style in Magnolia.

Improvements:

* BLOSSOM-226 Call to annotated method @RequestMapping on a template ends in Argument type mismatch
* BLOSSOM-228 Compatibility with new I18n API

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.1.2/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.1.2/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.1.2/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.1.2)

**Version 3.1.1**

Adds integration with the Magnolia Site module allowing Blossom templates to be merged with the template prototype.  
Blossom now requires the Site module to be installed but using the template prototype is still optional.

Improvements:

* BLOSSOM-221 Allow setting type and subtype on templates
* BLOSSOM-222 Integration with site module
* BLOSSOM-223 Support Channel as an argument to @RequestMapping methods
* BLOSSOM-224 Update Blossom sample for site module support
* BLOSSOM-225 Compatibility fixes for Spring 4.2

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.1.1/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.1.1/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.1.1/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.1.1)

**Version 3.0.7**

Minor update that fixes a problem with permissions on components and a change to how the BlossomDispatcherServlet  
behaves on refresh allowing templates and dialogs to be reregistered.

Improvements:

* BLOSSOM-208 Permissions on components have no effect
* BLOSSOM-209 BlossomDispatcherServlet should send BlossomDispatcherInitializedEvent on ApplicationContext refresh

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.0.7/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.7/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.7/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.0.7)

**Version 3.0.6**

Minor update that makes using Spring java config easier with new methods in BlossomModuleSupport for starting context  
and servlets, configuring template view renderers and ordering handler mappings. Also adds more arguments to annotated  
methods called to create dialogs and opens the tab ordering logic to extension.

Improvements:

* BLOSSOM-201 Provide support for additional arguments in dialog construction callbacks
* BLOSSOM-202 Customize tab sorting
* BLOSSOM-203 Enhancements for spring java config

We'd like to thank everyone that has invested time in testing and providing feedback. Special thanks goes out to  
Frank Sommer.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.0.6/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.6/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.6/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.0.6)

**Version 3.0.5**

Adds post create callbacks for dialogs called after its tab factories have been run allowing post processing such as  
configuring fields and adding tabs. This release also adds a way of checking if handler is invoked during  
pre-execution, and performance improvements which should be especially notable when using freemarker or thymeleaf.

Improvements:

* BLOSSOM-109 Add tool annotations and flag attributes as required in xsd
* BLOSSOM-195 Performance improvements for forward and include wrappers
* BLOSSOM-199 Exception thrown when dialog factory fails should include called method in message

Added features:

* BLOSSOM-38 Add a mechanism for testing whether a controller is executed within pre-execution
* BLOSSOM-47 Post create callbacks for template, area, component and DialogFactory classes

We'd like to thank everyone that has invested time in testing and providing feedback. Special thanks goes out to  
Declan Newman.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.0.5/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.5/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.5/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.0.5)

**Version 3.0.4**

Minor update that adds support for setting the node creation flag on areas, and for flash attributes when using  
placeholder and uuid redirects. To enable flash support the standard RequestMappingHandlerAdapter has to be replaced  
with the specialized BlossomRequestMappingHandlerAdapter. For more details see <https://jira.spring.io/browse/SPR-12054>.

Improvements:

* BLOSSOM-181 Flash attributes not working with uuid redirects
* BLOSSOM-189 BlossomDispatcherServlet should implement all DispatcherServlet constructors
* BLOSSOM-191 Area annotation should also support area node creation flag

We'd like to thank everyone that has invested time in testing and providing feedback. Special thanks goes out to  
Frank Sommer and Andreas Antener.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.0.4/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.4/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.4/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.0.4)

**Version 3.0.3**

Adds compatibility with DAM 2.x, a short-hand syntax for redirecting to the current content and adds a flag for  
exposing model attributes when doing uuid redirects.

Improvements:

* BLOSSOM-184 Add support for DAM 2.x new package name of DamConfig.java

Added features:

* BLOSSOM-182 Support short-hand syntax for redirecting to the current content
* BLOSSOM-178 Add exposeModelAttributes property on UuidRedirectViewResolver

We'd like to thank everyone that has invested time in testing and providing feedback. Special thanks goes out to  
Dominika and Jean-Charles.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.0.3/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.3/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.3/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.0.3)

**Version 3.0.2**

Small feature release adding support for Spring 3.1 MVC handler methods, callbacks for auto generation of components in  
areas and post processing of definitions prior to registration.

Improvements:

* BLOSSOM-175 MagnoliaMessageSource should have configurable basenames

Added features:

* BLOSSOM-104 Support for inheriting areas from super class
* BLOSSOM-111 Add support for auto generation of components in areas
* BLOSSOM-167 Support Spring MVC 3.1 MVC handler methods
* BLOSSOM-172 Add post processing callback prior to registration in templates, areas and dialogs
* BLOSSOM-176 Placeholder for redirecting to the current page

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.0.2/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.2/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.2/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.0.2)

**Version 3.0.1**

Small feature release adding support for limiting the number of components in an area and adds access to the Vaadin Item  
being used by a dialog.

Improvements:

* BLOSSOM-152 DialogCreationContext now have methods for accessing the JCR Node directly
* BLOSSOM-156 The Vaadin Item used by a dialog is now available in DialogCreationContext and methods building dialogs

Added features:

* BLOSSOM-147 Support for specifying maximum components in an area

We'd like to thank everyone that has invested time in testing and providing feedback. Special thanks goes out to  
Francisco J. Giner.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.0.1/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.1/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0.1/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.0.1)

**Version 3.0**

Major update for Magnolia 5 style dialogs, compatible with Magnolia 5.1 and later. In version 5 Magnolia has a brand new  
user interface and with it comes a new API for dialogs. This version of Blossom is an update for this API and also  
contains a few other improvements.

The new user interface is built using Vaadin. In previous versions of Magnolia you would create and configure controls,  
now you build an object model that defines how the dialog and its fields should appear and behave. This definition  
model is then used to create and configure vaadin components. The fields have better support for validation which is why  
@TabValidator and @DialogValidator have been removed in this version.

For composing the dialogs Blossom 3 uses a builder style API that creates a more fluent programming style. A new set of  
classes replaces the previous API. These classes include a new TabBuilder and DialogBuilder and provides builders for  
each of the built-in fields, making it easier to configure properties on the fields.

A new feature in this release is meta-annotations for component availability. It allows for creating custom annotations  
that can then be used on components to categorize them. They can then be used for specifying which components can go  
into an area with the @AvailableComponentClasses annotation. All components having a specified annotation will be  
available.

Using Springs freemarker macros for forms is now straight forward. When rendering freemarker views Blossom will expose  
a RequestContext object which these macros require, it will also expose everything in the model as request attributes.  
This feature is enabled by default but can be switched off on the FreemarkerTemplateViewRenderer.

The multipart support that bridges Springs multipart handling to the multipart support in Magnolia has been updated for  
API changes made in Spring 3.1.

The groupId changes for this version from info.magnolia to info.magnolia.blossom.

Improvements:

* BLOSSOM-2 Spring macro helper and the model is now exposed as request attributes in freemarker templates
* BLOSSOM-89 Multipart resolver now implements methods required in Spring 3.1
* BLOSSOM-108 UUID redirects are not resolved in pre-execution
* BLOSSOM-114 BeanFactoryUtils should use SLF4J instead of commons logging
* BLOSSOM-115 BLOSSOM-121 Update for Magnolia 5.1
* BLOSSOM-119 Add spring schema for Blossom 3.0
* BLOSSOM-120 Update bootstrap files for metadata mixins
* BLOSSOM-127 Change groupId to info.magnolia.blossom
* BLOSSOM-130 Fix for using the same JSP nested multiple times

Added features:

* BLOSSOM-106 Extension hooks in TemplateExporter and DialogExporter for post processing definitions
* BLOSSOM-117 BLOSSOM-135 Add support for Magnolia 5 style dialogs
* BLOSSOM-131 Meta annotations for component availability
* BLOSSOM-133 Abstract support class for autowiring of rendering models
* BLOSSOM-136 Log all registered templates after template export
* BLOSSOM-142 Provide Node2BeanTransformer with autowiring support

We'd like to thank everyone that has invested time in testing and providing feedback. Special thanks goes out to  
Wolfgang Wachsmuth and TLN.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/blossom/magnolia-module-blossom/3.0/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/3.0/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-3.0)

**Version 2.0.8**

Minor update that opens the tab ordering logic to extension.

Improvements:

* BLOSSOM-202 Customize tab sorting

We'd like to thank everyone that has invested time in testing and providing feedback. Special thanks goes out to  
Frank Sommer.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0.8/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.8/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.8/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0.8)

**Version 2.0.7**

Adds a way of checking if handler is invoked during pre-execution, and performance improvements which should be  
especially notable when using Freemarker or Thymeleaf.

Improvements:

* BLOSSOM-195 Performance improvements for forward and include wrappers

Added features:

* BLOSSOM-38 Add a mechanism for testing whether a controller is executed within pre-execution

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0.7/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.7/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.7/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0.7)

**Version 2.0.6**

Adds a short-hand syntax for redirecting to the current content and adds a flag for exposing model attributes when doing uuid redirects.

Added features:

* BLOSSOM-178 Add exposeModelAttributes property on UuidRedirectViewResolver
* BLOSSOM-182 Support short-hand syntax for redirecting to the current content

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0.6/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.6/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.6/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0.6)

**Version 2.0.5**

Small feature release adding support for inheriting areas from super class and a shortcut for redirecting to the current page.

Improvements:

* BLOSSOM-175 MagnoliaMessageSource should have configurable basenames

Added features:

* BLOSSOM-104 Support for inheriting areas from super class
* BLOSSOM-176 Placeholder for redirecting to the current page

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0.5/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.5/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.5/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0.5)

**Version 2.0.4**

Small feature release adding support for limiting the number of components in an area.

Added features:

* BLOSSOM-147 Support for specifying maximum components in an area

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0.4/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.4/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.4/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0.4)

**Version 2.0.3**

Maintenance release for a regression in version 2.0.2.

Regressions:

* BLOSSOM-159 NPE when opening a dialog

Thanks to Jean-Charles for reporting this.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0.3/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.3/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.3/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0.3)

**Version 2.0.2**

Maintenance release fixing a number of reported issues.

Added features:

* BLOSSOM-112 Template and area definitions should be available for arguments to mvc handler methods

Improvements:

* BLOSSOM-2 Expose spring macro helper and the model as request attributes in freemarker templates
* BLOSSOM-96 Exception thrown when trying to export template without required Template annotation should include the class
* BLOSSOM-101 TemplateDefinitionBuilder#resolveTemplateId throws IllegalStateException (with empty error string) when class is missing Template annotation
* BLOSSOM-102 Configuration nodes for autodetected templates should be removed when updating from 1.x
* BLOSSOM-108 UUID redirects are not resolved in pre-execution
* BLOSSOM-130 Nesting the same JSP + Performance impact
* BLOSSOM-152 DialogCreationContext now have methods for accessing the JCR Node directly

Thanks to everyone who has reported issues and provided feedback. Special thanks goes out to Tomas Brimor, TLN and Francisco J. Giner.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0.2/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.2/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.2/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0.2)

**Version 2.0.1**

Small feature release adding support for finding annotations on proxies and a message on update about migration.

Improvements:

* BLOSSOM-88 Annotations are not found when using cglib proxies
* BLOSSOM-94 Added message about migration from 1.2.x

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0.1/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.1/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0.1/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0.1)

**Version 2.0**

Major update for Magnolia 4.5 and its new rendering engine, page editor and templating concepts. With Magnolia 4.5  
areas are explicit entities rather than implicit as node collections. The term paragraph is replaced with component.  
Components and templates have been streamlined and are internally identical, the only real difference is that they  
have different id formats which in turn controls where they're applicable.

@Paragraph has been deprecated and removed. Instead use @Template and set an id of the form <moduleName>:components/\*

@ParagraphDescription has been renamed to @TemplateDescription

Templates now have a dialog, use the dialog annotations in the same way as you did for paragraphs.

Ids for templates and components are no longer generated from their @RequestMapping paths. As of this version they need  
to be specified on the @Template annotation explicitly.

Paragraphs used to require a dialog even if there was nothing to set. This is no longer the case, a component or  
template that doesn't use any of the annotations used to create a dialog will simply not have a dialog. A component is  
added to the page immediately without a dialog being shown.

Areas are defined using nested classes (public static class) within a template or another area in which it belongs. They  
are normal Spring Web MVC controllers the same as templates and you annotate them with @Area. You can configure which  
components are available in them using the @AvailableComponents or @AvailableComponentClasses annotations. Areas have  
their own dialog.

An area can inherit components from areas of the same name in the page's parent pages. By default it will only inherit  
components marked for inheritance.

The content API, centered on info.magnolia.cms.core.Content, has been deprecated. It's still around and fully backwards  
compatible and will be for some time. Instead we favour using the JCR API directly. Therefore in all the places where  
Blossom was giving you a Content object you can now instead choose to receive a javax.jcr.Node object. This includes all  
the methods you've annotated with blossom annotations as well as your @RequestMapping methods.

No longer necessary to use the ServletContextExposingContextListener as the functionality it provided is now part  
of Magnolia itself. The class has been removed, you need to update your web.xml.

The <blossom:configuration> that was previously required in your applicationContext.xml is no longer required and need  
to be removed.

Added features:

* BLOSSOM-65 Templates are dialog factories for their own dialog
* BLOSSOM-71 @TemplateDescription replaces @ParagraphDescription
* BLOSSOM-72 @Paragraph is replaced with @Template
* BLOSSOM-74 Templates are identified with id
* BLOSSOM-78 Area configuration using nested classes
* BLOSSOM-82 Areas can have available components set using the classes defining the components
* BLOSSOM-87 @Available methods can receive the TemplateDefinition for the template

Improvements:

* BLOSSOM-59 FactoryBeans for beans configured in JCR use JCR API
* BLOSSOM-61 BlossomWebArgumentResolver supports passing object being rendered as JCR Node instances
* BLOSSOM-79 ServletContextExposingContextListener is dropped
* BLOSSOM-81 Empty dialogs are not displayed
* BLOSSOM-83 Template ids needs to be set explicitly, generation from @RequestMapping is dropped
* BLOSSOM-84 Template title is set with title field instead of value field

Behind the scenes:

* BLOSSOM-58 Update for Java 6 and JCR 2.0
* BLOSSOM-60 upgraded test to JUnit 4
* BLOSSOM-62 UuidRedirectView and resolver use JCR terminology (workspace instead of repository)
* BLOSSOM-63 MagnoliaComponentFactoryBean uses Components instead of FactoryUtil
* BLOSSOM-64 Templates are registered in Magnolia using providers
* BLOSSOM-67 Update rendering for Magnolia 4.5
* BLOSSOM-69 TemplateExporter is customizable in beans xml and have full control of the created definition
* BLOSSOM-70 DialogExporter is customizable in beans xml and have full control the the created definition
* BLOSSOM-73 Dialog registry is a magnolia component
* BLOSSOM-75 Dialogs are registered in Magnolia using providers
* BLOSSOM-76 Update template availability callbacks for Magnolia 4.5

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/2.0/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/2.0/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-2.0)

**Version 1.2.4**

Small feature release adding support for finding annotations on proxies.

Improvements:

* BLOSSOM-88 Annotations are not found when using cglib proxies

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/1.2.4/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2.4/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2.4/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-1.2.4)

**Version 1.2.3**

Extends the feature set of TabBuilder for adding multi select controls and introduces additional getters. Also makes it  
possible to directly add dialog descriptions to the dialog registry. This makes it possible to have new strategies for  
dialogs which is required by the new module magnolia-module-blossom-extension-data that adds support for using blossom  
dialogs with node types in the data module.

Added features:

* BLOSSOM-43 Adds TabBuilder.addMultiSelect() addInclude() and addControl()
* BLOSSOM-50 Adds TabBuilder.getContext()
* BLOSSOM-53 Adds TabBuilder.getDialog()

Behind the scenes:

* BLOSSOM-51 Dialog registry accepts dialog descriptions directly and DialogDescriptionBuilder is more open to extension

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/1.2.3/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2.3/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2.3/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-1.2.3)

**Version 1.2.2**

Adds an enhancement to BlossomWebArgumentResolver that allows the content objects for both page and paragraph to be  
used as argument on annotated controllers. Also fixes two reported bugs.

Added features:

* BLOSSOM-36 Support both page and paragraph content as arguments on controllers

Bug fixes:

* BLOSSOM-37 Empty dialog when editing version 0.5 content or using Magnolia before 4.3.2
* BLOSSOM-40 Model not emptied after rendering of paragraph completes

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/1.2.2/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2.2/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2.2/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-1.2.2)

**Version 1.2.1**

Bug fix release. Fixes two issues related to context path and bridging from content rendering into Spring's DispatcherServlet and an issue with context path and uuid redirects.

Bug fixes:

* BLOSSOM-33 - Simulated forwards should use the webapps context path
* BLOSSOM-34 - Simulated forward attributes are not visible after an include
* BLOSSOM-35 - Redirects based on uuid should include contextPath

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/1.2.1/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2.1/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2.1/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-1.2.1)

**Version 1.2**

Big enhancements to the feature set and fixes two important bugs. Most notably added support for having beans that are  
configured in the repository with support for reloading them transparently when their configuration changes. The feature  
builds on an extension to Content2Bean that allows dependency injection, lifecycle events and applying AOP proxies using  
the ApplicationContext. As for the ApplicationContext there's a new feature that brings in singletons from Magnolia so  
that they can be specified as dependencies in bean definitions or used for autowiring. Also new in this release is a  
number of enhancements for working with content in controllers even easier.

Added features:

* BLOSSOM-14 Make possible to set a SaveHandler for a dialog
* BLOSSOM-19 Support for DI and AOP in Content2Bean
* BLOSSOM-20 FactoryBeans for creating beans configured in repository with support for observed reloading
* BLOSSOM-23 WebArgumentResolver for Content, User, MgnlUser, AggregationState and Context
* BLOSSOM-24 FactoryBean that exposes magnolia singletons in ApplicationContext
* BLOSSOM-29 ViewResolver for sending redirects based on uuid
* BLOSSOM-30 Support for filter and servlet proxies that defer initialization until Magnolia has finished update/install phase
* BLOSSOM-31 @Available should be overridable in subclasses

Bug fixes:

* BLOSSOM-16 BlossomDispatcherServlet fails to render when handlerPath is a subset of the requestURI
* BLOSSOM-18 Multipart support doesn't work with Spring 3

Behind the scenes:

* BLOSSOM-22 Move BlossomFilter deeper into filter chain to make sure that AggregationState is properly filled in during pre execution
* BLOSSOM-28 Upgrade to parent pom version 19
* BLOSSOM-32 Anonymous dialogs for paragraphs should be registered in repository like normal dialogs

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/1.2/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.2/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-1.2)

**Version 1.1.1**

Small feature and bug fix release. The javadoc documentation has been greatly improved and extended. You can now  
annotate a method in your templates to control where the template is available. Blossom now also support multipart  
requests by exposing Magnolias built in support through the Spring APIs. VirtualURIMappings are now discovered  
and automatically exported. There's also support for declaring them using an annotation @VirtualURIMapper. Also, if  
you've done customizations to add more argument types to methods for creating and validating dialogs and tabs you'll  
be pleased to see that this extension point is now more comprehensible and easy to use.

Added features:

* BLOSSOM-1 Support for @Available to control where a template can be used
* BLOSSOM-4 Support for exposing VirtualURIMappings and classes using @VirtualURIMapper
* BLOSSOM-11 Support for multipart requests, bridges Springs abstraction onto Magnolia multipart processing

Bug fixes:

* BLOSSOM-5 NPE while detecting dialog factories and encounters abstract bean
* BLOSSOM-7 ContextLoader is not released correctly on module reload
* BLOSSOM-12 The autodetected node is created only when needed so it won't be considered as a dialog when its empty
* BLOSSOM-13 Use of configNode when creating dialogs

Behind the scenes:

* BLOSSOM-8 Upgrade POM expand javadoc and pass checkstyle tests
* BLOSSOM-9 Improve package structuring and extract detection of templates, paragraphs and dialogs to separate classes

We'd like to thank everyone that has invested time in testing and providing feedback. Special thanks goes out to Danilo Ghirardelli, Thomas Duffey, Thomas Kalmar and Åke Argéus.

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/1.1.1/)   
[Javadoc](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.1.1/magnolia-module-blossom/apidocs/index.html)   
[Source cross-reference](https://nexus.magnolia-cms.com/content/sites/magnolia.public.sites/modules/magnolia-module-blossom/1.1.1/magnolia-module-blossom/xref/index.html)   
[Source repository](https://git.magnolia-cms.com/gitweb/?p=modules/blossom/blossom.git;a=tag;h=refs/tags/magnolia-module-blossom-1.1.1)

**Version 1.1**

Blossom is now maintained and supported by Magnolia International Ltd. Package names and XML  
namespace URIs change to reflect this. The Maven artifact is also changed to info.magnolia. Apart from this the release is a minor release with an important bug fix that prevented dialogs from working in Magnolia 4.3.2. Also adds support for initializing Spring within modules, adding radio button controls to dialogs and setting visibility on templates.

Added features:

* Support for properly getting Spring initialization within module install and startup
* Added radio button control to TabBuilder
* Added support for visibility on templates

Bug fixes:

* Fix for Magnolia 4.3.2, the paragraph edit dialog was broken due to changes in Magnolia
* TabBuilder.addFile no longer throws RepositoryException
* Methods in TemplateDescriptionBuilder changed from private to protected
* Tabs added to dialogs have their name set to match their label

API Changes:

* Maven artifact groupId is now info.magnolia
* Package names changed from se.issi.magnolia. to info.magnolia.
* Spring namespace changed from  http://www.issi.se/schema/blossom  to  http://www.magnolia-cms.com/schema/blossom

Behind the scenes:

* Custom paragraph select dialog is removed
* Blossom pre-execution filter enabled by default on install

[Download](https://nexus.magnolia-cms.com/content/repositories/magnolia.public.releases/info/magnolia/magnolia-module-blossom/1.1/)

**Version 1.0**

As of version 1.0 Blossom gives you a much bigger feature set and some of the functionality in the previous version has matured. Unfortunately this means that some changes have to be made when upgrading. But don't worry, your content does not have to be touched.

Added features:

* Support for exposing controllers as Templates
* Support for creating all types of dialogs with code
* Localization support for paragraphs, dialogs and templates
* Rendering of views is performed using the Magnolia render chain, supports both JSP and Freemarker
* Ordering of dialog tabs is customizable with @TabOrder
* The locale used by Magnolia can be made visible to Spring using MagnoliaLocaleResolver
* Magnolia's localization mechanism can be bridged into Spring using MagnoliaMessageSource
* JSP tag pecid-input is now also available as pecidInput

Behind the scenes:

* Blossom now exposes controllers are normal paragraphs instead of using a placeholder paragraph  
  that delegates rendering to the controller. This is a much cleaner design and allows migration  
  of existing content and paragraphs into Blossom.
* Customization of the paragraph select dialog is no longer necessary.

API Changes:

* Paragraphs are no longer named after the handlerPath they're mapped to. That is, a controller mapped to /text that was previously referred to in templates as /text is now simply text. This had to be done in order to support migration of existing configured paragraphs into Blossom. Using a forward slash / in a paragraph name was never a good idea in the first place. You don't have to specify a name on your paragraph, it will automatically use the handlerPath stripped from any slashes as its name.
* @ParagraphTitle is renamed to @Paragraph and moves to se.issi.magnolia.module.blossom.annotation
* @ParagraphDescription moves to se.issi.magnolia.module.blossom.annotation
* DialogTabBuilder is renamed to TabBuilder
* DialogTabBuilder.addText changes to TabBuilder.addEdit
* Methods in DialogTabBuilder no longer throw RepositoryException
* Methods in DialogTabBuilder now return the control they create
* @DialogTabFactory is renamed to @TabFactory and moves to se.issi.magnolia.module.blossom.annotation
* Ordering of tabs in dialogs are now set using the @TabOrder annotation
* <blossom:handler-mapping /> is deprecated and removed
* <blossom:paragraph-registry /> is renamed to <blossom:configuration />
* Extension hooks in DefaultBlossomParagraphRegistry have new arguments
* ValidatingParagraph is removed and replaced with @DialogValidator
* Interface DialogCreator has changed and DefaultDialogCreator changes accordingly.

[Download](https://sourceforge.net/projects/magnoliablossom/files/Magnolia%20Blossom%201.1/)

**Version 0.5**

Initial release.

[Download](https://sourceforge.net/projects/magnoliablossom/files/Magnolia%20Blossom%200.5/)

**Resources**

Wiki - [Blossom examples and articles](http://wiki.magnolia-cms.com/label/WIKI/blossom)

Blog - [Tobias Mattsson](http://tobias-mattsson-magnolia.blogspot.com/)

Tutorial - [Spring Framework and Magnolia: Creating Complex Java-based Websites](http://www.developer.com/java/web/spring-framework-and-magnolia-cms-creating-complex-java-based-websites.html)

Conference talk - [Integrating Magnolia with Spring Framework using Blossom at Magnolia Conference 2010](http://www.magnolia-cms.com/community/conference/2010/program/presentation-day/blossom)

Webinar - [US Navy Integrates Spring Applications with Flying Colors Thanks to Web CMS](http://www.magnolia-cms.com/resource-directory/webinars/webinar-navy)

* [module](https://documentation.magnolia-cms.com/label/DOCS/module)