Integration points

Related topics:

- Integration
- Integration modules

Magnolia is extremely flexible and built for easy integration. You can connect Magnolia to multiple third-party systems, tools and data sources.

Magnolia can be integrated in various ways ranging from filters and servlets to full Java framework integration.

- REST
  - REST service and API
  - REST Client
- Connecting to non-JCR data
- Custom filter
- Custom servlet
- Forwarding a request
- Custom renderer
- Models
- Direct calls to rendering engine
- Web services, AJAX and snippets
- Reading content directly
- Apps and actions
- Templates and front-end resources
- Form processors
- Groovy scripts
- Custom Java classes

REST

REST service and API

Magnolia REST module provides preconfigured REST endpoints for nodes, properties and commands allowing you to manipulate content through a Web API. You can create, read, update and delete nodes and properties in any JCR workspace. You can also execute commands to activate, export and import content.

Use the delivery endpoint API and its search, filter and sorting features to get exactly the content you need, including linked content with the reference resolving feature.

You can control access to REST resources with Magnolia's role-based security system and configure your desired CORS behavior.

The REST modules also facilitates the creation of custom endpoints: developers can create and maintain their own custom delivery APIs via YAML configuration.

See also REST API

REST Client

Use the Magnolia REST Client to centrally manage and supply access to external systems. Provide a REST Client with the remote URL and necessary credentials and keys, then developers can access the client in templates and throughout the system.

The REST client module enables easy creation, configuration and usage of REST clients. The module uses RESTEasy, provides an app to test clients and includes restfn templating functions that give access to REST clients within FreeMarker template scripts. You also can obtain a REST client within Java to build apps, custom fields for dialogs and similar things.

Connecting to non-JCR data

Content app with non-JCR content tutorial provides a step-by-step guide to creating an app that operates on data stored outside the JCR repository. The tutorial shows you how to create a custom content connector which is key to connecting to external content. The example app uses the content app framework but stores data on the local file system. It is an example of connecting to a custom data source. Editors don't have to leave Magnolia in order to work on third-party data.
Custom filter

Additional filters can be added in the filter chain or web.xml file. If you add them in the filter chain you benefit from Magnolia features such as security and caching. The additional filter can take over the handling of the request or delegate it to the remaining filter chain.

Filter chain integration

Custom filters explains how to create and configure a custom Filter.

Custom servlet

An independent servlet can be registered in the servlet filter. The easiest way to do so is to register the servlet in the module descriptor. If the servlet maps to the request it will be responsible for rendering.

Example: Responding to a request with a servlet:

1. A request arrives to the Magnolia filter chain.
2. The request is mapped to a servlet (red box).
3. The servlet handles the response.
Responding to a request with a servlet

**Registering a servlet** explains how to create and register a servlet.

**Forwarding a request**

As the Magnolia main filter is registered on forward, any servlet or filter can internally forward to a Magnolia page. This is normally used if the servlet executes some business logic but the result should be rendered in a Magnolia page. This strategy is often used to combine Magnolia with other frameworks. The servlet typically puts the results into the request attributes which are exposed in Magnolia through the context.

**Example:** A servlet within Magnolia (blue box) forwards the request to another page:

1. A request arrives to the Magnolia filter chain.
2. The request is mapped to a servlet.
3. The servlet handles the response.
4. A page is rendered.
Forwarding a request

Custom renderer

A custom renderer is needed if you would like to support different template script languages or want to perform additional processing. The template definition will then be of custom type. If you need to delegate the rendering to a servlet you can use a template definition of type `servlet`. Creating a custom renderer explains how to create your own renderer.

Models

Models can be implemented in Java or in JavaScript following the Light development approach.

Model is a Java program that holds your business logic. The model can do any task required (such as execute a Google search or calculate an insurance policy).

A template definition can be used to further define a modelClass.

Process:

- The model holds the results of the logic execution and passes them to a renderer.
- The renderer includes the results on the page as instructed by the template script.
- The script contains placeholders for content attributes (such as headings, images and the results of the business logic execution) and tells the renderer where to place the content.

With template models, templates can also run server-side code (JavaScript or Java) before they are rendered. This gives developers a useful integration point. For example, a template model can make a REST call to an external system to either retrieve or store some information. Retrieved information is then displayed during the rendering of the component. REST calls can also be made directly from the browser to the external system, but with server-side calls no sensitive information is exposed to the browser, such as keys or connection information.

See Model and How to work with JavaScript models.

Direct calls to rendering engine
Any Java code can call the rendering engine to let it render pages or paragraph snippets. The caller will pass a `Writer` object to which the result is written. If you use a `ByteArrayOutputStream` the result can be read into a `String`.

This is only possible if all involved template scripts are FreeMarker scripts.

Web services, AJAX and snippets

External applications can make requests to Magnolia to get rendered content/snippets. Magnolia supports for instance direct requests to components.

For example, to render just the teasers on the About page, request the teaser component's own URL:

https://demopublic.magnolia-cms.com/travel/about/main/00.html

Reading content directly

If the Magnolia context is initialized, which is for instance the case if the context filter was executed, you can read content directly from it:

```
MgnlContext
.getJCRSession("String workspaceName").
```

Apps and actions

Apps and actions are the important integration points in the Magnolia UI. The Magnolia UI concept is to have one app for one task or domain.

An integration module often supplies an app to interact with the external system. The apps in Magnolia can be created, modified and rearranged all via configuration.

Templates and front-end resources

Developers can supply front-end resources as files and use `themes` and the `resfn` module to include these resources on web pages.

Templates allow developers to provision reusable page layouts and components for content authors. Templates allow developers to integrate existing third party or in-house developed component libraries into Magnolia.

You can also use component templates to integrate with the massive ecosystem of SaaS-based web features such as web forms, maps, live chat, commenting features, social widgets, and so on. For these front-end integrations, the template script must include the embed code provided by the service.

Form processors

Developers can handle form submissions with server-side code (Javascript or Java). Form processors can be used to send emails to the website users and to administrators, to make REST calls to external systems or to store content to a repository.

Groovy scripts

For integration use-cases not covered by other mechanisms, developers have the option to supply code written in groovy.

Magnolia allows groovy scripts to be used anywhere that a Java class can be provided in configuration.

Custom Java classes

Additional integration points are available with Java development. Because Magnolia is written in Java and is open source, you can build your own Magnolia bundle.

With custom Java classes, there is no limit to the customisations you can implement. You can replace stock classes with your own implementation using two mechanisms:

- Supply the classpath in the configuration app. These can be changed while Magnolia is running.
- Supply the class or component in the module configuration files. See Dependency injection and inversion of control.