



# Ask the Experts – Using Sling and Adobe Experience Manager

Scott Macdonald | AEM Community Manager

Lokesh BS | Top AEM Community Member



# Agenda

- What is Sling and cheatsheet - [http://docs.adobe.com/docs/en/cq/5-6-1/developing/sling\\_cheatsheet.html](http://docs.adobe.com/docs/en/cq/5-6-1/developing/sling_cheatsheet.html) (L)
- Sling Selectors within AEM
- Default Sling Post Servlet
- Sling APIs (S)
- Sling Models (L)

# What is Sling

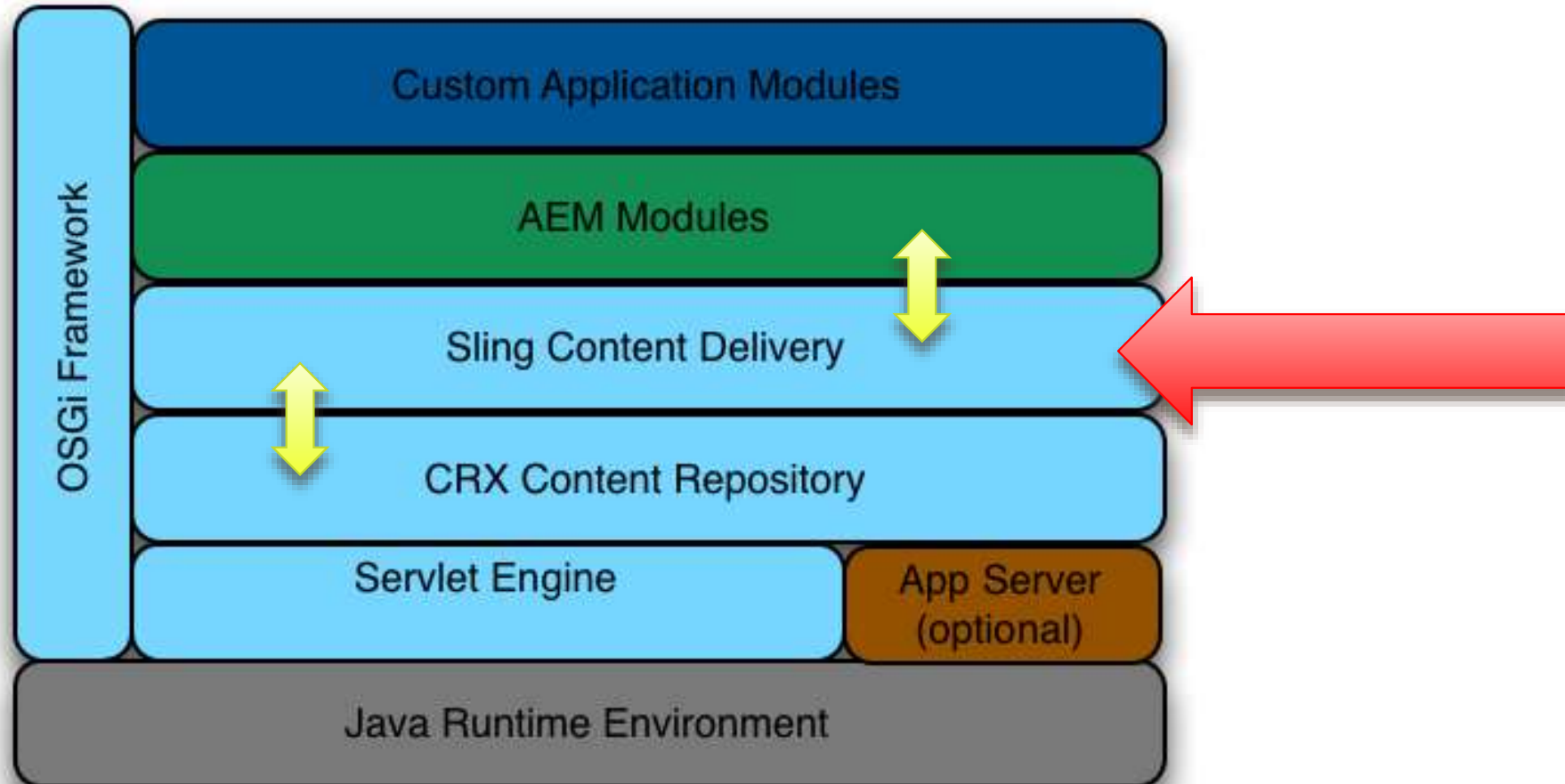
- REST based web framework
- Content-driven, using a JCR content repository
- A resource is a central part of Sling and it assumes everything in the JCR is a resource
- Powered by OSGi
- Scripting inside, multiple languages (JSP, server-side javascript)
- Apache Open Source project

\* With some exceptions

- **OSGi** —Sling applications are built as OSGi bundles and makes heavy use of a number of OSGi core and services.
- **Sling API** — The Sling API extends the Servlet API and provides more functionality to work on the content.
- **Request Processing** — Sling takes a unique approach to handling requests in that a request URL is first resolved to a resource, then based on the resource (and only the resource) it selects the actual servlet or script to handle the request.
- **Resources** —Sling considers everything in the JCR as a Resource. Based on the resource, a servlet or script is accessed to actually handle the request.
- **Servlets and Scripts** — Servlets and Scripts are handled uniformly in that they are represented as resources themselves and are accessible by a resource path.

\* With some exceptions

# AEM and Sling





understanding

Apache Sling

script resolution

1 HTTP Request

Method  
GET  
Path  
/wiki/Sling  
Extension  
.edit.html  
Suffix  
/richtext?  
Query Parameters  
simple=true HTTP/1.1

2 Content Resolution

/wiki/Sling

Node has properties

Property	Value
title	Sling Intro
body	<div>Hello World</div>
sling:resourceType	wiki/page

3 Get Resource Type

use: sling:resourceType=  
wiki/page  
Resource Type  
fallback: sling:resourceSuperType=  
«null»  
last resort: jcr:primaryType=  
nt:file

4 Script Locations

either: /apps/wiki/page/  
or: /libs/wiki/page/

5 Script Names

Best match:  
edit.html.esp  
Selector\*Extension  
edit.esp  
Selector  
html.esp  
Extension  
Worst match:  
GET.esp  
Method

6 Script

```
<% log.info("Executing my script");  
if (request.getRequestParameter("simple")) {  
    response.sendRedirect("http://localhost/");  
} %>  
<html>  
  <head><title><%=currentNode.title %></title></head>  
  <body>  
    <h1><%=resource.getPath() %></h1>  
    <% out.println(reader.toString()); %>  
    <% sling.include(resource.getPath() + "content",  
      "replaceSelectors= edit"); %>  
  </body>  
</html>
```

7 Include Options

```
sling.include("path",  
  "forceResourceType= wiki/body, (Powerful)  
  replaceSuffix= xhtml,  
  addSelectors= foo.bar");
```

# Using the SlingPostServlet

The **SlingPostServlet** is a front-end to JCR operations. To select a JCR operation to execute, the `:operation` request parameter is used.

Out of the box, the **SlingPostServlet** supports the following operations:

- **property not set or empty** -- Create new content or modify existing content
- **delete** -- Remove existing content
- **move** -- Move existing content to a new location
- **copy** -- Copy existing content to a new location
- **import** -- Import content structures from JSON/XML/Zip
- **nop** -- Explicitly requests to do nothing and just sets the response status
- **checkin** - Check in a versionable node
- **checkout** - Check out a versionable node

# SlingPostServlet Examples

- Create a node named *mynode*. Set (or overwrite) *title* and *body* properties.

```
<form action="/mynode" method="POST">  
  <input type="text" name="title">  
  <textarea name = "body"  
</form>
```

- Create a node below /mynode and make it the first child

```
<form action="/mynode/" method="POST">  
  <input type="text" name="dummy">  
  <input type="hidden" name=":order">  
    <vaule = "first"  
</form>
```



## SlingPostServlet Examples (Continued)

- Delete a node.

```
<form action="/node/" method="POST">  
  <input name=":operation" type="hidden">  
    <value = "delete" >  
</form>
```

- Create a node below /mynode and make it the first child

```
<form action="/old/node/" method="POST">  
  <input type="hidden" name=":operation" value="move">  
  <input type="hidden" name=":dest" value = "/new/place">  
</form>
```

# SlingPostServlet Cheatsheet

**Using the SlingPostServlet**  
this is the default handler for your  
POST requests. It can do nearly  
anything.

```
<form action="/mynode" method="POST">
  <input type="text" name="title">
  <textarea name="body">
</form>
```

Create or update /mynode, set title and body. Set lastModified and lastModifiedBy automatically

```
<form action="/mynode/" method="POST">
  <input type="text" name="dummy">
  <input type="hidden" name=":order"
    value="first">
</form>
```

Create new node below /mynode and make it the first child (also valid: last, before x, after x, 3, 7, 9).

```
<form action="/node" method="POST">
  <input name=":operation"
    type="hidden" value="delete">
</form>
```

Delete /node

```
<form action="/node" method="POST">
  <input type="hidden"
    name=":operation" value="delete">
  <input type="hidden"
    name=":applyTo" value="/node/one">
  <input type="hidden"
    name=":applyTo" value="/node/two">
</form>
```

Delete /node/one and /node/two

```
<form action="/mynode/" method="POST">
  <input type="hidden"
    name=":name" value="new_node">
  <input type="hidden"
    name=":nameHint" value="new node">
</form>
```

Create new node below /mynode, use name or name hint. Set created and createdBy automatically

```
<input type="text" name="customer">
<input type="hidden" value="John Doe"
  name="customer@DefaultValue" >
<input type="hidden" name="title@Delete">
```

Take default value for customer property, remove the title property

```
<form action="/old/node" method="POST">
  <input type="hidden"
    name=":operation" value="move">
  <input type="hidden"
    name=":dest" value="/new/place">
</form>
```

Move /old/node to /new/place

```
<input type="text" name="date1"
  value="2008-06-13T18:55:00">
<input type="text" name="date2">
<input type="hidden"
  name="date2@TypeHint" value="Date">
<input type="hidden" value="nt:file"
  name="./uploaded/jcr:primaryType">
```

Guess property type from date pattern, set property type explicitly and set node type explicitly

```
<form action="/old/node" method="POST">
  <input type="hidden" name=":operation"
    value="copy">
  <input type="hidden" name=":dest"
    value="/new/place">
  <input type="hidden" name=":replace"
    value="true">
</form>
```

Copy /old/node to /new/place and replace the existing node there.

```
<input type="text" name="oldtitle">
<input type="hidden" value="oldtitle"
  name="newtitle@ValueFrom">
```

Get value for property title from field oldtitle

```
<input type="hidden" value="/node/prop"
  name="title@CopyFrom">
```

Copy property title from other node's property

● Day

# SlingPostServlet DEMO



- The *Sling API* defines a presentation framework to build Web Applications.
- The Sling API is resource centric. That is, the request URL does not address a servlet or a portlet but a resource represented by an instance of the **`org.apache.sling.api.resource.Resource`** interface
- The Sling API uses the URL to select a resource to be delivered. You can retrieve content from the AEM JCR using Sling APIs. You can use the Sling API from within an OSGi bundle to retrieve a resource from within the AEM JCR.
- To use the Sling API from within an OSGi component, you inject an **`org.apache.sling.api.resource.ResourceResolverFactory`** instance into the service
- When using the Sling API to query the AEM JCR, you have access to helper methods that are not available when using the JCR API. For example, the **`adaptTo`** method converts a resource into an appropriate object representing a certain aspect of this resource
- For example to translate a `Resource` object to the corresponding **`Node`** object, you invoke the **`adaptTo`** method:

```
Node node = resource.adaptTo(Node.class) ;
```

# Sling API “Live” Coding

# Sling Models



# Design Goals

- Entirely annotation driven. "Pure" POJOs
- Use of standard annotations
- OOTB supports resource properties, SlingBindings, OSGi services
- Adapt multiple objects
- Support both classes and interfaces.
- Work with existing Sling infrastructure

# First Model

- POJO Class

```
@Model(adaptables=Resource.class)
public class TestModel{
    @Inject
    private String propertyName;
}
```

- Interface

```
@Model(adaptables=Resource.class)
public interface TestModel{
    @Inject
    String getPropertyName();
}
```

- Required

In Manifest file:

```
<Sling-Model-Packages>
    org.apache.sling.models.mymodels
</Sling-Model-Packages>
```

# Annotations

## **@Model**

declares a model class or interface

## **@Inject**

marks a field or method as injectable

## **@Named**

declare a name for the injection (otherwise, defaults based on field or method name).

## **@Optional**

marks a field or method injection as optional

## **@Source**

explicitly tie an injected field or method to a particular injector (by name). Can also be on other annotations.

## **@Filter**

an OSGi service filter

## **@PostConstruct**

methods to call upon model option creation (only for model classes)

## **@Via**

use a JavaBean property of the adaptable as the source of the injection

## **@Default**

set default values for a field or method

## Model Instantiation

- **adaptTo()**

```
TestModel model = resource.adaptTo(TestModel.class)
```

- **ModelFactory**

```
try {  
    TestModel model = modelFactory.createModel(resource,  
TestModel.class);  
} catch (Exception e) {  
    //exception  
}
```



**Adobe**