



**adaptTo()**

APACHE SLING & FRIENDS TECH MEETUP  
BERLIN, 22-24 SEPTEMBER 2014

Lazy AEM developer  
Feike Visser, Adobe, @heervisscher

# Sling Models & Sightly in Action

# Introduction to Sling Models / Sightly

# What is Sling Models

- Creating an adaptable class from a POJO by annotations

```
Resource r = getResource();  
return r.adaptTo(YourCustom.class);
```

```
@Model(adaptables = Resource.class)  
public class YourCustom {  
    ...  
}
```

# What is Sling Models

## ■ The ‘old-days’....

```
@Component
@Service
@Properties({
    @Property(name = "adaptables", value = {"Resource" }),
    @Property(name = "adapters", value = {"YourCustom" })
})
public class MyAdapterFactory implements AdapterFactory{
    public <AdapterType> AdapterType
        getAdapter(final Object adaptable, Class<AdapterType> type){
            return new MyClassAdapter(adaptable);
        }
}
```

# What is Sling Models

## ■ Injecting

```
@Model(adaptables = Resource.class)
public class YourCustom {
    @Inject // we expected always an email-property
    private String email;

    @Inject @Optional // firstname can be empty
    private String firstName;

    // read property 'surname' if empty, use 'empty'
    @Inject @Named("surname") @Default(values="empty")
    private String lastName;
}
```

# What is Sling Models

```
@Model(adaptables = Resource.class)
public class YourCustom {
    @Inject // OSGi Service
    private Externalizer externalizer;

    @PostConstruct
    protected void init() {
        // gets executed after the class is created
        // set to protected, so it can be unit-tested
    }
}
```

# What is Sling Models

```
@Model(adaptables = Resource.class)
public class YourCustom {

    @Self //(available in 1.1.10)
    private Resource resource;

    public YourCustom(Resource resource) {
        // to get access to the adaptor
        this.resource = resource;
    }
}
```



# Sling Models

Before



After



# What is Sling Models

- Available in AEM6
- Can be installed in CQ5.6.1
- <http://sling.apache.org/documentation/bundles/models.html>
- Updates: <http://bit.ly/sling-models-package>

- Use HTML5-attributes to implement basic logic

```
<div data-sly-test="${wcmmode.edit}">  
  Show this in edit mode...  
</div>
```

- Standard bindings are available

```
<h2>${currentPage.title}</h2>
<h3>${pageProperties.jcr:title || 'No title'}</h3>
<h4>${properties['address/officeName']}</h4>
<ul data-sly-list="${currentPage.listChildren}">
  <li>Page title: ${item.title}</li>
</ul>
```

- Basic comparisons are available

```
<h2>${currentPage.title}</h2>
```

```
<div data-sly-test="${ (currentPage.title == 'test') }">
```

```
    // show the parsys when the title is test
```

```
    <div data-sly-resource="${ 'par' }
```

```
        @ resourceType='foundation/components/parsys'">
```

```
    </div>
```

```
</div>
```

## ■ What about non-basic logic?

```
<div data-sly-use.sample="com.yourproject.YourComponent">  
    // display the value from the Java-class  
    ${ sample.myCustomValue }  
</div>
```

1. Class extends WCMUse-class
2. Class implements Use-interface
3. Class is adaptable from Resource
4. Class is adaptable from Request

# What is Sightly

JSP



Sightly



# What is Sightly

- Available since AEM6
- Name your component file .html
- No need to include anything like 'global.jsp'
- No taglibs
- Compiled code is in /var/classes/sightly
- <http://docs.adobe.com/docs/en/aem/6-0/develop/sightly.html>



# How Sling Models and Sightly meet?

# How Sightly and Sling Models meet?

- data-sly-use does the trick with a class adaptable from Resource or Request

```
<div data-sly-use.myClass="mysite.myproject.HeaderComponent">
```

```
  ${ myClass.fullName }
```

```
</div>
```

# How Sightly and Sling Models meet?

```
@Model(adaptables = Resource.class)
public class HeaderComponent {

    @Inject @Default(values="Feike")
    public String firstName; // maps to property firstName

    public String fullName;

    @PostConstruct
    protected void init() {
        fullname += firstName + " Visser";
    }
}
```

# How Sightly and Sling Models meet?

- Can we pass in parameters?

```
<div data-sly-use.myClass="${ `mysite.myproject.HeaderComponent` @  
param1=currentPage, param2='advanced' }">
```

```
  ${ myClass.fullName }
```

```
</div>
```

Only works if class is adaptable from Request

# How Sightly and Sling Models meet?

```
@Model(adaptables = SlingHttpServletRequest.class)
public class HeaderComponent {

    @Inject
    public Page param1; // maps to param1 parameter

    @Inject
    public String param2; // maps to param2 parameter

}
```

# How Sightly and Sling Models meet?

- Can we re-use the bindings?

```
${ currentPage.title }  
${ pageProperties.jcr:title }
```

```
@Model(adaptables = SlingHttpServletRequest.class)  
public class HeaderComponent {  
  
    @Inject  
    public Page currentPage; // maps to currentPage binding  
  
}
```

# How Sightly and Sling Models meet?

- Common sense still applies, only write code if needed

```
<div data-sly-use.person="project.Person">  
    ${person.firstName}  
</div>
```

```
<div>  
    ${properties.firstName}  
</div>
```

## Example: Lat-Long component



# Lat-long component

- User enters address, lat+long is displayed

```
<div class="address" data-sly-use.geocode="components.LatLongComponent">  
  <div class="latlong">  
    Lat : ${geocode.coordinates.lat},  
    Long : ${geocode.coordinates.lng}  
  </div>  
</div>
```

# Lat-long component

```
@Model(adaptables=Resource.class)
public class LatLongComponent {
    @Inject @Named("address") @Default(values=DEFAULT)
    protected String addressDescription;

    @Inject // Injecting Service here
    private GeocodeProvider geocode;

    @PostConstruct
    protected void init() throws AddressException {
        coordinates = geocode.geocode(addressDescription);
    }
}
```

# Lat-long component

- Sightly supports different ways to access props

```
<div class="address" data-sly-use.geocode="components.LatLongComponent">
  <div class="latlong">
    Lat : ${geocode.coordinates.lat},
    Long : ${geocode.coordinates.getLng}
  </div>
  <div class="addressDescription"
    data-sly-text="${properties.address || geocode.DEFAULT}">
  </div>
</div>
```

## Example: Absolute URL

- Render the absolute url of the current-page

```
<meta
  data-sly-use.externalizer="components.ExternalUrl"
  property="og:url"
  content="${externalizer.absoluteUrl @ context='uri'}.html" />
```

# Absolute URL

```
@Model(adaptables = SlingHttpServletRequest.class)
public class ExternalUrl {
    @Inject
    private Externalizer externalizer;

    @Inject // To get the currentPage
    private PageManagerFactory pageMan;

    private SlingHttpServletRequest request;

    public ExternalUrl(SlingHttpServletRequest request) {
        // needed for the Externalizer
        this.request = request;
    }
}
```

# Absolute URL

```
@Model(adaptables = SlingHttpServletRequest.class)
public class ExternalUrl {
    @PostConstruct
    protected void init() {
        String path = getCurrentPage().getPath();
        absoluteUrl = externalizer.absoluteLink(request, "http", path);
    }

    private Page getCurrentPage() {
        PageManager pm = pageMan.getPageManager(request.getResourceResolver());
        return pm.getContainingPage(request.getResource());
    }
}
```

# This is not lazy enough

- Have to code the `getCurrentPage()`
- Fixed to `currentPage`
- Not reusable enough



# Absolute URL

```
@Model(adaptables = SlingHttpServletRequest.class)
public class ExternalUrl {
    @Inject
    private Externalizer externalizer;

    @Inject // Using the bindings from Sightly (${currentPage.path})
    protected Page currentPage;

    @Inject @Optional // parameter from data-sly-use
    protected String path;
}
```

# Absolute URL

```
@Model(adaptables = SlingHttpServletRequest.class)
public class ExternalUrl {
    @PostConstruct
    protected void init() {
        String relPath = currentPage.getPath();
        if ( path != null) { // check if there is a parameter
            relPath = path;
        }
        absoluteUrl = externalize(relPath);
    }

    protected String externalize(String path) {
        return externalizer.absoluteLink(request, "http", path);
    }
}
```

- Parameters can be specified after the @

```
<meta
  data-sly-use.externalizer="${ 'components.ExternalUrl'
    @ path=resourcePage.path }"
  property="og:url"
  content="${externalizer.absoluteUrl @ context='uri'}.html" />
```

# Absolute URL (unit-testing?)

@Spy

```
private ExternalUrl externalUrl = new ExternalUrl(null);
```

@Before

```
public void setup() {
```

```
    String path = "/content/adaptTo/example";
```

```
    String extPath = "http://localhost:4502/adaptTo";
```

```
    doReturn(extPath).when(externalUrl).externalize(path);
```

```
}
```

@Test

```
public void testWhenNoPathParameterIsSpecified() {
```

```
    externalUrl.init();
```

```
    Assert.assertNotNull(externalUrl.absoluteUrl);
```

```
}
```

# Bindings, Bindings, Bindings

# Bindings example

- Page-oriented functionality
- Reuse the Sightly bindings
- Expose this via a custom bindings
- Use the binding in your Sling model class

## ■ Custom Page-class

```
@Model(adaptables=Page.class)
```

```
public class MyCustomPage {
```

```
    private Page page;
```

```
    public String getTitle() {
```

```
        return "MyProject : " + page.getTitle();
```

```
    }
```

```
}
```

- Re-use Sightly binding, adding new binding

```
public class CustomBindingProvider implements BindingsValuesProvider {  
  
    @Override  
    public void addBindings(Bindings bindings) {  
  
        if ( bindings.containsKey(WCMBindings.CURRENT_PAGE)) {  
            Page current = (Page) bindings.get(WCMBindings.CURRENT_PAGE);  
            bindings.put("customPage", current.adaptTo(MyCustomPage.class)) ;  
        }  
    }  
}
```



- Make sure your BVP is *\*after\** the Sightly-BVP

```
@Service
@Component(immediate = true)
@Properties({
    @Property(name = "javax.script.name", value = "sightly"),
    @Property(name = "service.ranking", intValue = 100)
})
public class CustomBindingProvider implements BindingsValuesProvider {
    ...
}
```

- New binding available in your components

```
<div>  
    <h1>${customPage.title}</h1>  
</div>
```

- And Injectable in your Sling Model class

```
@Model(adaptables=SlingHttpServletRequest.class)
public class ContentComponent {

    @Inject
    private MyCustomPage customPage;
}
```

# data-sly-template example

# data-sly-template example

- Multiple page layouts
- Can be chosen by the author
- Shows the power for data-sly-template

- Can take parameters, can be in a separate file

```
<template data-sly-template.page="${ withParsys }">
<div class="content">
  <div
    data-sly-test="${ withParsys != 'false' }"
    data-sly-resource="${ 'par' @
resourceType='foundation/components/parsys' }">
    </div>
  </div>
</div>
</template>
```

- data-sly-call invokes the template

```
<div  
    data-sly-test.layoutFile="${ properties.layout || 'layout1.html' }"  
    data-sly-use.layout="${layoutFile}"  
    data-sly-call="${layout.page @ withParsys = 'true'}">  
</div>
```

## (bonus) Sightly questions from the field



# If-then-else?

- Can we do an if-then-else?

```
<div data-sly-test.authorMode="${wcmmode.edit || wcmmode.preview}"></div>  
<div data-sly-test="${!authorMode}"></div>
```

# String-concat?

- String concatenation can be done via @ format

```
<div
  data-sly-test.concat="${ 'This is page {0}, with title {1}' @
    format=[currentPage.name,currentPage.title] }">
  ${concat}
</div>
```

# data-sly-unwrap, friend or enemy?

- data-sly-unwrap *\*can\** be used for declaration

```
<sightly data-sly-use.section="components.YourComponent" data-sly-unwrap/>
```

```
<sightly  
  data-sly-test.localVar="${wcmmode.edit || wcmmode.preview}"  
  data-sly-unwrap/>
```

- Can I increment a counter? No..., via data-sly-use

data-sly-list offers quite a range a helper values

```
<div data-sly-list="${currentPage.listChildren}">
${itemList.index}
${itemList.count}
${itemList.first}
${itemList.last}
${itemList.odd}
${itemList.even}
</div>
```

- Is there (date)-formatting? No, via data-sly-use

```
<div data-sly-use.dateFormat="\${ `yourClass` @ date=dateValue}">  
    ${dateFormat.formattedValue}  
</div>
```

# Questions?