Default Mapping and Rendering (OBSOLETE)

Page Status

This is obsolete content copied from the Sling website on 2010-02-04.

Default Content Mapping and Request Rendering

Page Status

2008-02-13: this page is **out of sync** with the current codebase, needs to be reviewed and updated.

One of the big obstacles in quick adoption of Sling might arguably be the requirement for multiple developments, such as...

1. Creating a `Content` implementation (or decide on reusing an existing implementation)
2. Defining the mapping descriptor to map the repository contents to the `Content` object and vice versa
3. Optionally create a node type defintion file in *CND* format
4. Creating a `Component` implementation (or decide on reusing an existing implementation)
5. Package this all up into an OSGi Bundle for deployment

While these steps make sense in an ideal world we all know does not exist (with the exception of Utopia, but there are no computers in Utopia), helpers for rapid development are needed. These helpers come in the form of usefull defaults on various levels.

Default Content Mapping

When a request is processed by Sling, one step is to resolve the request URL into a `Content` object. This works by checking the request URL for the longest match with an existing JCR repository node. The path of this node is then used to load the `Content` object through the `ContentManager.load(String)` method. If no mapping exists for the given node, an exception is thrown and the request fails.

In such a case of missing content mapping, a default `Content` mapping is defined in the form of the `org.apache.sling.content.jcr.DefaultContent` class. This mapping has the following features:

- The `DefaultContent` class is a `java.util.Map`. Thus all properties may be accessed using the familiar `Map` API.
- All non-protected properties of the node are loaded. Single value properties become scalar objects, while multi value properties become `java.util.List` objects.
- The types of the repository values are mapped according to the JCR specification for mapping between Property types and Java types.
- A few properties have special significance. See below.
- Creating new instances of this class and inserting these into the repository creates nodes of type `nt:unstructured`. When loading instances of this class the actual primary type of the node does not matter.

Property	Type	Description
`path`	String	The path of the node from which the content was loaded. This must not be modified by application programs, unless you are prepared for unexpected behaviour when storing the object.
`primaryType`	String	The primary node type of the (existing) node. This property is purely informational and will never be used when inserting new content or writing back content.
`mixinTypes`	List of String	The mixin node types of the (existing) node. This property is purely informational and will never be used when inserting new content or writing back content. If the node has no mixin node types, this property does not exist.
`sling:componentId`	String	The component ID of the component used to handle requests to this content. This property may be modified by application programs (though you should be aware of the consequences) and is used as the result of the `Content.getComponentId()` method.

Default Component Selection

After having mapped the JCR repository node into the `Content` object the `Component` to actually handle the request must be resolved. This is done by calling the `Content.getComponentId()` method and looking up this component ID in an internal table. If either the `Content.getComponentId()` method returns `null` or no component is registered with the requested component ID a default resolution processing takes place as follows:

1. Let `cid` be the result of calling `Content.getComponentId()`
2. If `cid` is `null`, let `cid` be the result of calling `Content.getPath()` (this is never `null`}
3. Check for a component with the given `cid` and use it if existing
4. Otherwise, remove any leading slash from `cid` and replace slashes by dots and check for a component with this modified `cid` and use it if existing
5. Otherwise, let `cid` be the fully qualified name of the `Content` object class and check for a component with this modified `cid` and use it if existing
6. Otherwise and if `cid` ends with the string *Content*, remove that suffix and check for a component with this modified `cid` and use it if existing
7. Otherwise, append `Component` to the end of `cid` and and check for a component with this modified `cid` and use it if existing
8. Otherwise, let `cid` be the value of the `org.apache.sling.components.DefaultComponent.ID` field and check for a component with this modified `cid` and use it if existing
9. Finally, fail without having found a component to use - this is highly unlikely, though, because the default component is part of the Sling Core bundle and should always be available.

DefaultComponent

The default component first checks whether the request is sent with parameters and will update the `Content` object with the parameters as follows:

- If the `Content` object is a `java.util.Map` (such as is the case for the `DefaultContent`) the properties will directly accessed through the `Map`} API. Otherwise, the `Content` object is wrapped inside a `org.apache.commons.collections.BeanMap` to access the fields through the `Map` API.
- Any properties listed in the `_delete` parameter are removed. The `_delete` parameter may contain a comma-separated list of property names and may occurr multiple times.
- All other parameters are used to set property values, where any existing properties will be replaced and all properties not listed in the parameters remain unmodified. If a parameter occurrs only once a single value property is set, if parameter occurrs multiple times, a multi value property is set as a list of strings. Note, that any data type conversion may happen only by the `BeanMap` as required and thus lead to failure to update a single proeperty.

After the optional update phase, the fields of the `Content` object are written back. Again, the `Content` object is either accessed as a `Map` directly if it is a `Map` or packed in a `BeanMap` otherwise. The format of the output is deduced from the request URL's extension as returned by the `ComponentRequest.getExtension()` method:

Extension	Format
`html`, `htm`	HTML, UTF-8 encoded
`xml`	XML, UTF-8 encoded
`txt`	Plain text, UTF-8 encoded
`properties`	Java Properties file format suitable for a normal properties file
`json`	JSON, UTF-8 encoded

Default Script

The easiest way to develop and deploy a component is to create a scripted component in the repository by just creating a node of type `sling:scriptedComponent` and creating a single JSP script at `jsp/start.jsp` below the component node. After that you can refer to that component by the path of the component node and get the `start.jsp` script called.

For more more elaborate script selection you may of course create more scripts and refer to them below the `sling:scripts` node of the component node.

Rapid Development Primer

To summarize, for rapid development you will have to execute the following steps:

1. Create a `sling:ScriptedComponent` node, for example at `/some/sample/component`
2. Create a JSP script file at `jsp/start.jsp` below that node; that would be `/some/sample/component/jsp/start.jsp` in the example
3. Create one or more nodes of any type, for example `nt:unstructured`, which have a single value string property named `sling:componentId` referring to the component via its path
4. Request the node by typing its path in your browser's address field