HTTP API Framework

Brian Chaikelson

dan mcweeney

Chris Trubiani

Agenda

- Business Context
- HTTP API
- JSON Format
- Annotation Overview
- OAuth improvements
- New Assets API Example

Business Context

Why invest in an HTTP API Framework? Consider benefits for 4 groups:

1. AEM Engineering
   - Can focus on biz-logic over plumbing → more features exposed as HTTP endpoints
2. ISVs – software companies (CRM, ESP, MRM, content marketing, commerce, etc)
   - 6.3 effort to scale connectors built by these partners
   - Integration patterns include custom components, Sling schedulers for data import/export, servlets to get/put assets in the DAM
   - Framework speeds up Adobe’s ability to deliver endpoints useful for connectors
   - High level endpoints limit need for deep AEM expertise (servlets, packages, JCR)

Business Context

Why invest in an HTTP API Framework? Consider benefits for 4 groups:

3. Customers
   - Easier development for ISVs → more connectors available (greater choice)
   - HTTP APIs compatible with OAuth, satisfying security reqs

4. System Integrators
   - Contracted by customers (and sometimes ISVs) to write code
   - Similar benefits as customers and ISVs

Each group benefits from a consistent API design, which the Framework enforces

Thanks

- To Roland Schaer and Paolo Mottadelli for the initial design/development of the pattern and code
- And other who’ve submitted bugs / PRs / Participated in the DL-dev discussion!

HTTP API

- Starting with 6.2 AEM has shipped a API endpoint:
  http://localhost:4502/api.json
- Meant to standardize HTTP calls from customer to customer
- Attempts to segregate Adobe's API space from our customer's implementations
  - Distinct URL space
  - Separate resource types
  - Implemented as a resource provider
- Currently only uses Siren hypermedia type

Siren Format[0]

- “a hypermedia specification for representing entities”
- Entities
  - Children
  - Different serialization
- Links
  - Connections between entities
  - Relationships defined by Web Linking Spec[1]
- Actions
  - Things that can be done to the entity
  - Maps to HTTP Verbs + some Sling-ish additions
- Properties
- Classes

Sample API.json output

```
{
    "class": [
        "core/services"
    ],
    "links": [
        {
            "rel": [
                "self"
            ],
            "href": "http://host:port/api.json"
        },
        {
            "rel": [
                "content"
            ],
            "href": "http://host:port/api/content.json"
        },
        {
            "rel": [
                "assets"
            ],
            "href": "http://host:port/api/assets.json"
        }
    ],
    "properties": {
        "name": "api"
    }
}
```

Sample API.json output

```json
{
    "class": [
        "core/services"
    ],
    "links": [
        {
            "rel": [
                "self"
            ],
            "href": "http://host:port/api.json"
        },
        {
            "rel": [
                "content"
            ],
            "href": "http://host:port/api/content.json"
        },
        {
            "rel": [
                "assets"
            ],
            "href": "http://host:port/api/assets.json"
        }
    ],
    "properties": {
        "name": "api"
    }
}
```

Each one of these links represents a “category” in the API space.
A category allows developers to have various POJOs representing resources

ApiModel

```
@ApiModel(category="assets",
type={"aem-io/assets/fragment"},
resourceType="dam/types/contentfragment")
```

ApiModel

```
@ApiModel(category="assets",
type={"aem-io/assets/fragment"},
resourceType="dam/types/contentfragment")
```

Adds this model to the category.

ApiModel

```
@ApiModel(category="assets",
type={"aem-io/assets/fragment"},
resourceType="dam/types/contentfragment")
```

Gives this model a which can be different from the resource types in AEM. This allows developers to surface another categorization. This aligns with Siren’s “class” list.

ApiModel

```
@ApiModel(category="assets",
type={"aem-io/assets/fragment"},
resourceType="dam/types/contentfragment")
```

This binds this model to a resource type. If the resource has a sling:resourceType with this value, this model will be used when serializing. This is the simplest binding you can define. The other is a ModelLookup that can allow the developer to supply binding logic.

ApiRoot

```
@ApiRoot(baseResource = "/content/dam")
```

Marks this model as the Root for the “category”

Instructs the framework to use this resource as this path for its starting point. Allows you to inject a different resource into the Pojo when you are viewing the “root” of the API category

ApiEntities

```
@ApiEntities
public Iterable<Resource>
getChildren() {
    return children;
}
```

Can be placed on a method. It must return an iterable of Resources.

```
"entities": [
  {
    "class": [
      "aem-io/assets/folder"
    ],
    "links": [
      {
        "rel": [
          "self"
        ],
        "href":
"http://host:port/api/assets/content/dam/projects
.json"
      }
    ],
    "rel": [
      "child"
    ],
    "properties": {
      "name": "projects"
    }
  },
```

ApiLink

```java
@ApiLink(rel = "root")
public String root = "/api/assets.json";
```

Can be placed on a property or method.
It can return a single string value or a
Iterable of Strings

```
"links": [
   {
     "rel": [
       "root"
     ],
     "href": "http://host:port/api/assets.json"
   },
   {
     "rel": [
       "assets"
     ],
     "href": "http://host:port/api/assets.json"
   },
   {
     "rel": [
       "self"
     ],
     "href":
"http://host:port/api/assets/content/dam.json"
   },
   {
     "rel": [
       "parent"
     ],
     "href":
"http://host:port/api/assets/content.json"
   }
 ],
```

ApiLink

```
@ApiLink(rel = "root")
public String root = "/api/assets.json";
```

```
"links": [
    {
      "rel": [
        "root"
      ],
      "href": "http://host:port/api/assets.json"
    },
    {
      "rel": [
        "assets"
      ],
      "href": "http://host:port/api/assets.json"
    },
    {
      "rel": [
        "self"
      ],
      "href":
"http://host:port/api/assets/content/dam.json"
    },
    {
      "rel": [
        "parent"
      ],
      "href":
"http://host:port/api/assets/content.json"
    }
]
```

If you need multiple relations just add more properties with the same target value. A link can also return an iterable of Strings so you can point one relation to many places.

ApiLink

```java
@ApiLink(rel = "root")

public String root = "/api/assets.json";
```

```json
"links": [
  {
    "rel": [
      "root"
    ],
    "href": "http://host:port/api/assets.json"
  },
  {
    "rel": [
      "assets"
    ],
    "href": "http://host:port/api/assets.json"
  },
  {
    "rel": [
      "self"
    ],
    "href":
"http://host:port/api/assets/content/dam.json"
  },
  {
    "rel": [
      "parent"
    ],
    "href":
"http://host:port/api/assets/content.json"
  }
]
```

ApiProperty

```
@ApiProperty(name = "cq:name")
public String getName() {
    return “name”
}
```

```
  "rel": [
      "parent"
    ],
    "href":
"http://radiohead.corp.adobe.com:4502/api/assets/content/dam/geom
etrixx-outdoors/brand.json"
   }
  ],
"properties": {
   "cq:parentPath": "/content/dam/geometrixx-outdoors/brand",
   "name": "brand_1_c02.jpg",
   "cq:name": “name",
   "srn:paging": {
    "total": 1,
    "limit": 20,
    "offset": 0
   },
   "metadata": {
    "dc:format": "image/jpeg",
    "xmp:CreatorTool": "Adobe Photoshop CS6 (Macintosh)",
    "dc:modified": "2014-03-18T13:40:37.482+02:00"
   },
   "related": {

   }
  },
```

ApiProperty

```
@ApiProperty(name = "cq:name")
public String getName() {
    return “name”
}
```

```
 "rel": [
    "parent"
  ],
  "href":
"http://radiohead.corp.adobe.com:4502/api/assets/content/dam/geom
etrixx-outdoors/brand.json"
 }
 ],
"properties": {
  "cq:parentPath": "/content/dam/geometrixx-outdoors/brand",
  "name": " "brand_1_c02.jpg",
  "cq:name": "name",
  "srn:paging": {
   "total": 1,
   "limit": 20,
   "offset": 0
  },
  "metadata": {
   "dc:format": "image/jpeg",
   "xmp:CreatorTool": "Adobe Photoshop CS6 (Macintosh)",
   "dc:modified": "2014-03-18T13:40:37.482+02:00"
  },
  "related": {

  }
 },
```

ApiProperty

```
@ApiProperty(name = "cq:name")
public String getName() {
    return "name"
}
```

```
  "rel": [
     "parent"
   ],
   "href":
"http://radiohead.corp.adobe.com:4502/api/assets/content/dam/geom
etrixx-outdoors/brand.json"
  }
 ],
"properties": {
  "cq:parentPath": "/content/dam/geometrixx-outdoors/brand",
  "name": " "brand_1_c02.jpg",
  "cq:name": "name",
  [illegible]
   "total": 1,
   "limit": 20,
   "offset": 0
  },
  "metadata": {
   "dc:format": "image/jpeg",
   "xmp:CreatorTool": "Adobe Photoshop CS6 (Macintosh)",
   "dc:modified": "2014-03-18T13:40:37.482+02:00"
  },
  "related": {

  }
 },
```

ApiAction

```
@ApiAction(name = "add-folder", title = "Add Folder")

public void addFolder(@HttpRequestParam
SlingHttpServletRequest request,
@HttpFormParam(value = "name") String name) { }
```

```
"actions": [
  {
   "title": "Add Folder",
   "name": "add-folder",
   "method": "POST",
   "href": "/api/assets/content/dam/geometrixx-outdoors/brand",
   "fields": [
    {
     "name": ":operation",
     "value": "add-folder",
     "type": "hidden"
    },
    {
     "name": "name",
     "type": "text"
    }
   ]
  },
  {
   "title": "Add Asset",
   "name": "add-asset",
   "method": "POST",
   "href": "/api/assets/content/dam/geometrixx-outdoors/brand",
   "fields": [
    {
     "name": ":operation",
     "value": "add-asset",
     "type": "hidden"
    },
    {
     "name": "name",
     "type": "text"
    },
    {
     "name": "file",
     "type": "text"
    }
   ]
  }
```

ApiAction

```
@ApiAction(name = "add-folder", title = "Add Folder")

public void addFolder(@HttpRequestParam
SlingHttpServletRequest request,
@HttpFormParam(value = "name") String name) {
```

The default method is always POST so it can be omitted

```
"actions": [
 {
  "title": "Add Folder",
  "name": "add-folder",
  "method": "POST",
  "href": "/api/assets/content/dam/geometrixx-outdoors/brand",
  "fields": [
   {
    "name": ":operation",
    "value": "add-folder",
    "type": "hidden"
   },
   {
    "name": "name",
    "type": "text"
   }
  ]
 },
 {
  "title": "Add Asset",
  "name": "add-asset",
  "method": "POST",
  "href": "/api/assets/content/dam/geometrixx-outdoors/brand",
  "fields": [
   {
    "name": ":operation",
    "value": "add-asset",
    "type": "hidden"
   },
   {
    "name": "name",
    "type": "text"
   },
   {
    "name": "file",
    "type": "text"
   }
  ]
 }
```

ApiAction

```java
@ApiAction(name = "add-folder", title = "Add Folder")

public void addFolder(@HttpRequestParam
SlingHttpServletRequest request,
@HttpFormParam(value = "name") String name) { }
```

```json
"actions": [
  {
   "title": "Add Folder",
   "name": "add-folder",
   "method": "POST",
   "href": "/api/assets/content/dam/geometrixx-outdoors/brand",
   "fields": [
    {
     "name": ":operation",
     "value": "add-folder",
     "type": "hidden"
    },
    {
     "name": "name",
     "type": "text"
    }
   ]
  },
  {
   "title": "Add Asset",
   "name": "add-asset",
   "method": "POST",
   "href": "/api/assets/content/dam/geometrixx-outdoors/brand",
   "fields": [
    {
     "name": ":operation",
     "value": "add-asset",
     "type": "hidden"
    },
    {
     "name": "name",
     "type": "text"
    },
    {
     "name": "file",
     "type": "text"
    }
   ]
  }
```

ApiAction

```
@ApiAction(name = "add-folder", title = "Add Folder")

public void addFolder(@HttpRequestParam
SlingHttpServletRequest request,
@HttpFormParam(value = "name") String name) { }
```

```
"actions": [
 {
  "title": "Add Folder",
  "name": "add-folder",
  "method": "POST",
  "href": "/api/assets/content/dam/geometrixx-outdoors/brand",
  "fields": [
   {
    "name": ":operation",
    "value": "add-folder",
    "type": "hidden"
   },
   {
    "name": "name",
    "type": "text"
   }
  ]
 },
 {
  "title": "Add Asset",
  "name": "add-asset",
  "method": "POST",
  "href": "/api/assets/content/dam/geometrixx-outdoors/brand",
  "fields": [
   {
    "name": ":operation",
    "value": "add-asset",
    "type": "hidden"
   },
   {
    "name": "name",
    "type": "text"
   },
   {
    "name": "file",
    "type": "text"
   }
  ]
 }
```

Other Annotations

- ApiQuery
  - Allows you to intercept queries on the URL to return a set of resources
- ApiFilter (WIP)
  - Allows you to define filters that reduce the set of entities returned as children

Enable the Framework

- Only looks at bundles with a specific header, must add to the bundle:

```
<HAF-IO-Packages>c.a.g.package</HAF-IO-Packages>
```

- All other bundles will be ignored

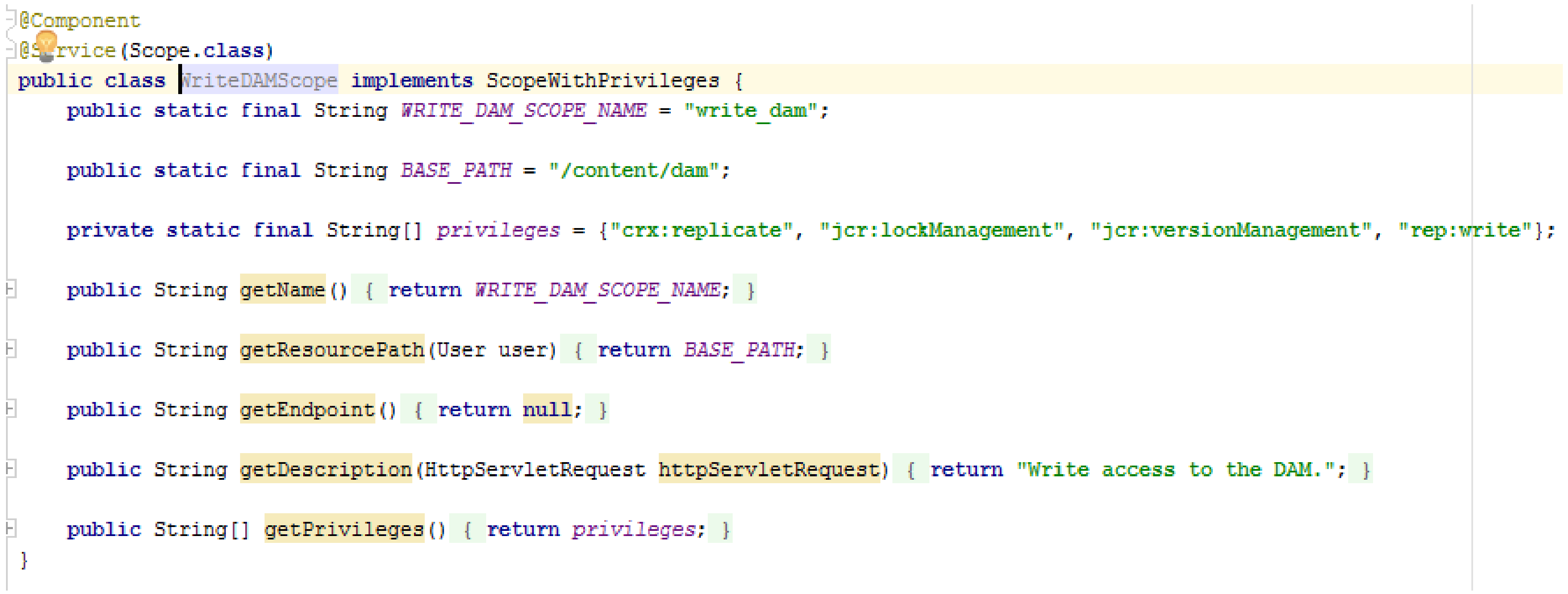
Oauth - Extensible Scopes

- Oauth is a common authorization protocol in use today
- AEM has an Oauth server (see Antonio CQ Gems presentation [2])
- Prior to 6.3
  - There was a small set of scopes available OOB (profile, offline, replicate)
  - Adding new ones required writing code directly in the oauth.server bundle
  - Only read access was supported
- With Extensible Scopes in 6.3
  - You create a service that implements an interface to define your custom scope
  - The service can live in any bundle and gets registered into the Oauth service when the bundle starts
  - Any privilege available in the repository can be used in a scope

Oauth - Extensible Scopes

- What is a scope in AEM?
  - A mapping of a path in the repository to a set of privileges
- To create a custom scope, you implement c.a.g.oauth.server.ScopeWithPrivileges
- ScopeWithPrivileges was created as an extension to the existing c.a.g.oauth.server.Scope interface to maintain backwards compatibility
- For a user to grant a scope, they must have the necessary privileges in the repository

Oauth – Extensible Scopes (example)



```
@Component
@Service(Scope.class)
public class WriteDAMScope implements ScopeWithPrivileges {
    public static final String WRITE_DAM_SCOPE_NAME = "write_dam";

    public static final String BASE_PATH = "/content/dam";

    private static final String[] privileges = {"crx:replicate", "jcr:lockManagement", "jcr:versionManagement", "rep:write"};

    public String getName() { return WRITE_DAM_SCOPE_NAME; }

    public String getResourcePath(User user) { return BASE_PATH; }

    public String getEndpoint() { return null; }

    public String getDescription(HttpServletRequest httpServletRequest) { return "Write access to the DAM."; }

    public String[] getPrivileges() { return privileges; }
}
```

Then VS Now

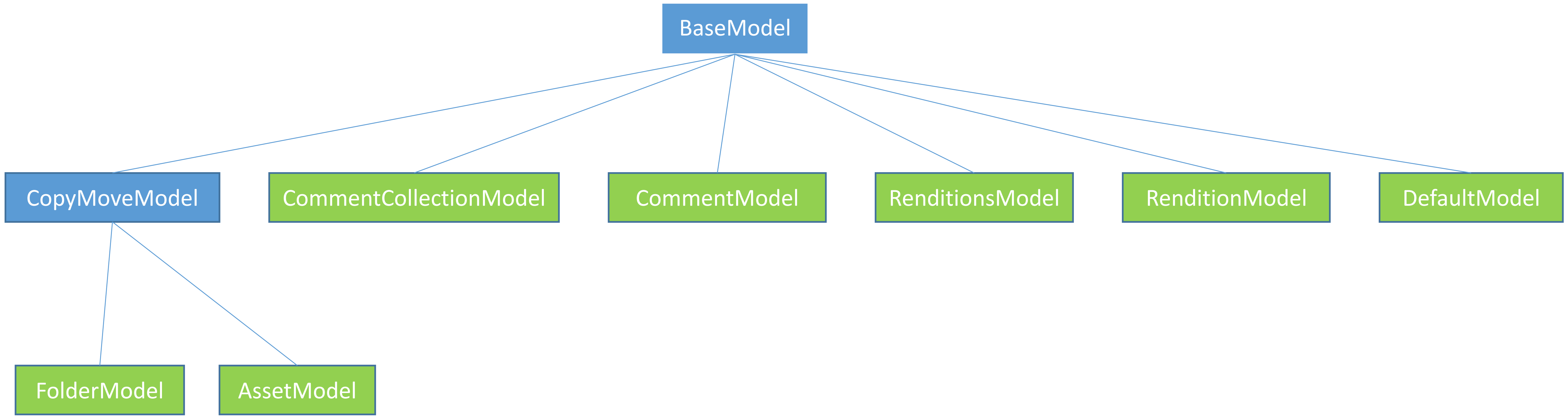
- In 6.2 the framework was very powerful but required a lot of boilerplate code in order to expose new resources and functionality
- Our goal was to use annotations to reduce the amount of code a developer needs to write
  - As well as to move common code like JSON serialization and Sling plumbing into the framework
- Let’s compare using the Assets HTTP API [3]

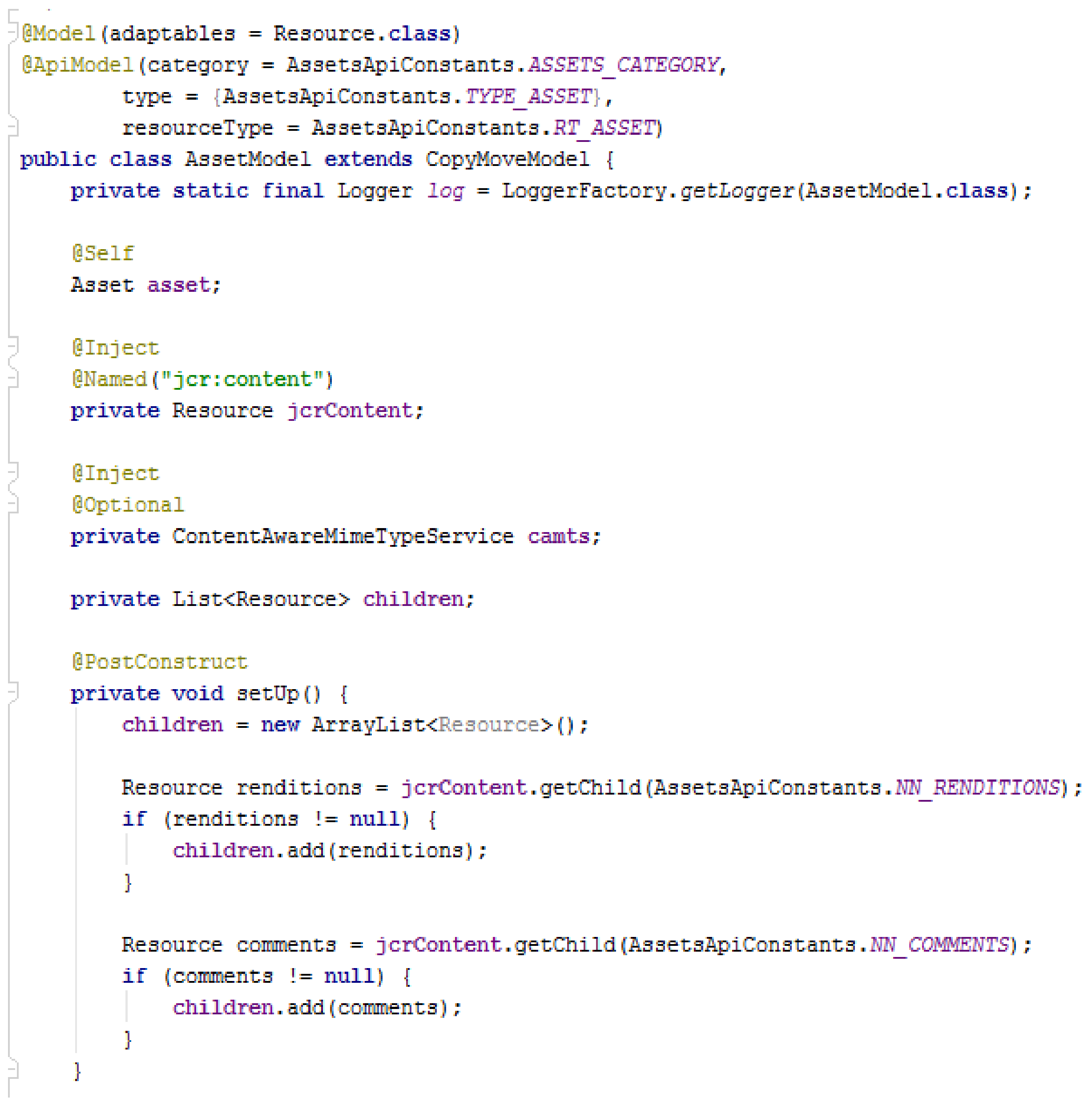
Then

- Exposing an Asset resource required 5 classes
  - AssetResource – The actual resource
  - AssetResourceConverter – To take the resource and serialize it into Siren JSON
  - AssetResourceProvider(Factory) – The provider of the custom resources and a factory to create it
  - AssetValueMapDecorator – To aggregate the useful properties from different nodes in the asset structure into the resource. To provide business functionality (ie: if you add a file property to the map, the decorator will call setRendition using the Assets Java API)

Now

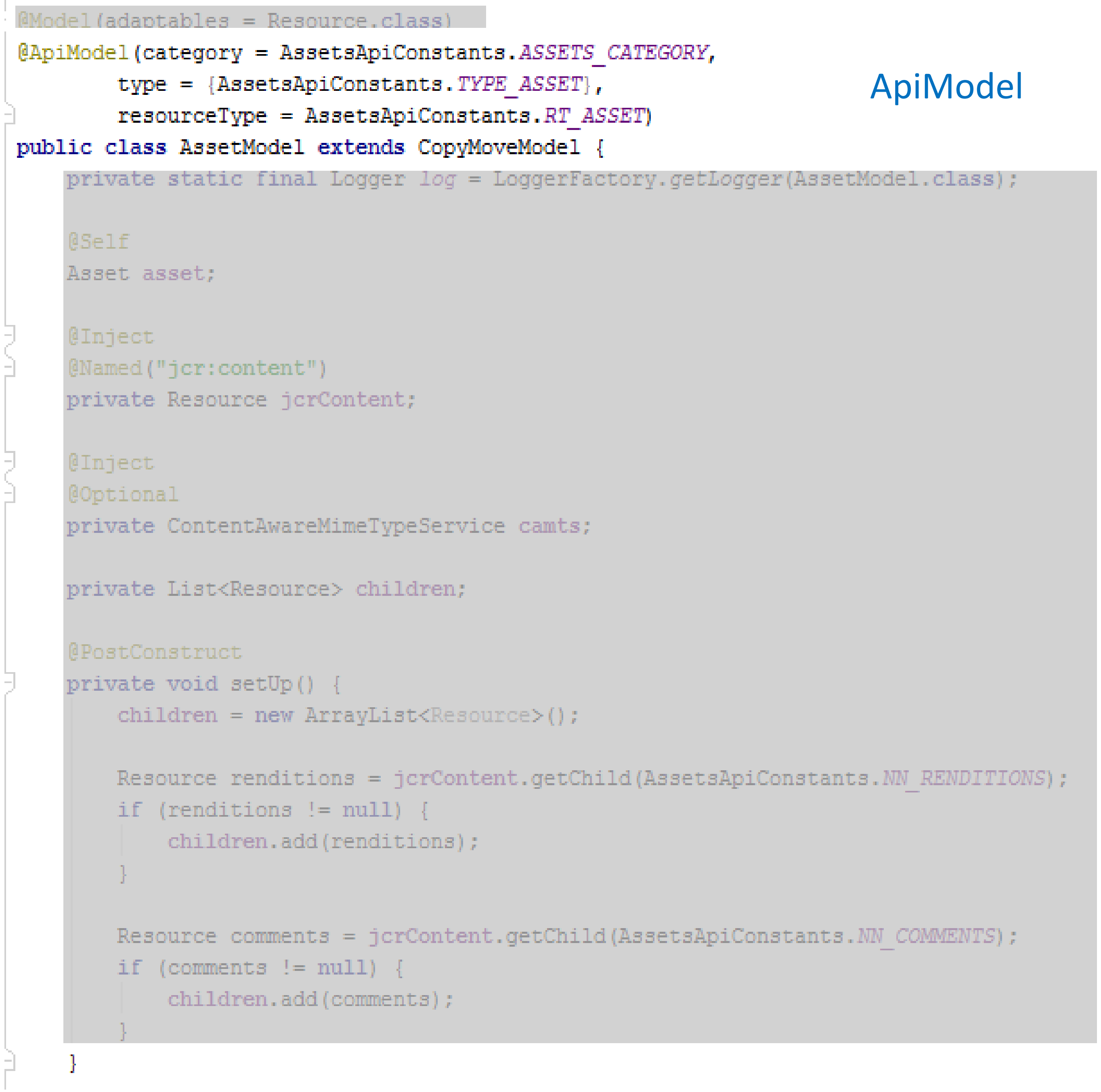
- Exposing an Asset resource requires 1 class
  - AssetModel
- The framework takes care of
  - JSON serialization (you don’t need to write code to create Siren JSON)
  - Sling internals
    - The framework defines the provider and uses a ResourceWrapper that knows how to interrogate the models based on metadata we build when processing the annotations
- Custom ValueMap decorators are no longer necessary


BaseModel
CopyMoveModel
CommentCollectionModel
CommentModel
RenditionsModel
RenditionModel
DefaultModel
FolderModel
AssetModel



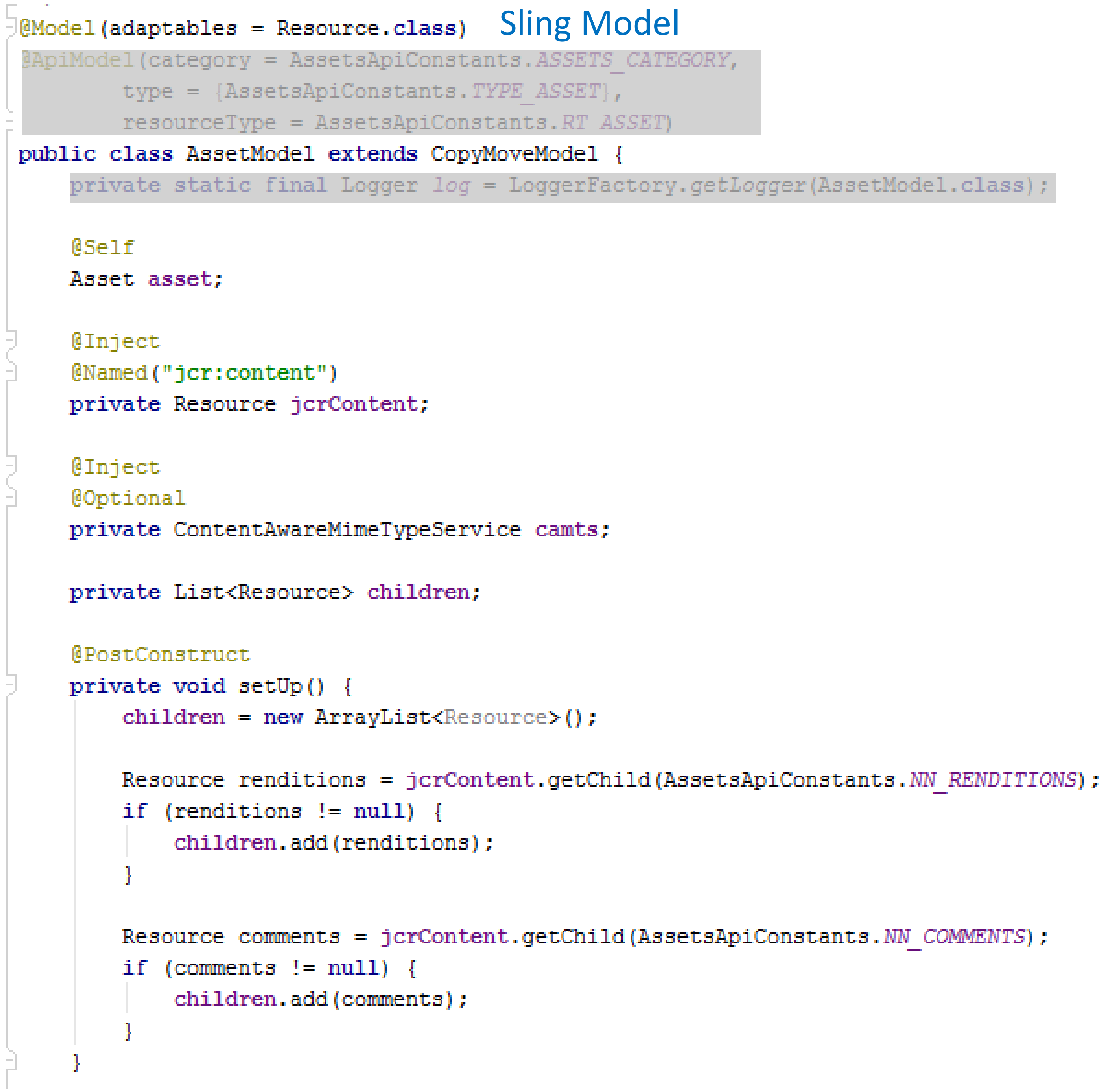
```java
@Model(adaptables = Resource.class)
@ApiModel(category = AssetsApiConstants.ASSETS_CATEGORY,
        type = {AssetsApiConstants.TYPE_ASSET},
        resourceType = AssetsApiConstants.RT_ASSET)
public class AssetModel extends CopyMoveModel {
    private static final Logger log = LoggerFactory.getLogger(AssetModel.class);

    @Self
    Asset asset;

    @Inject
    @Named("jcr:content")
    private Resource jcrContent;

    @Inject
    @Optional
    private ContentAwareMimeTypeService camts;

    private List<Resource> children;

    @PostConstruct
    private void setUp() {
        children = new ArrayList<Resource>();

        Resource renditions = jcrContent.getChild(AssetsApiConstants.NN_RENDITIONS);
        if (renditions != null) {
            children.add(renditions);
        }

        Resource comments = jcrContent.getChild(AssetsApiConstants.NN_COMMENTS);
        if (comments != null) {
            children.add(comments);
        }
    }
```

ApiModel

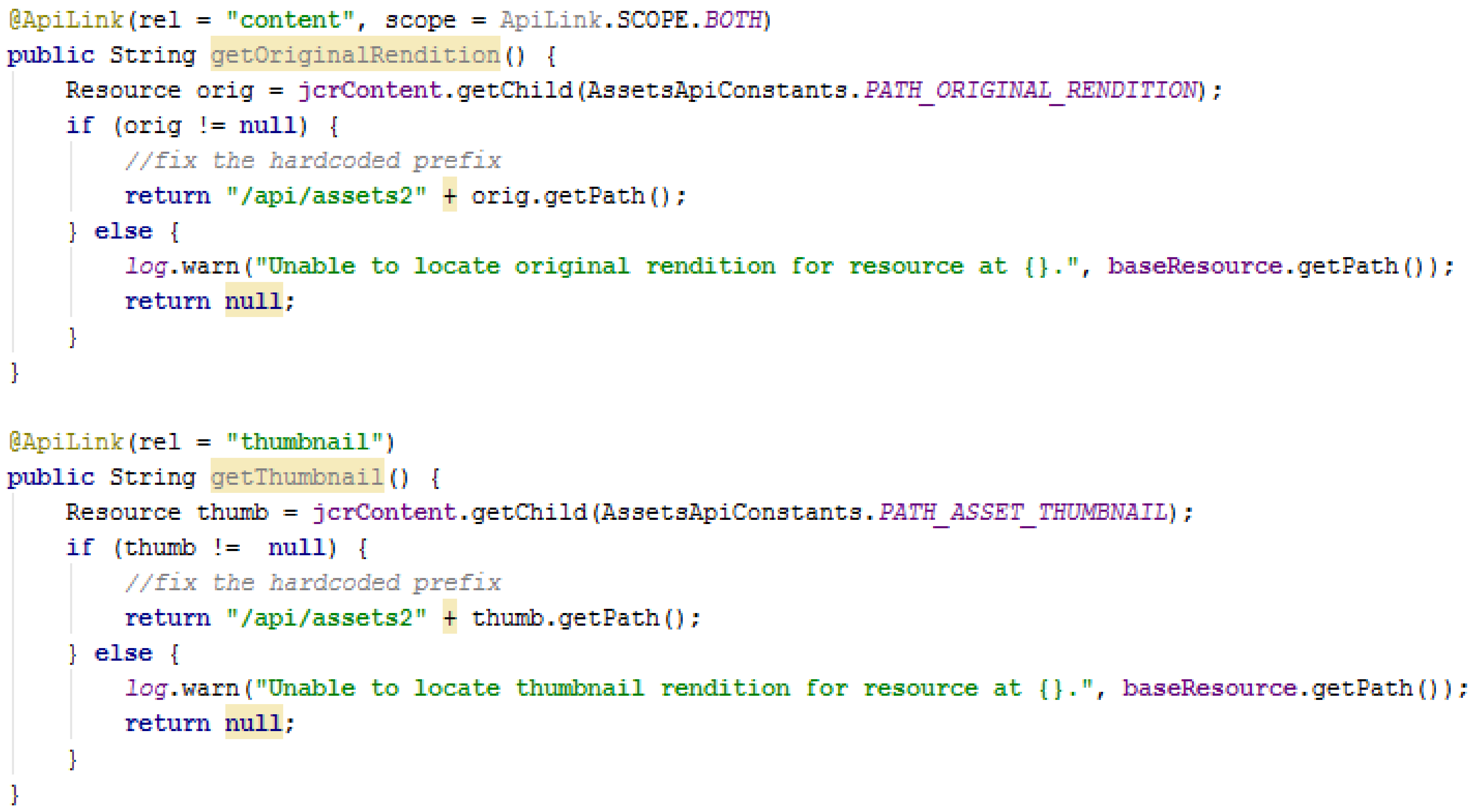


```java
@Model(adaptables = Resource.class)
@ApiModel(category = AssetsApiConstants.ASSETS_CATEGORY,
        type = {AssetsApiConstants.TYPE_ASSET},
        resourceType = AssetsApiConstants.RT_ASSET)
public class AssetModel extends CopyMoveModel {
    private static final Logger log = LoggerFactory.getLogger(AssetModel.class);

    @Self
    Asset asset;

    @Inject
    @Named("jcr:content")
    private Resource jcrContent;

    @Inject
    @Optional
    private ContentAwareMimeTypeService camts;

    private List<Resource> children;

    @PostConstruct
    private void setUp() {
        children = new ArrayList<Resource>();

        Resource renditions = jcrContent.getChild(AssetsApiConstants.NN_RENDITIONS);
        if (renditions != null) {
            children.add(renditions);
        }

        Resource comments = jcrContent.getChild(AssetsApiConstants.NN_COMMENTS);
        if (comments != null) {
            children.add(comments);
        }
    }
```

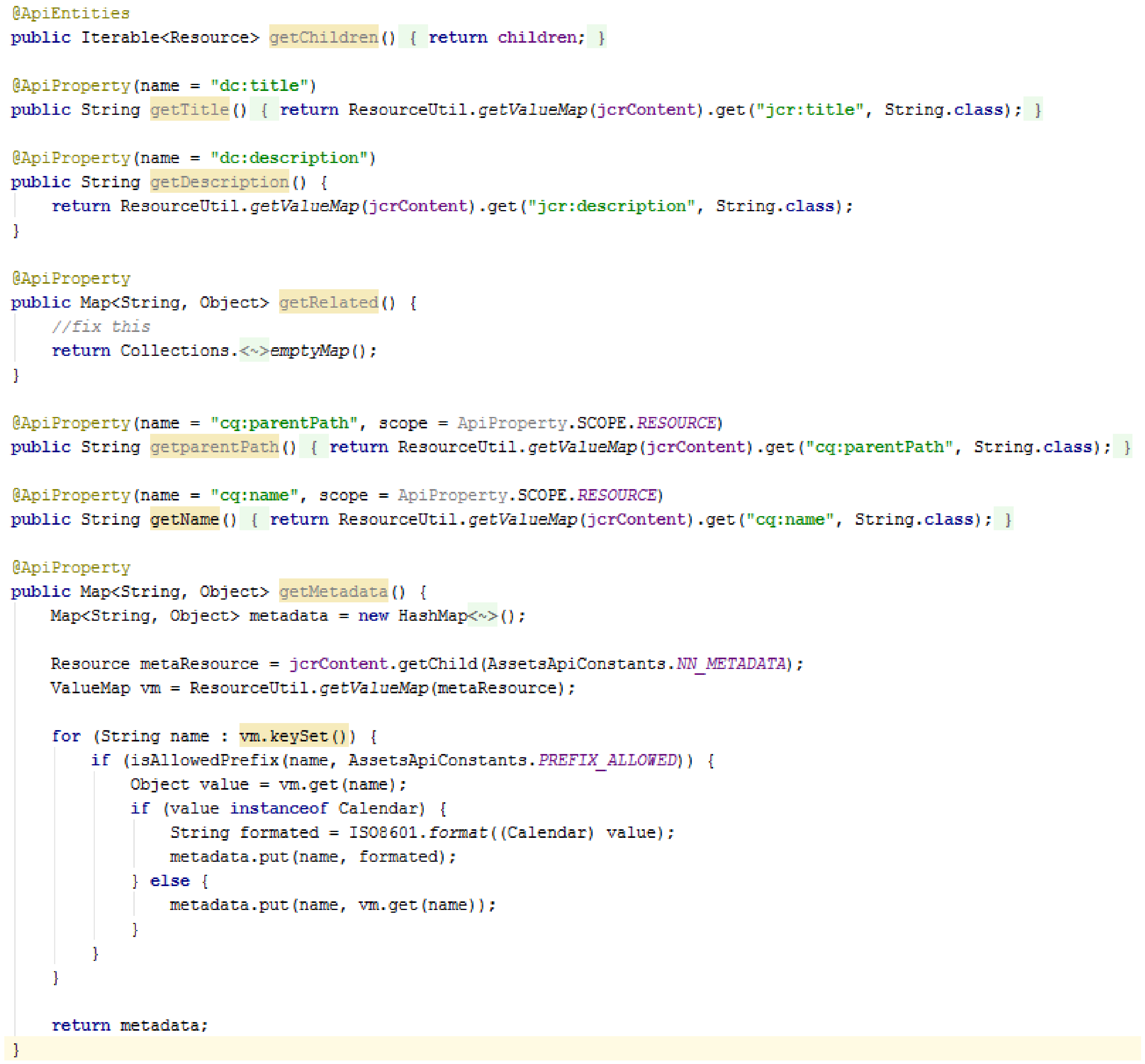
Sling Model



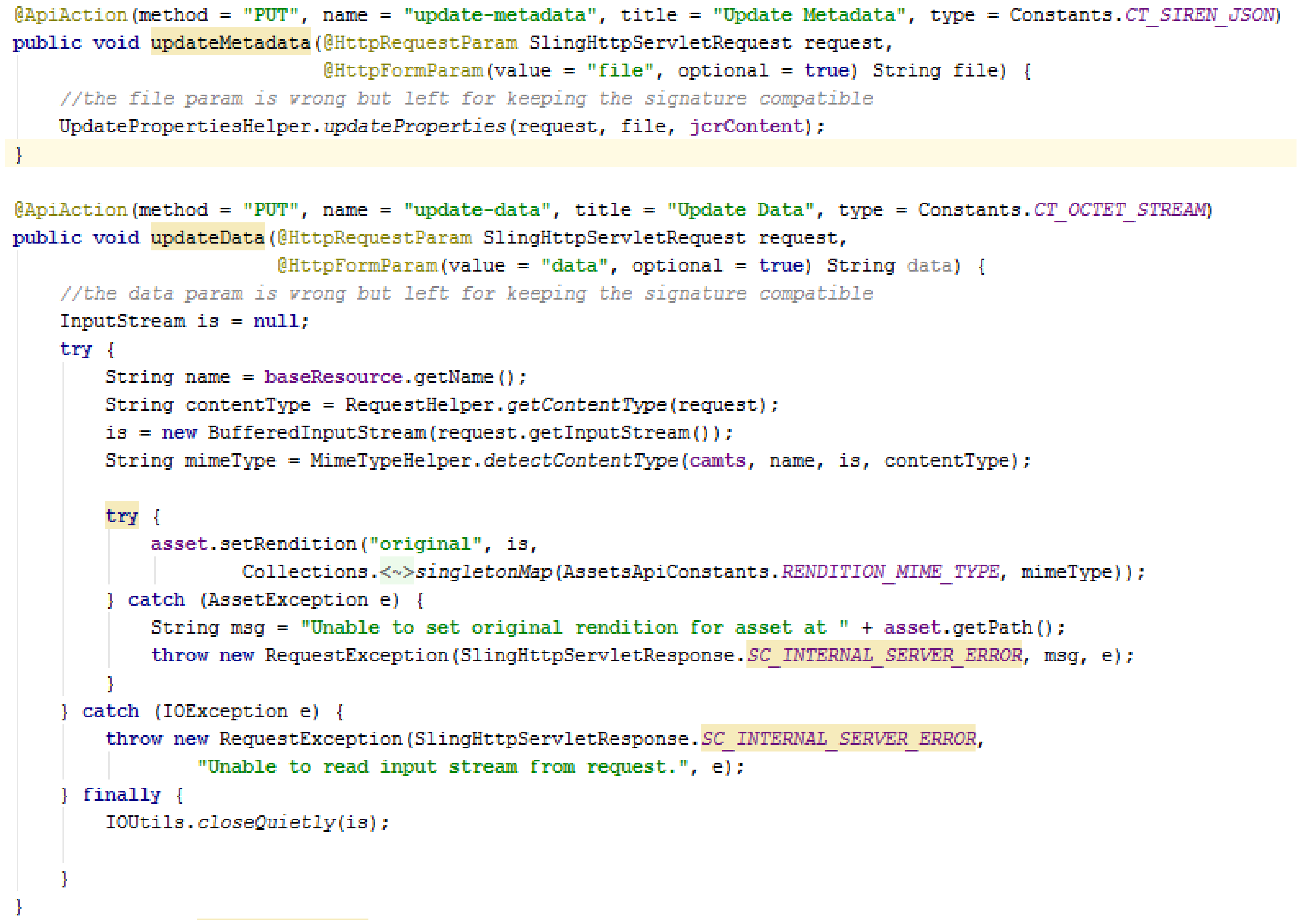
```java
@Model(adaptables = Resource.class)
@ApiModel(category = AssetsApiConstants.ASSETS_CATEGORY,
        type = {AssetsApiConstants.TYPE_ASSET},
        resourceType = AssetsApiConstants.RT_ASSET)
public class AssetModel extends CopyMoveModel {
    private static final Logger log = LoggerFactory.getLogger(AssetModel.class);

    @Self
    Asset asset;

    @Inject
    @Named("jcr:content")
    private Resource jcrContent;

    @Inject
    @Optional
    private ContentAwareMimeTypeService camts;

    private List<Resource> children;

    @PostConstruct
    private void setUp() {
        children = new ArrayList<Resource>();

        Resource renditions = jcrContent.getChild(AssetsApiConstants.NN_RENDITIONS);
        if (renditions != null) {
            children.add(renditions);
        }

        Resource comments = jcrContent.getChild(AssetsApiConstants.NN_COMMENTS);
        if (comments != null) {
            children.add(comments);
        }
    }
```



```java
@ApiLink(rel = "content", scope = ApiLink.SCOPE.BOTH)
public String getOriginalRendition() {
    Resource orig = jcrContent.getChild(AssetsApiConstants.PATH_ORIGINAL_RENDITION);
    if (orig != null) {
        //fix the hardcoded prefix
        return "/api/assets2" + orig.getPath();
    } else {
        log.warn("Unable to locate original rendition for resource at {}.", baseResource.getPath());
        return null;
    }
}

@ApiLink(rel = "thumbnail")
public String getThumbnail() {
    Resource thumb = jcrContent.getChild(AssetsApiConstants.PATH_ASSET_THUMBNAIL);
    if (thumb !=  null) {
        //fix the hardcoded prefix
        return "/api/assets2" + thumb.getPath();
    } else {
        log.warn("Unable to locate thumbnail rendition for resource at {}.", baseResource.getPath());
        return null;
    }
}
```



```java
@ApiEntities
public Iterable<Resource> getChildren() { return children; }

@ApiProperty(name = "dc:title")
public String getTitle() { return ResourceUtil.getValueMap(jcrContent).get("jcr:title", String.class); }

@ApiProperty(name = "dc:description")
public String getDescription() {
    return ResourceUtil.getValueMap(jcrContent).get("jcr:description", String.class);
}

@ApiProperty
public Map<String, Object> getRelated() {
    //fix this
    return Collections.<~>emptyMap();
}

@ApiProperty(name = "cq:parentPath", scope = ApiProperty.SCOPE.RESOURCE)
public String getparentPath() { return ResourceUtil.getValueMap(jcrContent).get("cq:parentPath", String.class); }

@ApiProperty(name = "cq:name", scope = ApiProperty.SCOPE.RESOURCE)
public String getName() { return ResourceUtil.getValueMap(jcrContent).get("cq:name", String.class); }

@ApiProperty
public Map<String, Object> getMetadata() {
    Map<String, Object> metadata = new HashMap<~>();

    Resource metaResource = jcrContent.getChild(AssetsApiConstants.NN_METADATA);
    ValueMap vm = ResourceUtil.getValueMap(metaResource);

    for (String name : vm.keySet()) {
        if (isAllowedPrefix(name, AssetsApiConstants.PREFIX_ALLOWED)) {
            Object value = vm.get(name);
            if (value instanceof Calendar) {
                String formated = ISO8601.format((Calendar) value);
                metadata.put(name, formated);
            } else {
                metadata.put(name, vm.get(name));
            }
        }
    }

    return metadata;
}
```



```java
@ApiAction(method = "PUT", name = "update-metadata", title = "Update Metadata", type = Constants.CT_SIREN_JSON)
public void updateMetadata(@HttpRequestParam SlingHttpServletRequest request,
                           @HttpFormParam(value = "file", optional = true) String file) {
    //the file param is wrong but left for keeping the signature compatible
    UpdatePropertiesHelper.updateProperties(request, file, jcrContent);
}

@ApiAction(method = "PUT", name = "update-data", title = "Update Data", type = Constants.CT_OCTET_STREAM)
public void updateData(@HttpRequestParam SlingHttpServletRequest request,
                       @HttpFormParam(value = "data", optional = true) String data) {
    //the data param is wrong but left for keeping the signature compatible
    InputStream is = null;
    try {
        String name = baseResource.getName();
        String contentType = RequestHelper.getContentType(request);
        is = new BufferedInputStream(request.getInputStream());
        String mimeType = MimeTypeHelper.detectContentType(camts, name, is, contentType);

        try {
            asset.setRendition("original", is,
                    Collections.<~>singletonMap(AssetsApiConstants.RENDITION_MIME_TYPE, mimeType));
        } catch (AssetException e) {
            String msg = "Unable to set original rendition for asset at " + asset.getPath();
            throw new RequestException(SlingHttpServletResponse.SC_INTERNAL_SERVER_ERROR, msg, e);
        }
    } catch (IOException e) {
        throw new RequestException(SlingHttpServletResponse.SC_INTERNAL_SERVER_ERROR,
                "Unable to read input stream from request.", e);
    } finally {
        IOUtils.closeQuietly(is);

    }
}
```

The Future

- Automatic doc generation
- Request throttling
- Support for Synthetic Resources
- Filtering
- URL vs Content Path Mapping (in progress)
- Alternative Serialization Formats (oData, JSON-LD)
- Need something else? Let's talk![4]

Links

[0] - https://github.com/kevinswiber/siren
[1] - http://tools.ietf.org/html/rfc5988
[2] - https://docs.adobe.com/ddc/en/gems/oauth-server-functionality-in-aem---embrace-federation-and-unlea.html
[3] - https://git.corp.adobe.com/Granite/com.adobe.granite.rest.assets
[4] - https://wiki.corp.adobe.com/display/granite/ISV+Engineering