**HTL – JAVA USE API**

The HTML Template Language (HTL) Java Use-API enables a HTL file to access helper methods in a custom Java class. This allows all complex business logic to be encapsulated in the Java code, while the HTL code deals only with direct markup production.

**A Simple Example**

We'll start with an HTL component that *does not* have a use-class. It consists of a single file,  
**/apps/my-example/components/info.html**:

**/apps/my-example/component/info/info.html**

|  |  |
| --- | --- |
|  | **<div>**  **<h1>${properties.title}</h1>**  **<p>${properties.description}</p>**  **</div>** |

We also add some content for this component to render at **/content/my-example/**:

**http://localhost:4502/content/my-example.json**

|  |  |
| --- | --- |
|  | **{**  **"sling:resourceType": "my-example/component/info",**  **"title": "My Example",**  **"description": "This Is Some Example Content."**  **}** |

When this content is accessed, the HTL file is executed. Within the HTL code we use the context object **properties**to access the current resource's **title** and **description** and display them. The output HTML will be:

**view-source:http://localhost:4502/content/my-example.html**

|  |  |
| --- | --- |
|  | **<div>**  **<h1>My Example</h1>**  **<p>This Is Some Example Content.</p>**  **</div>** |

**Adding a Use-Class**

The **info** component as it stands does not need a use-class to perform its (very simple) function. There are cases, however, where you need to do things that cannot be done in HTL and so you need a use-class. But keep in mind the following:

**Note:**

*A use-class should only be used when something cannot be done in HTL alone.*

For example, suppose that you want the **info** component to display the **title** and **description** properties of the resource, but all in lowercase. Since HTL does not have a method for lowercasing strings, you will need a use-class. We can do this by adding a Java use-class and changing the **info.html** as follows:

**/apps/my-example/component/info/info.html**

|  |  |
| --- | --- |
|  | **<div data-sly-use.info="Info">**  **<h1>${info.lowerCaseTitle}</h1>**  **<p>${info.lowerCaseDescription}</p>**  **</div>** |

**/apps/my-example/component/info/Info.java**

|  |  |
| --- | --- |
|  | **package apps.my\_example.components.info;**    **import com.adobe.cq.sightly.WCMUse;**    **public class Info extends WCMUse {**  **private String lowerCaseTitle;**  **private String lowerCaseDescription;**    **@Override**  **public void activate() throws Exception {**  **lowerCaseTitle = getProperties().get("title", "").toLowerCase();**  **lowerCaseDescription = getProperties().get("description", "").toLowerCase();**  **}**    **public String getLowerCaseTitle() {**  **return lowerCaseTitle;**  **}**    **public String getLowerCaseDescription() {**  **return lowerCaseDescription;**  **}**  **}** |

In the following sections we walk through the different parts of the code.

**Local vs Bundle Java Class**

The Java use-class can be installed in two ways: **local**or **bundle**. **This example uses a local install.**

In a local install, the Java source file is placed alongside the HTL file, in the same repository folder. The source is automatically compiled on demand. No separate compilation or packaging step is required.

In a bundle install, the Java class must be compiled and deployed within an OSGi bundle using the standard AEM bundle deployment mechanism (see [Bundled Java Class](https://helpx.adobe.com/experience-manager/htl/using/use-api-java.html#BundledJavaClass)).

**Note:**

A **local Java use-class** is recommended when the use-class is specific to the component in question.

A **bundle Java use-class** is recommended when the Java code implements a service that is accessed from multiple HTL components.

**Java package is repository path**

When a local install is used, the package name of the use-class must match that of the repository folder location, with any hyphens in the path replaced by underscores in the package name.

In this case **Info.java** is located at **/apps/my-example/components/info** so the package is **apps.my\_example.components.info** :

**/apps/my-example/component/info/Info.java**

|  |  |
| --- | --- |
|  | **package apps.my\_example.components.info;**    **import com.adobe.cq.sightly.WCMUse;**    **public class Info extends WCMUse {**    **...**    **}** |

**Note:**

Using hyphens in the names of repository items is a recommended practice in AEM developement. However, hyphens are illegal within Java package names. For this reason, **all hyphens in the repository path must be converted to underscores in the package name**.

**Extending WCMUse**

While there are number of ways of incorporating a Java class with HTL (see Alternatives to WCMUse), the simplest is to extend the WCMUse class:

**/apps/my-example/component/info/Info.java**

|  |  |
| --- | --- |
|  | **package apps.my\_example.components.info;**    **import com.adobe.cq.sightly.WCMUse;**    **public class Info extends WCMUse {**    **...**  **}** |

**Initializing the class**

When the use-class is extended from **WCMUse**, initializiation is performed by overriding the **activate** method:

**/apps/my-example/component/info/Info.java**

|  |  |
| --- | --- |
|  | **...**    **public class Info extends WCMUse {**  **private String lowerCaseTitle;**  **private String lowerCaseDescription;**    **@Override**  **public void activate() throws Exception {**  **lowerCaseTitle = getProperties().get("title", "").toLowerCase();**  **lowerCaseDescription = getProperties().get("description", "").toLowerCase();**  **}**    **...**    **}** |

**Context**

Typically, the [**activate**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html#activate()) method is used to precompute and store (in member variables) the values needed in your HTL code, based on the current context (the current request and resource, for example).

The **[WCMUse](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html)** class provides access to the same set of context objects as are available within an HTL file (see [Global Objects](https://helpx.adobe.com/experience-manager/htl/using/global-objects.html)).

In a class that extends **WCMUse**, context objects can be accessed *by name* using

[**<T> T get(String name, Class<T> type)**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html#get(java.lang.String, java.lang.Class))

Alternatively, commonly used context objects can be accessed directly by the appropriate **convenience method**:

**[PageManager](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/day/cq/wcm/api/PageManager.html" \t "_blank) [getPageManager()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getPageManager()" \t "_blank)**

[**Page**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/day/cq/wcm/api/Page.html)**[getCurrentPage()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getCurrentPage()" \t "_blank)**

[**Page**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/day/cq/wcm/api/Page.html)**[getResourcePage()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getResourcePage()" \t "_blank)**

[**ValueMap**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/resource/ValueMap.html)**[getPageProperties()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getPageProperties()" \t "_blank)**

[**ValueMap**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/resource/ValueMap.html)**[getProperties()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getProperties()" \t "_blank)**

[**Designer**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/day/cq/wcm/api/designer/Designer.html)**[getDesigner()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getDesigner()" \t "_blank)**

[**Design**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/day/cq/wcm/api/designer/Design.html)**[getCurrentDesign()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getCurrentDesign()" \t "_blank)**

[**Style**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/day/cq/wcm/api/designer/Style.html)**[getCurrentStyle()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getCurrentStyle()" \t "_blank)**

[**Component**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/day/cq/wcm/api/components/Component.html)**[getComponent()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getComponent()" \t "_blank)**

[**ValueMap**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/resource/ValueMap.html)**[getInheritedProperties()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getInheritedProperties()" \t "_blank)**

[**Resource**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/resource/Resource.html)**[getResource()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getResource()" \t "_blank)**

[**ResourceResolver**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/resource/ResourceResolver.html)**[getResourceResolver()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getResourceResolver()" \t "_blank)**

[**SlingHttpServletRequest**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/SlingHttpServletRequest.html)**[getRequest()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getRequest()" \t "_blank)**

[**SlingHttpServletResponse**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/SlingHttpServletResponse.html)**[getResponse()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getResponse()" \t "_blank)**

[**SlingScriptHelper**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/scripting/SlingScriptHelper.html)**[getSlingScriptHelper()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getSlingScriptHelper()" \t "_blank)**

[**SightlyWCMMode**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/SightlyWCMMode.html)**[getWcmMode()](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html" \l "getWcmMode()" \t "_blank)**

**Getter methods**

Once the use-class has initialized, the HTL file is run. During this stage HTL will typically pull in the state of various member variables of the use-class and render them for presentation.

To provide access to these values from within the HTL file you must define custom getter methods in the use-class **according to the following naming convention**:

* A method of the form **get*Xyz*** will expose within the HTL file an object property called ***xyz***.

For example, in the following example, the methods **getTitle** and **getDescription** result in the object properties **title** and **description**becoming accessible within the context of the HTL file:

**/apps/my-example/component/info/Info.java**

|  |  |
| --- | --- |
|  | **...**    **public class Info extends WCMUse {**    **...**    **public String getLowerCaseTitle() {**  **return lowerCaseTitle;**  **}**    **public String getLowerCaseDescription() {**  **return lowerCaseDescription;**  **}**  **}** |

**data-sly-use attribute**

The  **data-sly-use** attribute is used to initialize the use-class within your HTL code. In our example, the **data-sly-use** attribute declares that we want to use the class **Info**. We can use just the local name of the class because we are using a local install (having placed the Java source file is in the same folder as the HTL file). If we were using a bundle install we would have to specify the fully qualified classname (See [Use-class Bundle Install](https://helpx.adobe.com/experience-manager/htl/using/use-api-java.html#LocalvsBundleJavaClass)).

**/apps/my-example/component/info/info.html**

|  |  |
| --- | --- |
|  | **<div data-sly-use.info="Info">**  **<h1>${info.lowerCaseTitle}</h1>**  **<p>${info.lowerCaseDescription}</p>**  **</div>** |

**Local identifier**

The identifier '**info**' (after the dot in **data-sly-use.info**) is used within the HTL file to identify the class. The scope of this identifier is global within the file, after it has been declared. It is not limited to the element that contains the **data-sly-use**statement.

**/apps/my-example/component/info/info.html**

|  |  |
| --- | --- |
|  | **<div data-sly-use.info="Info">**  **<h1>${info.lowerCaseTitle}</h1>**  **<p>${info.lowerCaseDescription}</p>**  **</div>** |

**Getting properties**

The identifier **info** is then used to access the object properties **title** and **description** that were exposed through the getter methods **Info.getTitle** and **Info.getDescription**.

**/apps/my-example/component/info/info.html**

|  |  |
| --- | --- |
|  | **<div data-sly-use.info="Info">**  **<h1>${info.lowerCaseTitle}</h1>**  **<p>${info.lowerCaseDescription}</p>**  **</div>** |

**Output**

Now, when we access**/content/my-example.html** it will return the following HTML:

**view-source:http://localhost:4502/content/my-example.html**

|  |  |
| --- | --- |
|  | **<div>**  **<h1>my example</h1>**  **<p>this is some example content.</p>**  **</div>** |

**Beyond The Basics**

In this section we'll introdce some further features that go beyond the simple example above:

* Passing parameters to a use-class.
* Bundled Java use-class.
* Alternatives to **WCMUse**

**Passing Parameters**

Parameters can be passed to a use-class upon initialization. For example, we could do something like this:

**/content/my-example/component/info/info.html**

|  |  |
| --- | --- |
|  | **<div data-sly-use.info="${'Info' @ text='Some text'}">**  **<h1>${info.lowerCaseTitle}</h1>**  **<p>${info.lowerCaseDescription}</p>**  **<p>${info.upperCaseText}</p>**  **</div>** |

Here we are passing a parameter called **text**. The use-class then uppercases the string we retrieve and display the result with **info.upperCaseText**. Here is the adjusted use-class:

**/apps/my-example/component/info/Info.java**

|  |  |
| --- | --- |
|  | **package apps.my\_example.components.info;**    **import com.adobe.cq.sightly.WCMUse;**    **public class Info extends WCMUse {**    **...**    **private String reverseText;**    **@Override**  **public void activate() throws Exception {**    **...**    **String text = get("text", String.class);**  **reverseText = new StringBuffer(text).reverse().toString();**    **}**    **public String getReverseText() {**  **return reverseText;**  **}**    **...**  **}** |

The parameter is accessed through the **WCMUse** method

[**<T> T get(String paramName, Class<T> type)**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUse.html#get(java.lang.String, java.lang.Class))

In our case, the statement

**get("text", String.class)**

The string is then reversed and exposed via the method

**getReverseText()**

**Only Pass Parameters from data-sly-template**

While the above example is technically correct, it actually does not make much sense to pass a value from HTL to initialize a use-class, when the value in question is available in the execution context of the HTL code (or, trivially, the value is static, as above).

The reason is that the use-class will always have access to the same execution context as the HTL code. This brings up an import point of best practice:

**Note:**

Passing a parameter to a use-class should only be done when the use-class is used in a **data-sly-template** file which itself is called from another HTL file with parameters that need to be passed on.

For example, let's create a separate **data-sly-template** file alongside our existing example. We will call the new file **extra.html**. It contains a **data-sly-template** block called **extra**:

**/apps/my-example/component/info/extra.html**

|  |  |
| --- | --- |
|  | **<template data-sly-template.extra="${@ text}"**  **data-sly-use.extraHelper="${'ExtraHelper' @ text=text}">**  **<p>${extraHelper.reversedText}</p>**  **</template>** |

The template **extra**, takes a single parameter, **text**. It then initializes the Java use-class **ExtraHelper** with the local name **extraHelper** and passes it the value of the template parameter **text** as the use-class parameter **text**.

The body of the template gets the property **extraHelper.reversedText** (which, under the hood, actually calls **ExtraHelper.getReversedText()**) and displays that value.

We also adapt our existing **info.html** to use this new template:

**/apps/my-example/component/info/info.html**

|  |  |
| --- | --- |
|  | **<div data-sly-use.info="Info"**  **data-sly-use.extra="extra.html">**    **<h1>${info.lowerCaseTitle}</h1>**  **<p>${info.lowerCaseDescription}</p>**    **<div data-sly-call="${extra.extra @ text=properties.description}"></div>**    **</div>** |

The file **info.html** now contains two **data-sly-use** statements, the original one that imports the **Info** Java use-class and a new one that imports the template file under the local name **extra**.

Note that we could have placed the template block inside the **info.html** file to avoid the second **data-sly-use**, but a separate template file is more common, and more reusable.

The **Info** class is employed as before, calling its getter methods **getLowerCaseTitle()** and **getLowerCaseDescription()** through their corresponding HTL properties**info.lowerCaseTitle** and **info.lowerCaseDescription**.

Then we perform a **data-sly-call** to the template **extra** and pass it the value **properties.description** as the parameter **text**.

The Java use-class **Info.java** is changed to handle the new text parameter:

**/apps/my-example/component/info/ExtraHelper.java**

|  |  |
| --- | --- |
|  | **package apps.my\_example.components.info;**    **import com.adobe.cq.sightly.WCMUse;**    **public class ExtraHelper extends WCMUse {**  **private String reversedText;**  **...**    **@Override**  **public void activate() throws Exception {**  **String text = get("text", String.class);**  **reversedText = new StringBuilder(text).reverse().toString();**    **...**  **}**    **public String getReversedText() {**  **return reversedText;**  **}**  **}** |

The **text** parameter is retrieved with **get("text", String.class)**, the value is reversed and made available as the HTL object **reversedText** through the getter **getReversedText()**.

**Bundled Java Class**

With a bundle use-class the class must be compiled, packaged and deployed in AEM using the standard OSGi bundle deployment mechanism. In contrast with a local install, the use-class **package declaration** should be named normally:

**/apps/my-example/component/info/Info.java**

|  |  |
| --- | --- |
|  | **package org.example.app.components;**    **import com.adobe.cq.sightly.WCMUse;**    **public class Info extends WCMUse {**  **...**  **}** |

and, the **data-sly-use** statement must reference the *fully qualified class name*, as opposed to just the local class name:

**/apps/my-example/component/info/info.html**

|  |  |
| --- | --- |
|  | **<div data-sly-use.info="org.example.app.components.info.Info">**  **<h1>${info.title}</h1>**  **<p>${info.description}</p>**  **</div>** |

**Alternatives to WCMUse**

The most common way to create a Java use-class is to extend **WCMUse**. However, there are a number of other options. To understand these variants it helps to understand how the HTL **data-sly-use** statement works under the hood.

Suppose you have the following **data-sly-use** statement:

**<div data-sly-use.*localName*="*UseClass*">**

The system processes the statement as follows:

(1)

* If there exists a local file ***UseClass*.java**in the same directory as the HTL file, try to compile and load that class. If successful go to (2).
* Otherwise, interpret ***UseClass*** as a fully qualified class name and try to load it from the OSGi environment. If successful go to (2).
* Otherwise, interpret ***UseClass*** as a path to a HTL or JavaScript file and load that file. If successful goto (4).

(2)

* Try to adapt the current **Resource** to ***UseClass.*** If successful, go to (3).
* Otherwise, try to adapt the current **Request** to ***UseClass***.If successful, go to (3).
* Otherwise, try to instantiate ***UseClass*** with a zero-argument constructor. If successful, go to (3).

(3)

* Within HTL, bind the newly adapted or created object to the name ***localName***.
* If ***UseClass*** implements **[io.sightly.java.api.Use](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/io/sightly/java/api/Use.html)**then call the **init** method, passing the current execution context (in the form of a **javax.scripting.Bindings** object).

(4)

* If ***UseClass*** is a path to a HTL file containing a **data-sly-template**, prepare the template.
* Otherwise, if ***UseClass*** is a path to a JavaScript use-class, prepare the use-class (see [JavaScript Use-API](https://helpx.adobe.com/experience-manager/htl/using/use-api-javascript.html)).

A few significant points about the above description:

* Any class that is adaptable from **Resource**, adaptable from **Request**, or that has a zero-argument constructor can be a use-class. The class does not have to extend extend **WCMUse** or even implement **Use**.
* However, if the use-class *does* implement **Use**, then its **init** method will automatically be called with the current context, allowing you to place initialization code there that depends on that context.
* A use-class that extends **WCMUse** is just a special case of implementing **Use**. It provides the convenience context methods and its **activate** method is automatically called from **Use.init**.

**Directly Implement Interface Use**

While the most common way to create a use-class is to extend **WCMUse**, it is also possible to directly implement the **[io.sightly.java.api.Use](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/io/sightly/java/api/Use.html)**interface itself.

The **Use** interface defines only one method:

[**public void init(javax.script.Bindings bindings)**](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/io/sightly/java/api/Use.html#init(javax.script.Bindings))

The **init** method will be called on initialization of the class with a **Bindings** object that holds all the context objects and any parameters passed into the use-class.

All additional functionality (such as the equivalent of **WCMUse.getProperties()**) must be implmented explicitly using the **[javax.script.Bindings](http://docs.oracle.com/javase/7/docs/api/javax/script/Bindings.html)** object. For example:

**Info.java**

|  |  |
| --- | --- |
|  | **import io.sightly.java.api.Use;**    **public class MyComponent implements Use {**  **...**  **@Override**  **public void init(Bindings bindings) {**    **// All standard objects/binding are available**  **Resource resource = (Resource)bindings.get("resource");**  **ValueMap properties = (ValueMap)bindings.get("properties");**  **...**    **// Parameters passed to the use-class are also available**  **String param1 = (String) bindings.get("param1");**  **}**  **...**  **}** |

The main case for implementing the **Use** interface yourself instead of extending **WCMUse** is when you wish to use a subclass of an already existing class as the use-class.

**Adaptable from Resource**

Another option is to use a helper class that is adaptable from **[org.apache.sling.api.resource.Resource](https://helpx.adobe.com/experience-manager/htl/using/content/help/en/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/resource/Resource.html)**.

Let's say you need to write an HTL script that displays the mimetype of a DAM asset. In this case you know that when your HTL script is called, it will be within the context of a **Resource** that wraps a JCR **Node** with nodetype **dam:Asset**.

You know that a **dam:Asset** node has a structure like this:

**Repository Structure**

|  |  |
| --- | --- |
|  | **{**  **"content": {**  **"dam": {**  **"geometrixx": {**  **"portraits": {**  **"jane\_doe.jpg": {**  **...**  **"jcr:content": {**  **...**  **"metadata": {**  **...**  **},**  **"renditions": {**  **...**  **"original": {**  **...**  **"jcr:content": {**  **"jcr:primaryType": "nt:resource",**  **"jcr:lastModifiedBy": "admin",**  **"jcr:mimeType": "image/jpeg",**  **"jcr:lastModified": "Fri Jun 13 2014 15:27:39 GMT+0200",**  **"jcr:data": ...,**  **"jcr:uuid": "22e3c598-4fa8-4c5d-8b47-8aecfb5de399"**  **}**  **},**  **"cq5dam.thumbnail.319.319.png": {**  **...**  **},**  **"cq5dam.thumbnail.48.48.png": {**  **...**  **},**  **"cq5dam.thumbnail.140.100.png": {**  **...**  **}**  **}**  **}**  **}**  **}**  **}**  **}**  **}** |

Here we show the asset (a JPEG image) that comes with a default install of AEM as part of the example project geometrixx. The asset is called **jane\_doe.jpg** and its mimetype is **image/jpeg**.

To access the asset from within HTL, you can declare **[com.day.cq.dam.api.Asset](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/com/adobe/granite/asset/api/Asset.html)** as the class in the **data-sly-use** statement: and then use a get method of **Asset** to retrieve the desired information. For example:

**mimetype.html**

|  |  |
| --- | --- |
|  | **<div data-sly-use.asset="com.day.cq.dam.api.Asset">**  **<p>${asset.mimeType}</p>**  **</div>** |

The **data-sly-use** statement directs HTL to adapt the current **Resource** to an **Asset** and give it the local name **asset**. It then calls the **getMimeType** method of **Asset** using the HTL getter shorthand: **asset.mimeType**.

**Adaptable from Request**

It is also possible to emply as a use-class any class that is adaptable from **[org.apache.sling.api.SlingHttpServletRequest](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/SlingHttpServletRequest.html)**

As with the above case of a use-class adaptable from **Resource**, a use-class adaptable from **[SlingHttpServletRequest](https://helpx.adobe.com/experience-manager/6-2/sites/developing/using/reference-materials/javadoc/org/apache/sling/api/SlingHttpServletRequest.html)**can be specified in the **data-sly-use** statement. Upon execution the current request will be adapted to the class given and the resulting object will be made available within HTL.