Pdf.js Wicket Component

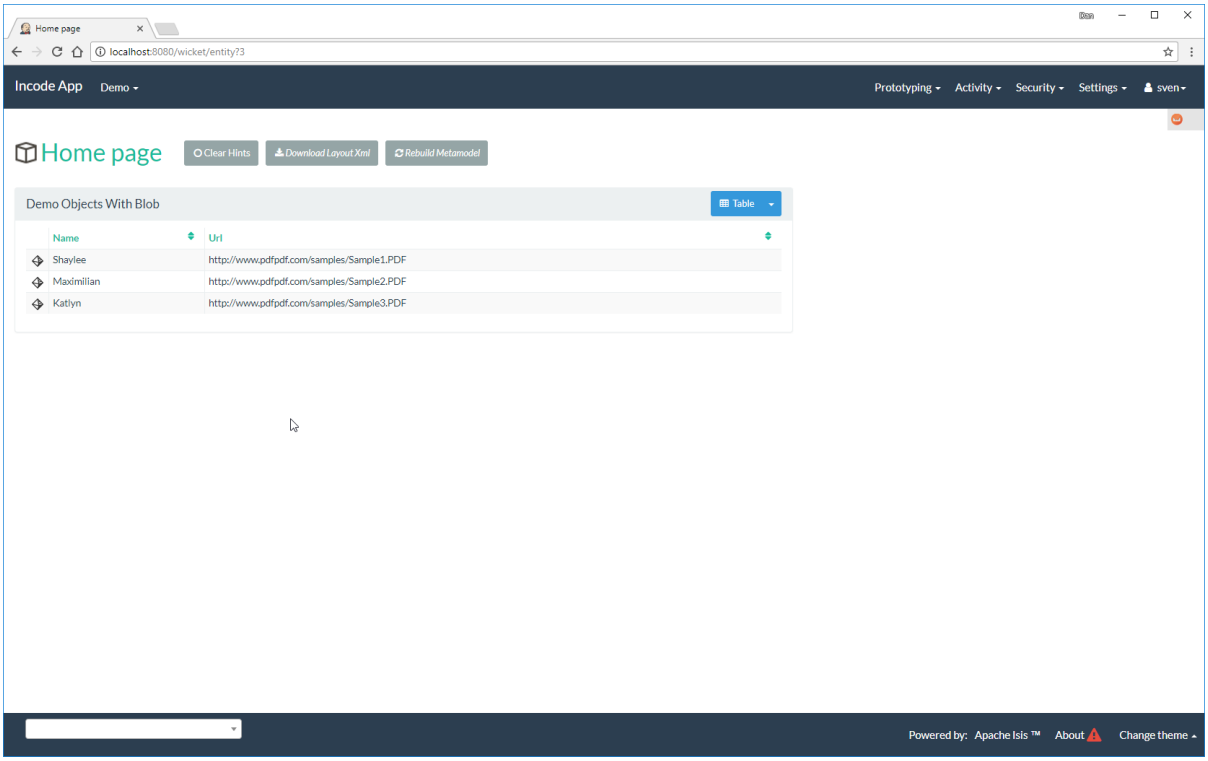
Table of Contents

This component (`isis-wicket-pdfjs`) allows a BLOB containing an PDF to be rendered as a panel using pdf.js.

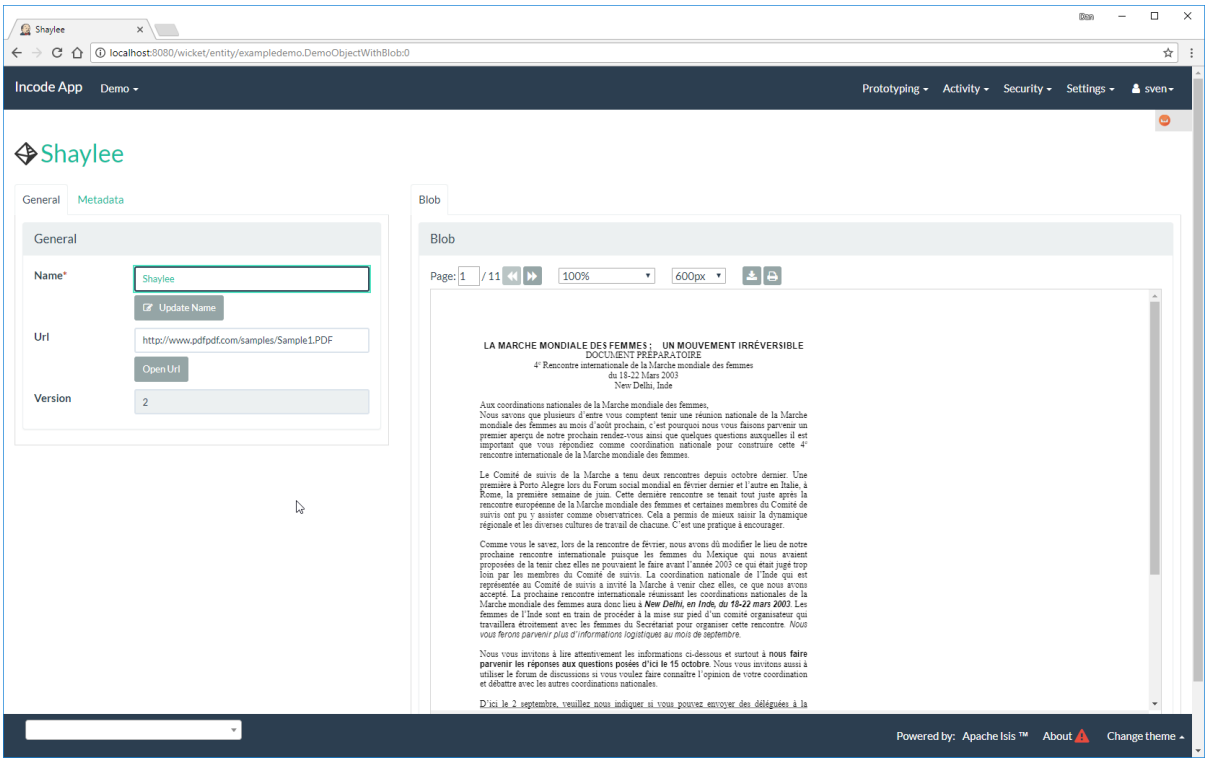
Screenshots

The module's functionality can be explored by running the quickstart with example usage using the `org.incode.domainapp.example.app.modules.ExampleDomWktPdfJsAppManifest`.

A home page is displayed when the app is run:



The component lets the user page through the PDF document, zoom in and out, and change the height of the panel. In addition, the user can download PDF and print the PDF.



API & Usage

The PDF viewer is only enabled for `Blob` properties that contain a PDF. These must also be explicitly annotated with the `@PdfJsViewer` annotation:

```
public @interface PdfJsViewer {
    int initialPageNum() default 1;                ①
    Scale initialScale() default Scale._1_00;      ②
    int initialHeight() default 800;               ③
}
```

① The page to render the first time this particular domain object('s property) is rendered..

② The scale to render; defaults to 100%.

③ The height of the panel; defaults to 800px.

For example:

```
@PdfJsViewer(
    initialPageNum = 1,
    initialScale = Scale._0_75,
    initialHeight = 600
)
public Blob getBlob() {
    return this.blob;
}
```

The `Scale` enum corresponds to the scale drop-down in the view, and is defined as:

```
public enum Scale {
    AUTOMATIC,      ①
    ACTUAL_SIZE,
    PAGE_FIT,
    PAGE_WIDTH,
    _0_50,          ②
    _0_75,
    ...
    _4_00;          ③
}
```

① predefined scaling strategies, depend on the width/height of the panel available to render in

② 50%

③ 400% etc

SPI Services

Often a user may need to browse through many documents at a time, for example to process a number of scanned documents. To fit their particular screen, they may want to adjust the zoom level and/or height of the panel. It would however be very tiresome of the next document viewed reset to the defaults specified in the `@PdfJsViewer`.

Related, support a user views a first document and navigate to some other page. She then moves on to second document, and then goes back to the first document once more. It would again be annoying if she had start back at page 1 and navigate once more to the page they were previously at.

To support these two use cases the component therefore provides an optional SPI service. Implementations of this SPI service can provide hints (`Advice`) which override the defaults of the `@PdfJsViewer` annotation.

The SPI is defined as:

```
public interface PdfJsViewerAdvisor {

    class InstanceKey { ... }                                   ①
    class Advice { ... }                                        ②

    Advice advise(InstanceKey key);                             ③

    void pageNumChangedTo(InstanceKey key, int pageNum);        ④
    void scaleChangedTo(InstanceKey key, Scale scale);          ④
    void heightChangedTo(InstanceKey key, int height);          ④
}
```

① Value type that identifies an object type and identifier, its (PDF) property and the user that is viewing the object.

② Value type that specifies the page number, scale and height to render the object

③ The main SPI called by the viewer;

④ Updates the service implementation whenever the user updates the page number, scale or height for a particular object/property/user (ie `ViewerKey`).

There can be multiple implementations of this service; the first implementation to return a non-null `Advice` is used. If there *are* multiple implementations, then *all* are called whenever the user updates the view.

The demo application shows one such implementation that fulfills the two user goals:

- it remembers the scale/height for each object type/property (per user), so that any other documents of the same type are shown with the same layout
- it remembers the page that each user was viewing a document, so resumes at that page if the same document is viewed more than once

To do this the demo implementation relies upon the inner value types `InstanceKey.TypeKey` and `Advice.TypeAdvice` which track the hints at the object type — rather than instance — level.

How to configure/use

Classpath

Add the component to your project's `dom` module's `pom.xml`:

```
<dependency>
    <groupId>com.eurocommercialproperties.pdfjsdemo</groupId>
    <artifactId>ecp-wicket-pdfjs-cpt</artifactId>
    <version>1.15.1.1</version>
</dependency>
```

Check for later releases by searching Maven Central Repo.

For instructions on how to use the latest `-SNAPSHOT`, see the contributors guide.

Bootstrapping

In the `AppManifest`, update its `getModules()` method, eg:

```
@Override
public List<Class<?>> getModules() {
     return Arrays.asList(
        ...
        org.isisaddons.wicket.pdfjs.cpt.PdfjsCptModule.class,
        ...
     );
}
```

Configuration Proeprties

Set up the facet factory in `isis.properties` (or in the `AppManifest#getConfigurationProperties()`):

isis.properties

```
isis.reflector.facets.include=\
    org.isisaddons.wicket.pdfjs.cpt.applib.PdfJsViewerFacetFromAnnotationFactory
```

Known Issues

The Javascript isn’t fully thread-safe, so avoid having more than one instance of this component rendered on the page at the same time. This also means that the component should never be rendered in a table ("compact" view).

Dependencies

Maven can report modules dependencies using:

```
mvn dependency:list -o -pl modules/wkt/pdfjs/impl -D excludeTransitive=true
```

which, excluding Apache Isis itself, returns these compile/runtime dependencies:

```
de.agilecoders.wicket:wicket-bootstrap-core:jar:0.10.16
```

For further details on 3rd-party dependencies, see:

- l0rdn1kk0n/wicket-bootstrap

It also includes a Javascript dependency on Mozilla PDF.js.