Docx (MS Word) Library

Table of Contents

This module (`isis-module-docx`) provides a mail-merge capability of input data into an MS Word `.docx` templates. The generated output document is either Word `.docx` or Acrobat `.pdf`.

Exporting to PDF requires more memory, both heap and permgen. If the PDF generation hangs then increase the memory settings, eg `-Xmx1024m -XX:MaxPermSize=128m`

The module consists of a single domain service, `DocxService`. This provides an API to merge a `.docx` template against its input data. The input data is represented as a simple HTML file.

The service supports several data types:

- plain text
- rich text
- date
- bulleted list
- tables

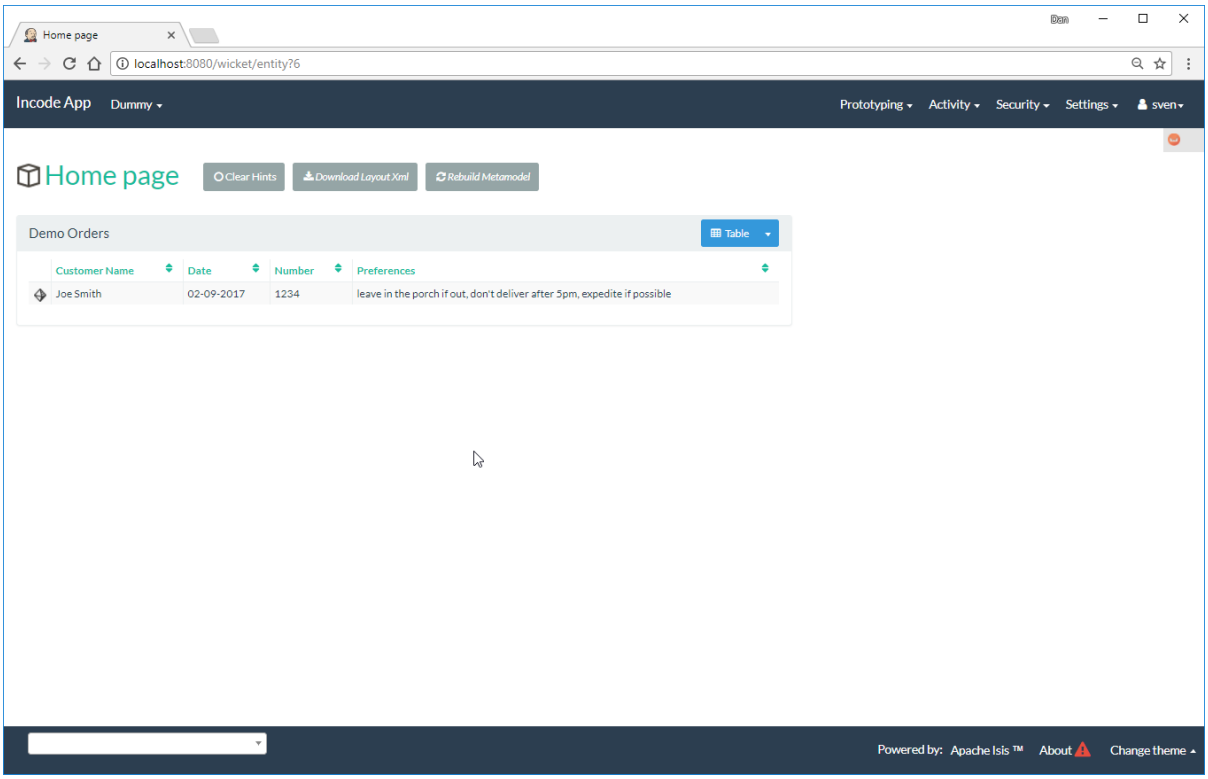
The implementation uses docx4j, guava and jdom2. Databinding to custom XML parts (the `.docx` file format's in-built support) is **not** used (as repeating datasets - required for lists and tables - was not supported prior to Word 2013).

Screenshots and Usage

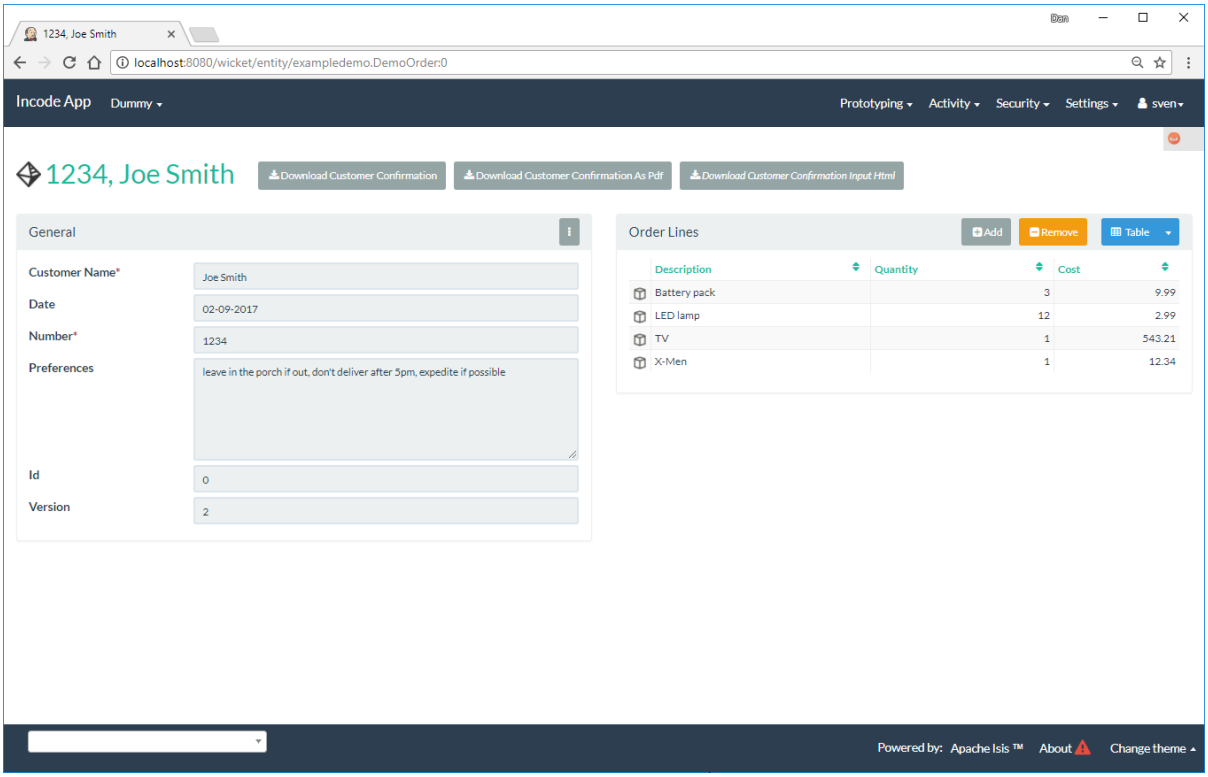
The module's functionality can be explored by running the quickstart with example usage using the `org.incode.domainapp.example.app.modules.ExampleDomLibDocxAppManifest`.

Installing the Fixture Data

A home page is displayed when the app is run:

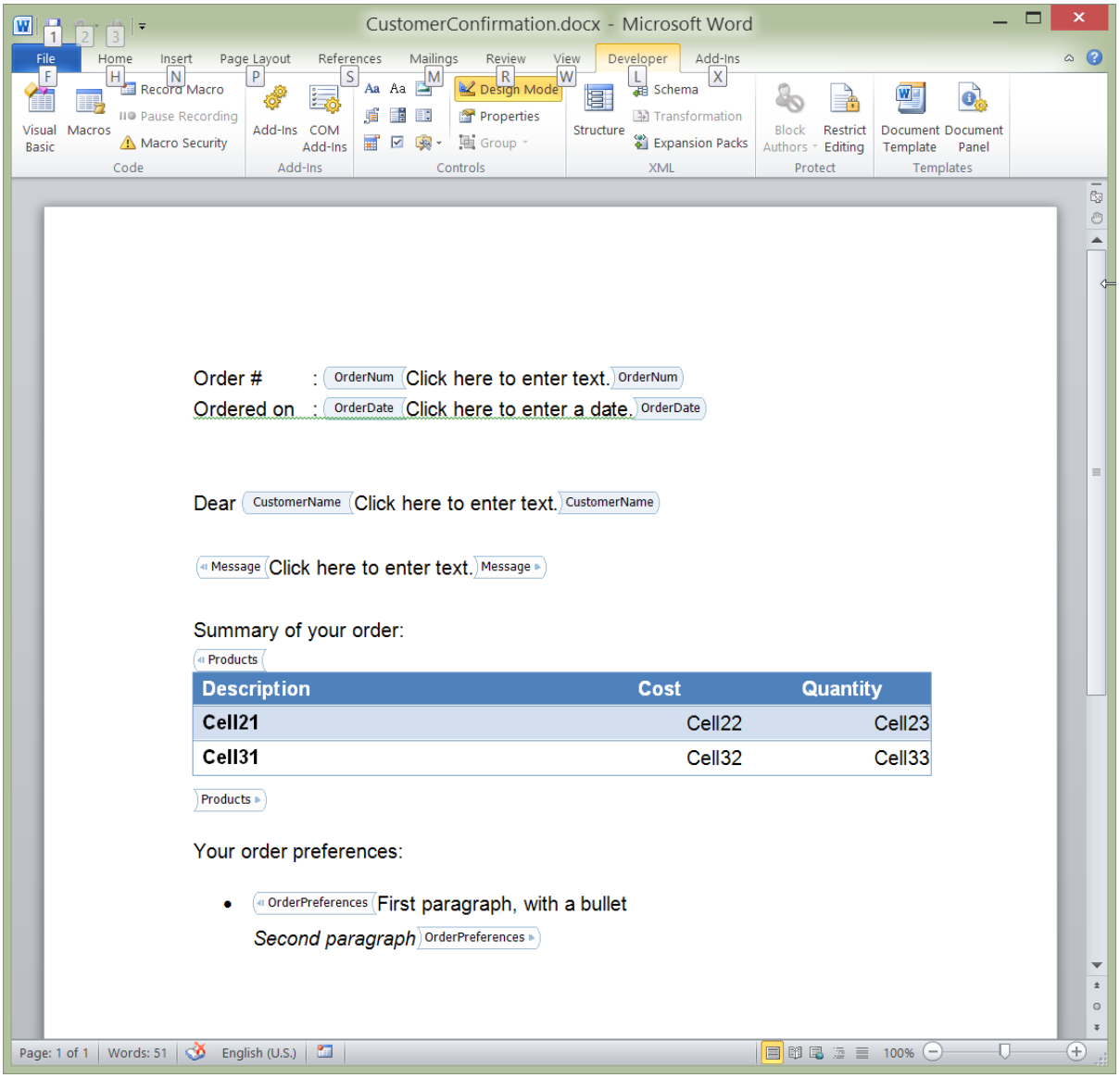


... creates a single demo `Order` entity, with properties of different data types and a collection of child (`OrderLine`) entities:



The `.docx` template

The template `.docx` itself is marked up using smart tags, as specified on the [DEVELOPER] tab (see "How to show the DEVELOPER tab in Word").



The actual `.docx` used in the example app can be found here.

Generating the Document

In the example app's design the `CustomerConfirmation` example domain service is in essence an intelligent wrapper around the `CustomerConfirmation.docx` template. It contributes two actions to `Order`, the more significant of which is `downloadCustomerConfirmation()`.

The `.docx` is simply loaded as a simple resource from the classpath:

```
@DomainService
public class CustomerConfirmation {

    private WordprocessingMLPackage wordprocessingMLPackage;

    @PostConstruct
    public void init() throws IOException, LoadTemplateException {
        final byte[] bytes = Resources.toByteArray(Resources.getResource(
                                this.getClass(), "CustomerConfirmation.docx"));
        wordprocessingMLPackage = docxService.loadPackage(new ByteArrayInputStream
(bytes));
    }
    ...
}
```

A more sophisticated service implementation could perhaps have retrieved and cached the `.docx` template bytes from a `Blob` property of a `CommunicationTemplate` entity, say.

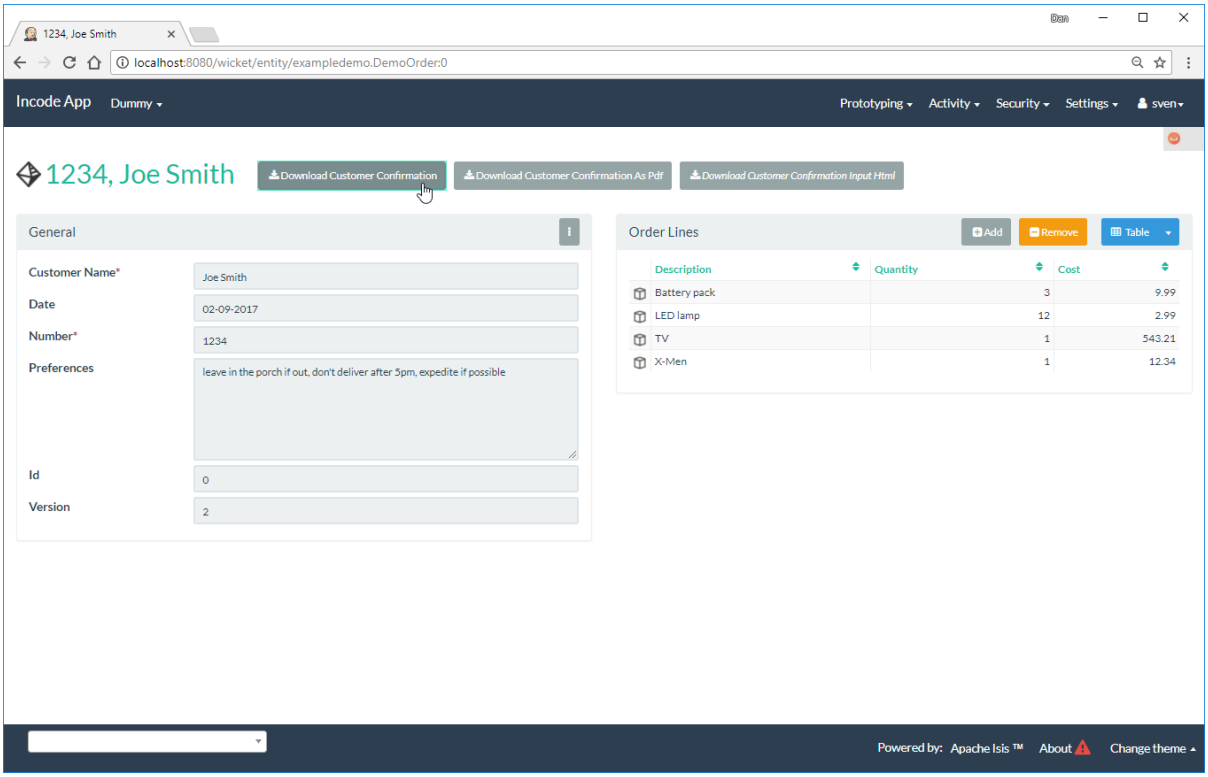
Then, in the `downloadCustomerConfirmation` contributed action the `CustomerConfirmation` performs several steps:

- it converts the `Order` into the HTML input for the `DocxService`
- it calls the `DocxService` to convert this HTML into a `.docx` file
- finally it emits the generated `.docx` as a Blob; in the web browser this is then downloaded:

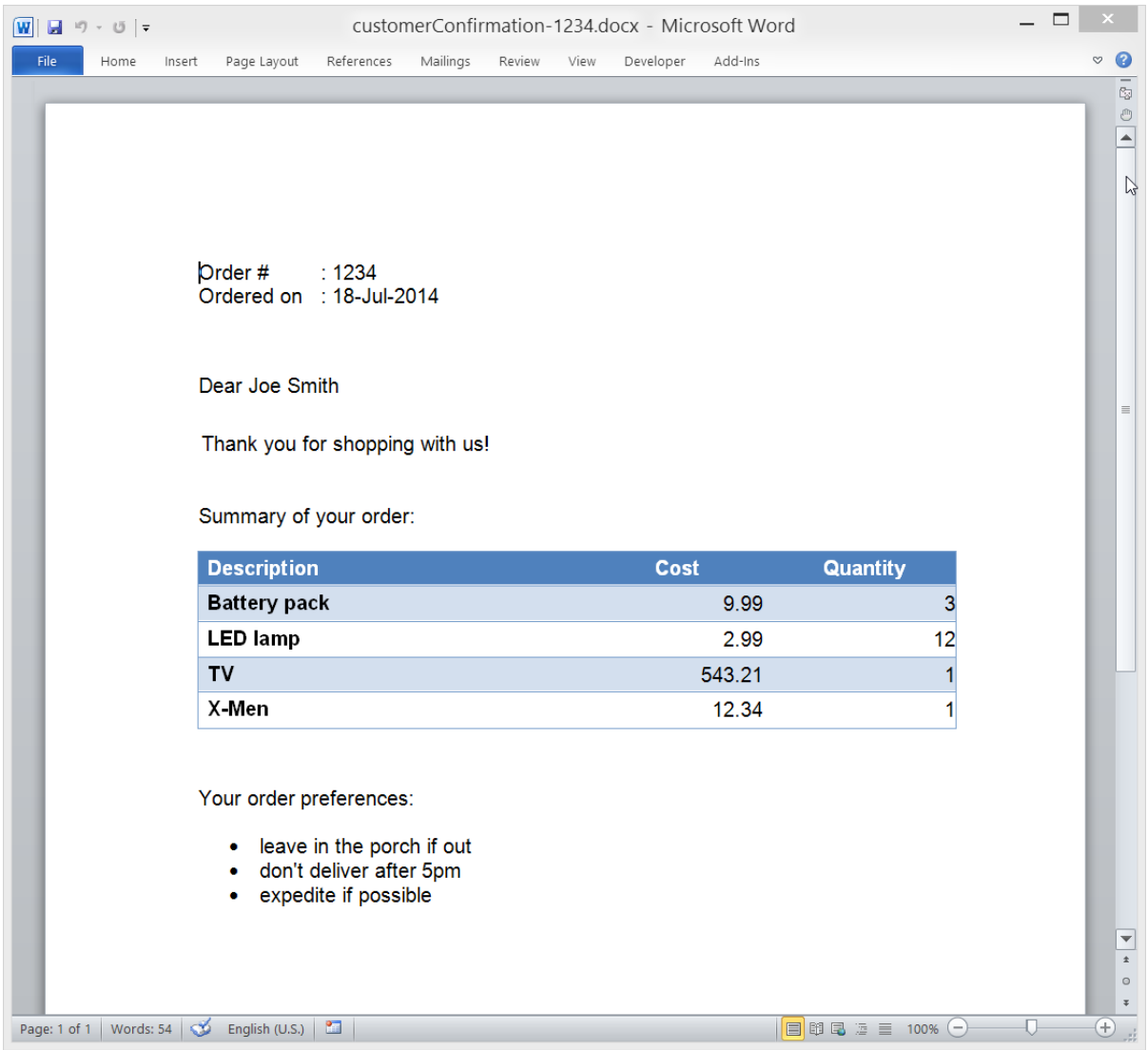
This can be seen below:

```
public Blob downloadCustomerConfirmation(
        final Order order) throws IOException, JDOMException, MergeException {

    final org.w3c.dom.Document w3cDocument = asInputW3cDocument(order);

    final ByteArrayOutputStream docxTarget = new ByteArrayOutputStream();
    docxService.merge(
        w3cDocument, wordprocessingMLPackage, docxTarget, DocxService.MatchingPolicy
.LAX);

    final String blobName = "customerConfirmation-" + order.getNumber() + ".docx";
    final String blobMimeType =
        "application/vnd.openxmlformats-officedocument.wordprocessingml.document";
    final byte[] blobBytes = docxTarget.toByteArray();

    return new Blob(blobName, blobMimeType, blobBytes);
}
```

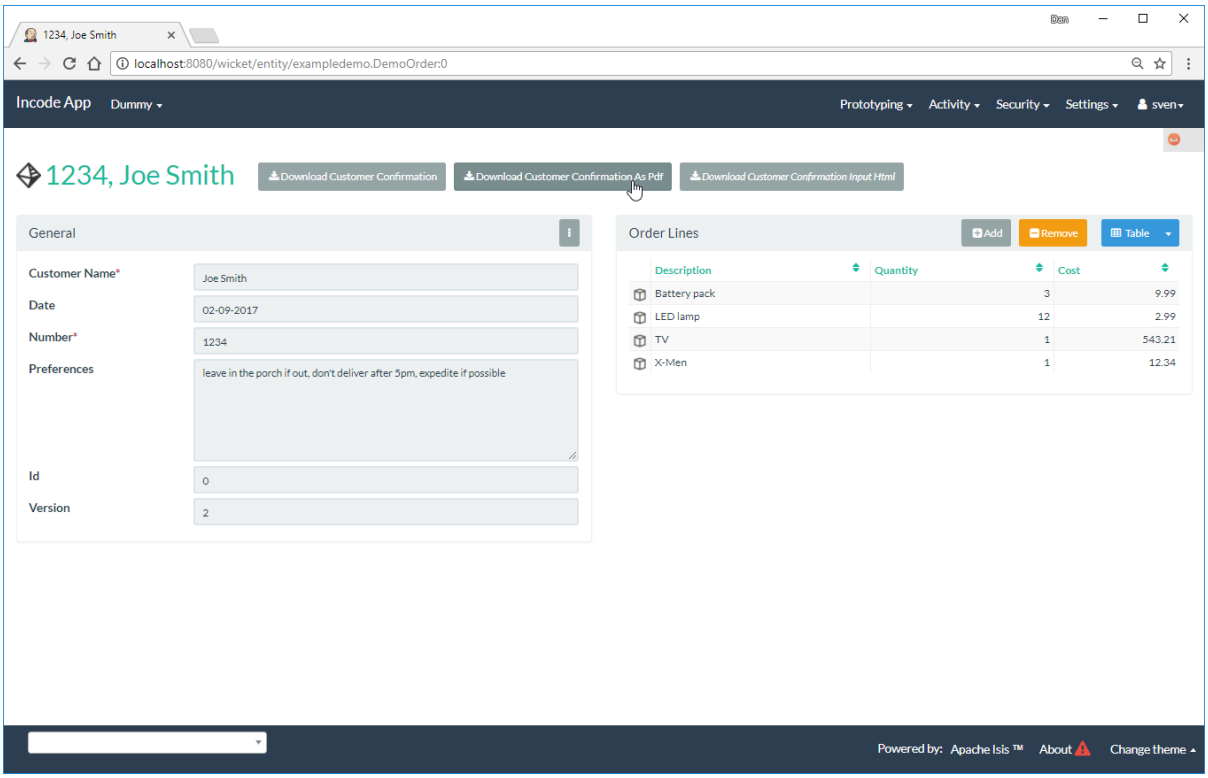
Invoking this action is shown below:



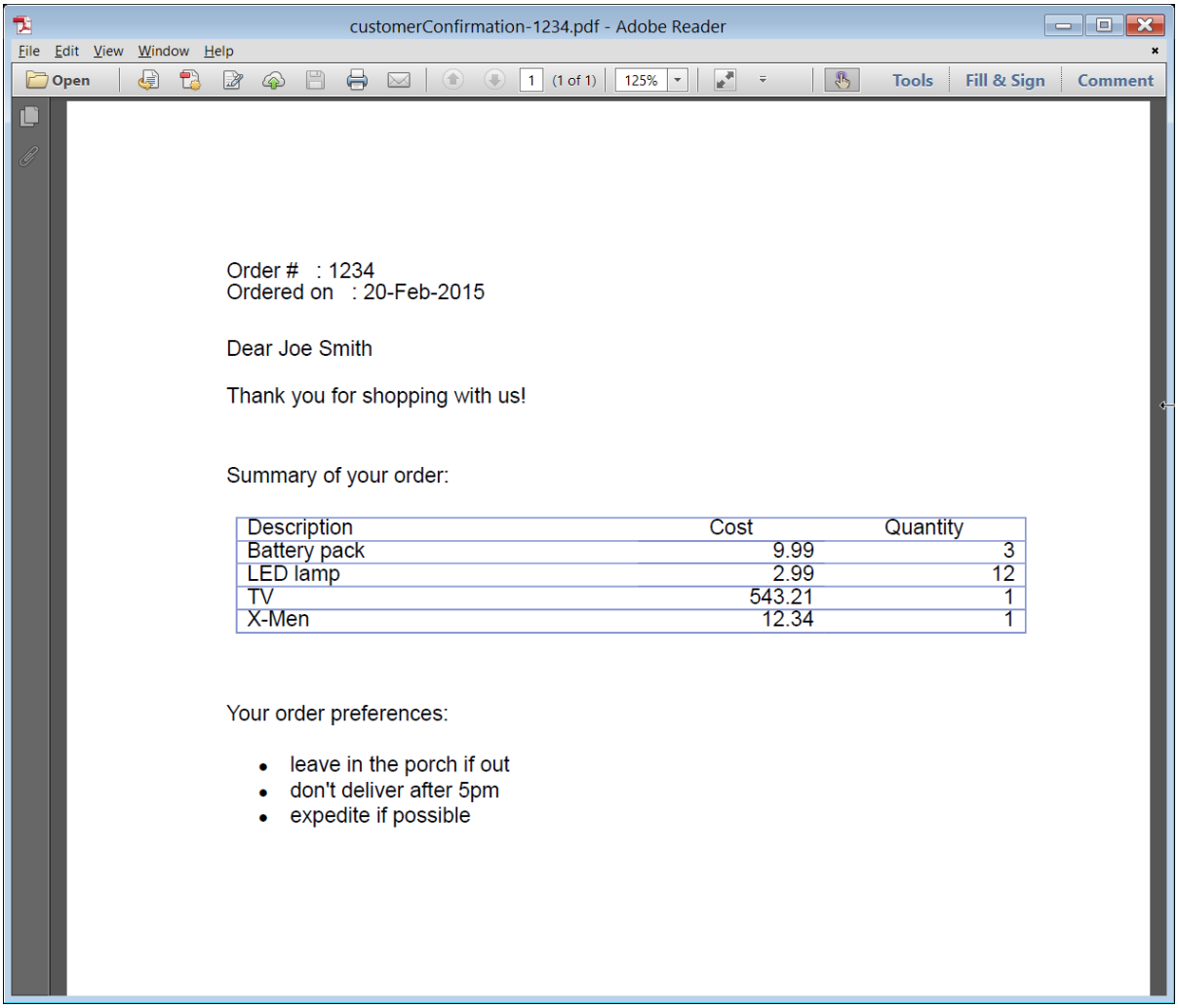
which when opened in MS Word looks like:



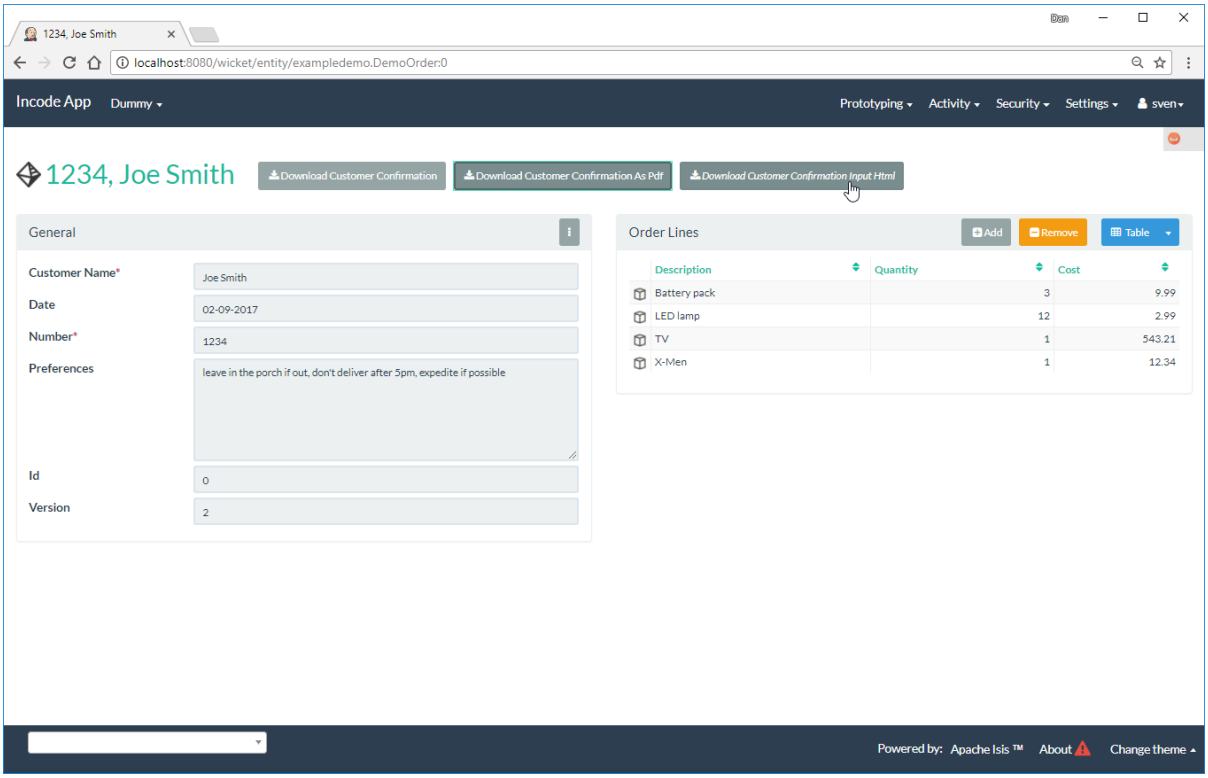
A similar action downloads the generated document as a PDF:



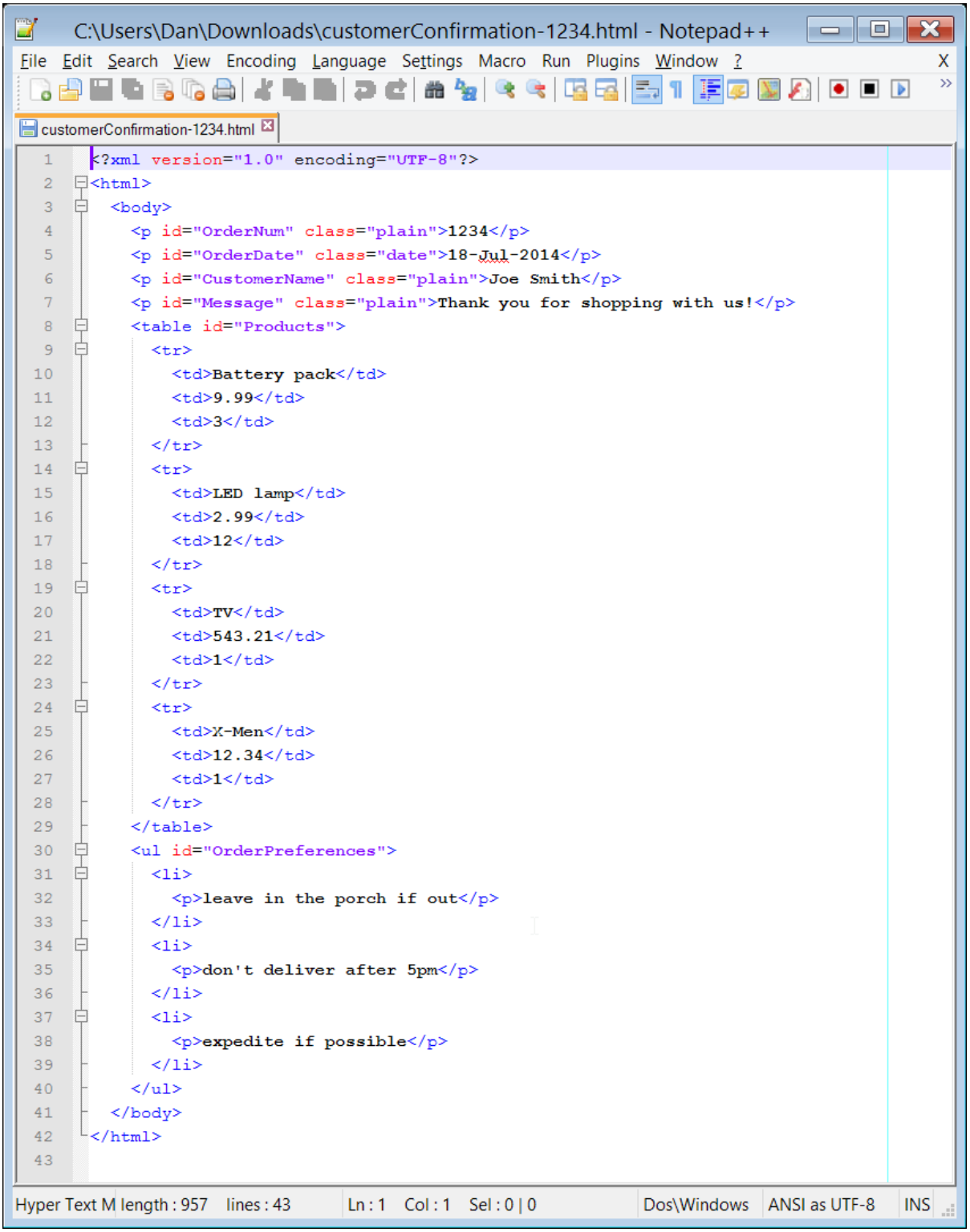
which when opened in Acrobat looks like:



The `CustomerConfirmation` service also contributes a second (prototype) action to allow the input HTML document (fed into the `DocxService`) to be inspected:



which when opened in a simple text editor looks like:



Note how the table rows are repeated for each `OrderLine` item, and similarly a new bullet list for each `Order` preference.

How to configure/use

Classpath

Update your classpath by adding this dependency in your dom project's `pom.xml`:

```
<dependency>
    <groupId>org.isisaddons.module.docx</groupId>
    <artifactId>isis-module-docx-dom</artifactId>
    <version>1.13.0</version>
</dependency>
```

Check for later releases by searching [Maven Central Repo](http://search.maven.org/#search|ga|1|isis-module-docx-dom).

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.module.docx.DocxModule.class,
    );
}
```

API & Implementation

The main API is:

```
public void merge(
         String html,
         InputStream docxTemplate,
         OutputStream docxTarget,
         MatchingPolicy matchingPolicy,     ①
         OutputType outputType)             ②
     throws LoadInputException,
            LoadTemplateException,
            MergeException
```

① The `MatchingPolicy` specifies whether unmatched input values or unmatched placeholders in the template are allowed or should be considered as a failure.

② The `OutputType` specifies the type of the generated output. Two possible types are supported: `DOCX` and `PDF`.

Overloaded versions of the `merge(…)` method exist:

- the `html` may instead be provided as a `org.w3c.dom.Document`
- the `docxTemplate` may instead be provided as a doc4j `WordprocessingMLPackage` (an in-memory object structure that could be considered as analogous to an w3c `Document`, but representing a `.docx`).

The `WordprocessingMLPackage` can be obtained from a supplementary API method:

```
public WordprocessingMLPackage loadPackage(
        InputStream docxTemplate)
    throws LoadTemplateException
```

This exists because the parsing of the input stream into a `WordprocessingMLPackage` is not particularly quick. Therefore clients may wish to cache this in-memory object structure. If calling the overloaded version of `merge(…)` that accepts the `WordprocessingMLPackage` then the service performs a defensive copy of the template.

In the example app the `CustomerConfirmation` domain service does indeed cache this package in its `init()` method.

input HTML

The input data is provided as an XHTML form, and the service merges using the `@id` attribute of the XHTML against the tag of the smart tag field in the `.docx`.

To specify a **plain** field, use:

```
<p id="CustomerId" class="plain">12345</p>
```

To specify a **date** field, use:

```
<p id="RenewalDate" class="date">20-Jan-2013</p>
```

To specify a **rich** field, use:

```
<p id="PromoText" class="rich">
    Roll up, roll up, step right this way!
</p>
```

To specify a **list** field, use:

```
<ul id="Albums">
    <li>
        <p>Please Please Me</p>
        <p>1963</p>
    </li>
    <li>
        <p>Help</p>
    </li>
    <li>
        <p>Sgt Peppers Lonely Hearts Club Band</p>
        <p>1965</p>
        <p>Better than Revolver, or not?</p>
    </li>
</ul>
```

To specify a **table** field, use:

```
<table id="BandMembers">
    <tr>
        <td>John Lennon</td>
        <td>Rhythm guitar</td>
    </tr>
    <tr>
        <td>Paul McCartney</td>
        <td>Bass guitar</td>
    </tr>
    <tr>
        <td>George Harrison</td>
        <td>Lead guitar</td>
    </tr>
    <tr>
        <td>Ringo Starr</td>
        <td>Drums</td>
    </tr>
</table>
```

Generated output

For simple data types such as plain text, rich text and date, the service simply substitutes the input data into the placeholder fields in the `.docx`.

For lists, the service expects the contents of the placeholder to be a bulleted list, with an optional second paragraph of a different style. The service clones the paragraphs for each item in the input list. If the input specifies more than one paragraph in the list item, then the second paragraph from the template is used for those additional paragraphs.

For tables, the service expects the placeholder to be a table, with a header and either one or two body rows. The header is left untouched, the body rows are used as the template for the input data. Any surplus cells in the input data are ignored.

Known issues

None known at this time.

Dependencies

Maven can report modules dependencies using:

```
mvn dependency:list -o -pl modules/lib/docx/impl -D excludeTransitive=true
```

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

```
org.jdom:jdom2:jar:2.0.5
org.docx4j:docx4j:jar:3.2.1
```

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

- JDOM
- docx4j