Pages / Helium Documentation Home / Helium Beginner's Tutorial

Lesson 9: CSV Upload and Parsing

Created by Adriaan Klue, last modified by Jacques Marais on Sep 05, 2017

> *As a Shop Owner I want to do bulk updates to my stock levels by uploading CSV files.*

- Lesson Outcomes
- New & Modified App Files
- File Upload
- Parsing a Blob as a CSV
- Lesson Source Code

Lesson Outcomes

By the end of this lesson you should know how to upload files and how to extract object data from an uploaded file if it was a CSV.

New & Modified App Files

`./model/objects/FileUpload.mez`

`./model/objects/StockUpdate.mez`

`./web-app/lang/en.lang`

`./web-app/presenters/StockLevelsUpdate.mez`

`./web-app/views/StockLevelsUpdate.vxml`

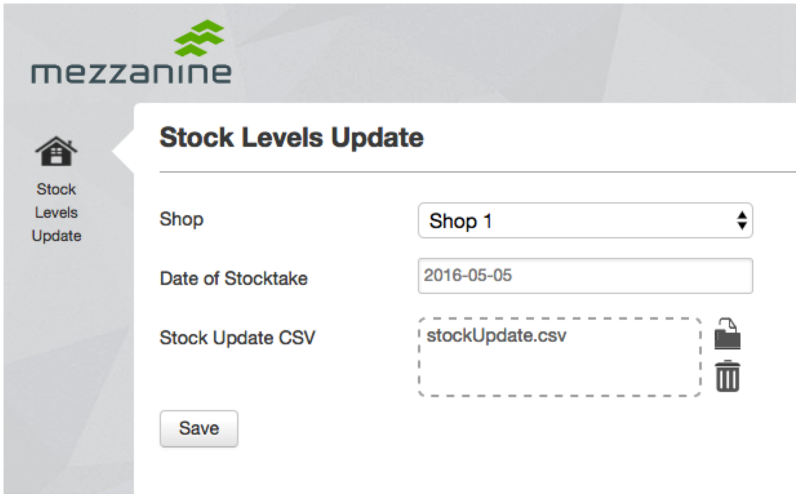
File Upload

The Helium platform allows for a file of any extension to be uploaded and saved as a blob data type. This is done with the `<fileUpload>` widget bound to a **`blob`** attribute of an object (which is just how the widget works - you can not bind it directly to a **`blob`** primitive).

view snippet

```
<fileupload label="fileupload.stock_update_CSV">
    <binding variable="fileUpload">
        <attribute name="data" />
    </binding>
</fileupload>
```

Add the above, as well as the other widgets depicted in the screenshot below, to a new view accessible by a shop owner.



Stock Levels Update

Shop: Shop 1

Date of Stocktake: 2016-05-05

Stock Update CSV: stockUpdate.csv

(With the `<select>` widget bound to a `selectedShop` variable, remembering that each shop owner can be linked to more than one shop.)

Parsing a Blob as a CSV

Our uploaded file will be a bulk update of the shop owner's stock levels, for example:

```
stockName,level,price
Corn,200,100
Wheat,300,50
Rice,150,50
Beans,400,60
Groundnut,440,10
Cowpea,100,50
A-grade grain fertilizer,100,100
B-grade grain fertilizer,200,40
A-grade legume fertilizer,150,120
B-grade legume fertilizer,300,50
```

The first, single header line for column names is required. Column names must match the attribute names as defined in the associated object type. For example, an object instance's `first_name` attribute will only be populated from the CSV file if the CSV file has a column with a header that is "first_name". Column headers that don't match attributes from the associated object type will be ignored.

Any parsing errors will result in the entire transaction being rolled back. Detailed feedback to users in terms of where the parser failed is a work in progress.

Complex attributes are supported. For example the CSV file may contain a column header shop.name. This will automatically populate the name attribute of an object that is associated with the primary object through a simple relationship with the name shop. Simple relationships are one-to-one and many-to-one relationships. Other relationships aren't supported and will be ignored. The object instance that represents the named relationships will only be created if the value in the column is not null. So if none of the complex attributes access through a specific relationships is set to a non-null value, then that related object won't be created.

Our comma separated values will correspond to the attributes of the object instance we want to save (or a subset thereof, provided we don't exclude required attributes), in this case a `StockUpdate` object:

StockUpdate object

```
persistent object StockUpdate {
   string stockName;
   int level;
   decimal price;
   date stocktakeDate;

   @ManyToOne
   Shop shop via stockUpdates;
   @ManyToOne
   Stock stock via stockUpdate;
}
```

Note the presence of the `stockName` attribute, seemingly redundant because we already have a `Stock` link. Since we want the CSV uploader to be able to specify a stock name and not an object record **`UUID`**, we need `stockName` as a **`string`** attribute so that we can use it to manually map the record in the CSV to a `Stock` record.

After uploading the CSV, execute the `fromCsv()` built-in function available on an object instance.

view snippet

```
<submit label="submit.save" action="saveStockUpdate" />
```

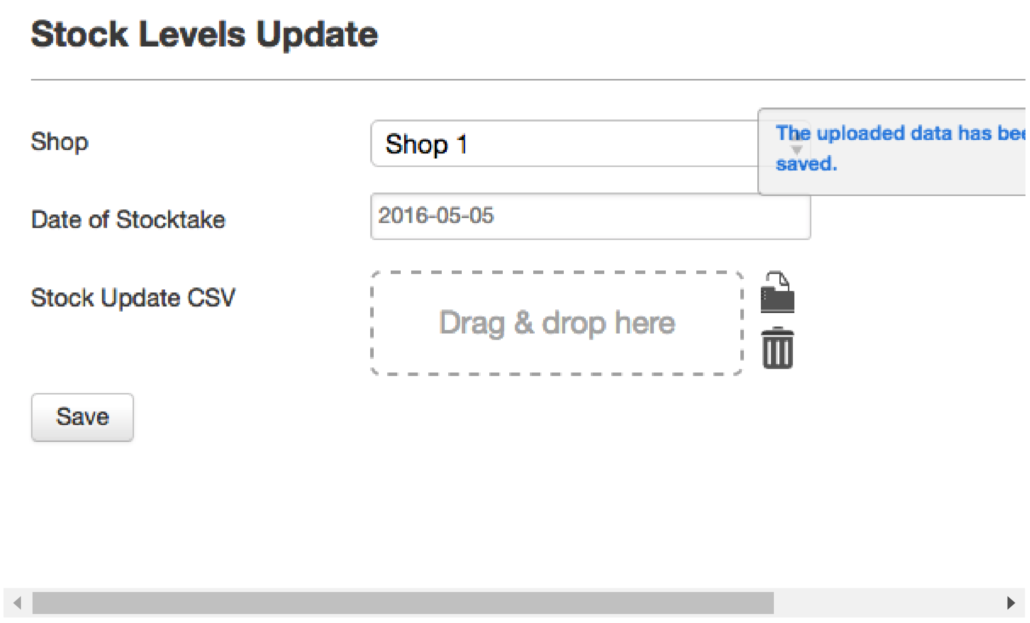
presenter snippet (StockLevelsUpdate.mez)

```
FileUpload fileUpload;

void init() {
   fileUpload = FileUpload:new();
   ...
}

void saveStockUpdate() {
   StockUpdate[] stockUpdates = StockUpdate:fromCsv(fi
   for (int i = 0; i < stockUpdates.length(); i++) {
      StockUpdate stockUpdate = stockUpdates.get(i);
      stockUpdate.stocktakeDate = selectedDateOfStockt
      stockUpdate.shop = selectedShop;
      stockUpdate.stock = getStockFromName(stockUpdate
      stockUpdate.save();
   }
   Mez:alert("alert.uploaded_data_saved");
   fileUpload = FileUpload:new();
}
```

As before, see the attached source code for the omitted parts, if necessary.

An alternative to using the otherwise redundant `stockName` attribute on `StockUpdate` would have been to use a separate, specialised, non-persistent object on which to execute `fromCsv` and copy over each value to the object we want to save.

Add a success notification in the form of a `Mez:alert()` as well to avoid a confusing user experience:



Stock Levels Update

Shop: Shop 1

Date of Stocktake: 2016-05-05

Stock Update CSV: Drag & drop here

The uploaded data has bee saved.

Lesson Source Code

Lesson 9.zip

csv upload blob fileupload file