JavascriptHttp Documentation & Manual

Cedric Zwahlen
cedric-vince@gmx.ch

June 20, 2023

Contents

1 Introduction

The aim of this package is to make it as easy as possible to *get* or *post* data from or to REST-API's from within a PDF document. Specifically, this package adds LaTeX commands to create highly customisable buttons with which to fetch, transform and display data from a remote location.
These features make use of Javascript, which means a PDF using them **only works with Adobe Acrobat Pro**. Other PDF viewers (including Acrobat Reader) may still display the interactive elements, but will not be able to run Javascript or access the internet.

2 Prerequisites

Acrobat Pro blocks Networking code by default. Because this package requires it, you must first change some settings.
Navigate to **Preferences>Security (Enhanced)**. Then, add the path to your PDF via the 'Add Folder Path' button. The javascripthttp package depends on the acrotex macro. More information about acrotex can be found here: `https://www.ctan.org/pkg/acrotex`

3 Textfields

\SimpleTextField {⟨unique identifier⟩}

Adds a writable textfield to the document. ⟨unique identifier⟩ should not be reused by other Simple elements. It must not contain an underscore '_'. Use this specifier to obtain a reference to the textfield in Javascript or as a ⟨target⟩ in other Simple elements.

3.1 Readonly

\SimpleTextFieldReadonly {⟨unique identifier⟩}

\SimpleTextFieldShortReadonly {⟨unique identifier⟩}

\SimpleTextFieldMultilineReadonly {⟨unique identifier⟩}

A readonly textfield does not accept user input. It's content can only be changed through Javascript code.

4 GET Requests

4.1 The Simple Way

\SimpleGET {⟨unique identifier⟩} {⟨label⟩} {⟨URL⟩} {⟨keypath⟩} [⟨target⟩]

If you want to get information from the internet, use this command.
This command adds a button that contains the text specified in ⟨label⟩.
The ⟨unique identifier⟩ should not be reused by other Simple elements. It must not contain an underscore '_'. This identifier can be used to obtain a reference to the button in Javascript.
⟨URL⟩ refers to the address of the REST API endpoint. Note that if you pass a string, you must surround it with either ' or ", just like you would with Javascript strings.
If the endpoint in question returns a json file, use ⟨keypath⟩ to refer to the property you would like to display in a textfield. As an example, consider the following JSON object:

```
{
  "abc" :
  [
    {
      "xyz" : "Starfruit"
    },
    {
      "xyz" : "Mango"
    }
  ]
}
```

To display the value "Mango", pass 'abc.1.xyz' (including ') as the keypath.
Finally, ⟨target⟩ specifies the name of the textfield, in which to display the value of the property specified in ⟨keypath⟩. If no such textfield exists, then nothing is displayed. This parameter is optional.

4.2 URL Interpolation

You can use URL interpolation to insert contents of a textfield into your URL. To do this, wrap the ⟨unique identifier⟩ of a textfield in curly braces where you want to insert its content in the URL. An example:

```
https://api.datamuse.com/words?rel_rhy={textfield1}
```

4.3 Using Lambdas

4.3.1 URL Argument

In some cases, ⟨URL⟩ interpolation might not be enough to produce the URL that you need. If this is the case, you can also pass a Javascript lambda (a.k.a

arrow function) as the parameter. This enables a great deal of freedom, but might present a security risk if you do not know the origin of a document.
If you pass a lambda as the ⟨URL⟩ parameter, the function must return the final URL as a string. The lambda does not take any arguments.
You can obtain a valid reference to 'this' in the lambdas context. Refer to `https://opensource.adobe.com/dc-acrobat-sdk-docs/library/jsapiref/index.html` for the Acrobat Javascript API Reference.

4.3.2 Keypath Argument

If you need to perform additional operations on received data, you can pass a Javascript lambda as the ⟨keypath⟩ parameter instead of a string. Allowing anyone to execute arbitrary code might introduce a security risk, so documents from unknown origins should not be trusted. The lambda takes two arguments; the data received from remote as a Javascript object, and a reference to 'this'. If your lambda returns a string, it will be displayed in the field specified by ⟨target⟩.
Consider the below example. A lambda is used to get a list of holidays from `https://date.nager.at/api/v3/publicholidays/2023/CH`, and then pick one at random.

```
(json,doc) => {
  // json is a top level array
  const x = Math.floor(Math.random() * json.length);
  // get localName from element x
  const h = doc.extractKeypath(json,x + '.localName',doc);
  return h;
}
```

5 POST Requests

\SimplePOST {⟨id⟩} {⟨label⟩} {⟨URL⟩} {⟨keypath⟩} {⟨factory⟩} {⟨errors⟩} [⟨target⟩]

If you need to post information to the internet, then use this command. The first four parameters of the **\SimplePOST** command work identically to the **\SimpleGET** command. Refer to GET Requests for more information.
The ⟨factory⟩ parameter takes a lambda with no arguments to construct the body of your post request. It must return a Javascript object.
If you want to display custom error messages in case an operation fails, you can pass a Javascript object specifying an error message for a given HTML error code, an example:

```
{ "400": "error 400", "300": "error 300" , "500" : "error 500"}
```

If you do not want to display any custom error messages, simply pass {} – an empty object.

6 Generic Button

\SimpleClosure {⟨id⟩} {⟨label⟩} {⟨closure⟩}

Also provided as part of this package is a button, that simply takes a lambda. This button can be used to provide other functionality to tie various components together.

7 Error Handling

Any errors that occur while posting or getting will be displayed in an alert box by default. If you do not want this, you can use the command **\ErrorField{}**. This command adds a Readonly Textfield with the ⟨unique identifier⟩ 'errorField'. If an error field exists in your document, any HTTP errors that occur will be silenced, and their error message displayed in the error field.