ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Jonathan Wafellman
SIN, Master Student

User Manual

xtigerTrans.js

Jonathan Wafellman
SIN, Master Student

Table des matières

Jonathan Wafellman
SIN, Master Student

1 Introduction

1.1 Description

The library serves to transform XTIGER templates into HTML representation. This transformation is applied recursively using deep first algorithm It uses a XHTML document to represent the transformation to apply for each XTIGER elements and types. This XHTML file has to be developed using a micro-format and has to be a valid XHTML document.

1.2 Structure of the user manual

In the second section, I define how to implement the library and launch the transformation directly in the web page.

In the third section, I will define useful information about XTIGER elements and how to deal changes.

In the fourth section I define the micro-format used for the transformation and give some examples.

2 Integration

2.1 Initialisation

Each function of the library are called through an object. This object contains all the informations needed to apply the transformation.

First, you have to create this object. It takes two arguments, a document and an URL.

```
var myObject = new  xtigerTrans(document, url);
```

The document represent the XTIGER document we want to transform. At its creation, the object extract the content of the head and the body to define the structures and types of the document. The URL corresponds to the transformations defined in a XHTML document.

For each transformation structure or XHTML document, you have to create a new object and apply the transformation on this one. You can only reuse the same object if the XTIGER document is the same (even if it has been modified) and you want to use the same transformation.

2.2 CallBack method

Each time a xtigerNode is found, the system launch a method callback to allow the user to do a mapping between the XML document and its representation. This method has to be defined by the

user and set in the object. By default it is an empty method. The method receive in argument the xtigerNode and the root of the generated representation.

```
Function myCallback(xtigerNode, htmlNode){
        mapNodes[xtigerNode] = htmlNode;
}
```

Here you can add all operations you need like adding an identifier to the htmlNode or a javascript function. It can be done in the XHTML file that content the transformation structures.

When the object of the transformation has been created, you have to set the function to it:

```
myObject.callBack = myCallBack;
```

2.3 Generate the visualisation

With this object you can launch the transformation, calling the function xtigerToHTML from the object we have created. This function takes two nodes in arguments.

```
myObject.xtigerToHTML(xtigerNode, targetNode);
```

The first node correspond to the upper node in the tree on which we want to apply the transformation. targetNode corresponds to the node that will contain the generated transformation. It has to be the clone of the xtigerNode. You can put it directly in the document, or use the variable targetNode later:

```
var targetNode = xtigerNode.cloneNode(true);
myObject.xtigerToHTML(xtigerNode, targetNode);
```

When you generate the interface, you should keep the XTIGER document and apply all transformations directly to the XHTML representation, using the mapping done with the callback.

2.4 Constraints

To avoid possible problems, the web pages containing the definition of the transformation and the result of it should be developed in XHTML. The XML languages are badly incorporated in HTML documents.

The library works in Mozilla Firefox only. The user must use it.

3 XTIGER elements

3.1 Component

The component has only one attribute: the name. It must be unique and works like an identifier. The component is a sort of box that contains elements. The name permit to refer to the structure. The children nodes of the component represent the structure. It can contains the others elements. The

component can be developed anywhere in the document and can be reused. It is important during the use of the application, if you create a new component, to add it to the object of the transformation. You have to regenerate the XHTML representation on the component modified.

```
var myObject.componentStructs[componentName] = componentNode;
```

And don't forget, if you create a new component to add it to the types:

```
myObject.unionList['anyComponent'].push('componentName');
myObject.unionList['any'].push('componentName');
```

And in all cases launch the generation of the new representation:

```
myObject.xtigerToHTML(componentNode, newNode);
```

3.2 Repeat

The repeat has three attributes: the label, minoccurs and maxoccurs. The repeat element works like a box that contains elements. In an instance of the XTIGER document, the box can be repeated and appears between minoccurs and maxoccurs times. The label serves only to give a name to the element but it can't be reuse.

If you delete a repeat element, you have just to delete its representation and if you add it, you have to generate the representation and add it to the HTML representation. The structures of the document don't depend about it.

3.3 Option

The option works like a repeat with minoccurs = 0 and maxoccurs = 1. It is a little bit different because you can define if the option element is present in an instance or not. But like the repeat element adding or removing an option doesn't modify structures or types defined.

3.4 Use

The use has four attributes : the label, the possibles types, the current type and the initial value. In a template the currentType and the initial value are not used. The label serves only to give a name to the use. It can't be reused. The types attribute defines which type of object can be inserted here. The use allow the insertion of only one element.

If the types change, you have to regenerate the use element, using simply the XTIGER node for which the types have changed:

```
myObject.xtigerToHTML(newUseNode, htmlNode);
```

3.5 Bag

The bag element corresponds to an use element in which you can insert more than one element. The types allowed are defined by three attributes : types, include and exclude. Types is a list of types allowed, where you can add types define in include and remove types defined in the exclude attribute.

The label attribute serves only to give a name, the bag can't be reused.

Like the use, if the bag's types change, you have to regenerate the bag element and to replace the old one.

4 XHTML micro-format

4.1 Constraints

First the XHTML document use to define structure is like another XHTML document. Each structure defined is a sub-tree. This sub-tree should be a direct child of the body and have a class name equals to the name of the element.

The variables are defined by their name with a # before.

For list of objects that could be represented, we use a parent's node with a class name equals to 'loop'. The use and the bag can loop only on their types; the component, repeat and option can only loop on their content (children nodes).

4.2 Rules

The document XHTML containing the transformation structures defined must be a valid XHTML. The structures must be direct child of the body node.

It finds the transformations using the class name of the node. The names defined are 'component', 'repeat', 'option', 'use', 'bag', 'anyElements', 'typesStructs'. The name 'loop' is used to define where the system has to loop and insert a list of objects.

With them you can define all the structures and the types of the target language.

The variables defined are '#types', '#typeStructs', '#label', '#name', '#minoccurs', '#maxoccurs' and '#content'. The variables you can use depend on the structure to change. Now, we will define each transformation structure.

4.3 Component

First see the following example:

```
<div class='component'>
```

```
        <div>
                        #name
        </div>
        <div class='loop'>
                        #content
        </div>
</div>
```

Here we define the transformation structure using the class name 'component'. The system will replace the component by this structure.

First it creates a block (div) containing the name of the component.

Secondary it loops to take each child and put it in a block (div). For an example of a component

```
<xt:component name='Report' >
        <h1>
                <xt:use label='title' types='string' />
        </h1>
        <xt:use label='subject' types='string' />
        <xt:bag label='content' types='anyElement anySimple' />
</xt:component>
```

it gives this representation:

```
<div class='component'>
        <div>
                Report
        </div>
        <div class='loop'>
                <h1>
                        <xt:use label='title' types='string' />
                </h1>
                <xt:use label='subject' types='string' />
                <xt:bag label='content' types='anyElement anySimple' />
        </div>
</div>
```

Of course, the transformation is recursive, so the use and bag elements will be transformed too.

4.4 Repeat

First another example :

```
<div class='repeat'>
        <div>
                #label ; occurrences between #minoccurs and #maxoccurs
        </div>
        <div class='loop'>
```

```
                    #content
        </div>
</div>
```

Here we have the structure for the repeat elements, first we develop a block (div) where the following text will appear : 'Subjects ; occurrences between 0 and 89' for example. Here the label take the value Subjects, minoccurs is equal to 0 and maxoccurs equals to 89. By default minoccurs is equal to 0 and maxoccurs is equal to infinite represented by '*'.

Here we can see the generated representation for an example:

```
<xt:repeat label='books' minoccurs='2'>
        <h1>
                <xt:use label='title' types='string' />
        </h1>
        <xt:use label='authors' types='authors' />
</xt:repeat>
```

It becomes :

```
<div class='repeat'>
        <div>
                books ; occurrences between 2 and *
        </div>
        <div class='loop'>
                <h1>
                        <xt:use label='title' types='string' />
                </h1>
                <xt:use label='authors' types='authors' />
        </div>
</div>
```

Here the repeat represents a list of books that contains each a title and authors. Here the maxoccurs equals *, that represents infinite, because it is this value by default.

4.5 Option

Why not an example ? :

```
<div class='option'>
        <div>
                #label
        </div>
        <div class='loop'>
                #content
        </div>
</div>
```

Here we have a structure quite similar to the repeat object. We don't have the attributes minoccurs

and maxoccurs. It works in the same manner.

4.6 Use

Maybe an example could help to understand:

```
<div class='use'>
        <div>
                #label  : #content
        </div>
        <select class='loop'>
                <option>
                        #types
                </option>
        </select>
        <div class='loop'>
                #typeStruct
        </div>
</div>
```

We define the structure for the use tag. We can use two variables : #types and #typeStruct. The loop class name allows the system to know what it will reproduces to represent each types. Here for example it creates a select object and for each types it put option in it.

Another variables can be used: #label and #content. The first one represent the label and the other represent the default value

The same effect after it with the #typeStruct. Here it will insert the structure of the HTML representation of the use. If it is a component it will insert it, if it is a simple type it will insert the structure define for it.

This use

```
<xt:use label='tralala' types='string number boolean' >A song</xt:use>
```

it will be represented like this:

```
<div class='use'>
        <div>
                tralala : A song
        </div>
        <select class='loop'>
                <option>
                        string
                </option>
                <option>
                        number
                </option>
                <option>
```

```
                boolean
            </option>
        </select>
        <div class='loop>
            <span>
                String
            </span>
            <span>
                number
            </span>
            <span>
                boolean
            </span>
        </div>
</div>
```

Here we have three types : string, number and boolean. The structure of the transformation corresponding to these types is the name of the type encapsulated in a span element.

4.7 Bag

An another example, what else?

```
<div class='bag'>
        <div class='loop'>
            <#types>
                #types
            </#types>
            <br>
        </div>
        <div  class='loop'>
            #typesStruct
        <div>
</div>
```

This bag example has only XHTML tag for the types. We can put the value of the type using <tagName> because the name of the tag is the same as the name of the type. The difference between the use and the bag corresponds to the number of element you can insert. The use is for only one element and it takes one of the types listed. The bag element is an insertion of more than one elements and all types defined can be used.

Here a bag element :

```
<xt:bag label='bla' types='h1 b' />
```

You can find its representation :

```
<div class='bag'>
```

```
<div class='loop'>
        <h1>
                h1
        </h1>
        <b>
                b
        </b>
</div>
<div class='loop'>
        <span>
                h1
        </span>
        <span>
                b
        </span>
</div>
```

We have two types: h1 and b. The first part is generated using simply the name of the types. The second correspond to the structure of the transformation define for this type.

4.8 AnyElement

AnyElement contains all types of the target Language. Most of the time it's the HTML tags. But you can define another like a subset of HTML. These types have to be separated by a space.

```
<div class='anyElement'>a img b h1 h2 h3 h4 </div>
```

All these types are now defined and can be used. A transformation structure is generated by default. This transformation is simply the name of the type encapsulate in a span node.

4.9 TypesStructs

This part of the XHTML allow to set special structures to the types. To define these transformations you have just to do as before using as classname the name of the type.

```
<div class='typesStructs'>
        <span class='a'>
                <p>Here is a link</p>
                <br />
        </span>
</div>
```

Now, for the types 'a', instead of generate, defined by default,

```
<span>a </span>
```

for the structure of the type a, it will generate

```
<span class='a'><p> here is a link</p><br /></span>
```