

Developing Firefox Extensions

Sven Vetsch / Disenchant
sven.vetsch@disenchant.ch

<http://disenchant.ch>

v1.0

August 19, 2007

Table of Contents

1	About this Tutorial	4
1.1	What is it about?	4
1.2	What is it not about?	4
1.3	Skills you need	4
1.4	Information and License	4
2	Firefox Extensions explained	5
2.1	Why does nobody know how to do it?	5
2.2	Interaction with the Web browser	5
2.3	File Structure	5
3	The Main Components	6
3.1	XML / XUL	6
3.2	Javascript	6
3.3	CSS	7
3.4	Other Components	7
4	The DOM Tree	8
5	Getting started with XUL	8
5.1	Comment in XUL	9
5.2	XUL Basics	9
5.3	Add a first Style to you XUL Document	10
5.4	Boxes and Textboxes	10
5.5	Spacers	12
5.6	Buttons	13
5.7	Tables	13
5.8	Groupboxes	15
5.9	Overlays and Menus	16
5.10	Combine XUL and CSS	19
5.11	XUL without Firefox	20
6	Let's play with Javascript	21
6.1	Javascript Basics	21
6.2	Split you code into different files	22
6.3	Build your own Functions	22
6.4	Accessing the DOM Tree	23
6.5	XMLHttpRequests	25
7	Interacting with XUL	27
8	Build your Extension Package	27



<i>TABLE OF CONTENTS</i>	3
9 Next Steps	28
9.1 XPCOM	28
9.2 Analyzing other Extensions	28
10 Further Reading	28
10.1 Books	29
10.2 Websites	29
11 About the Author	30
Listings	31
Images	31



1 About this Tutorial

1.1 What is it about?

We're now in the year 2007 and the Firefox web browser is one of the most used applications worldwide. From my point of view it's a fantastic philosophy behind this web browser, because it's not just another open source project. This application let you build extensions which can get full control over the whole browser and all it's behaviors and don't forget about the fact that extensions (normally) are fully platform independent.

Unfortunately mostly every time when I've talked to developers, even if these guys where really good at their job, they've got now clue of how Firefox extensions are built, they just exist and are useful. This was the point when I said to myself, that I've to change this situation and give the world a tool to easily getting into programming such extensions.

1.2 What is it not about?

This tutorial will **not** show you advanced stuff on this topic because for this there are some books out there which show you also that things and you'll find most information you need very easy on the Internet.

1.3 Skills you need

I'll try to explain everything as clearly as possible, so I think it should be understandable by everyone who has a little bit of a technical background but for sure it's a benefit if you have at least some basic knowledge in Javascript and XML.

1.4 Information and License

All the content is originally written by my (Sven Vetsch) and this document was released under the the "Creative Commons Attribution-Noncommercial-Share Alike 3.0" license (<http://creativecommons.org/licenses/by-nc-sa/3.0/>).



Every listening in this tutorial was tested with Firefox 2.0.0.6 on a Xubuntu 6.10 (Edgy Eft).



Sven Vetsch / Disenchant
<http://disenchant.ch>

2 Firefox Extensions explained

2.1 Why does nobody know how to do it?

As I've already wrote, the Firefox web browser is one of the most important application in the world and especially because of this, it's very difficult to understand, why that only a few people know how to program extensions for this web browser. You'll see, that it's not about rocket science at all and you can build simple "Hello World" extensions in about five minutes. From my point of view it's really because there are as far as I know not even one good resource on how to start with this, there are only online articles, books and so on which starts directly with stuff you probably never need if you only want to write very easy extensions which for example can automate things in the web for you. All in all, we can say that there's nothing out there on how to build basic Firefox extensions in little time. This is exactly what you'll learn by reading this paper.

2.2 Interaction with the Web browser

The extensions which are installed in Firefox always have full control over the web browser and can do anything, the browser is able to do, you should be aware of this at any time. The installed extensions are running in the so called "chrome", which means that these are authorized to do (nearly) what ever they want to do. Such extensions can for example change the layout of the web browser (there exist special "extensions" for that kind of stuff which are called Themes), modify actions like popping up a message when you try to access a predefined site, sending XMLHttpRequests (XHR) to any domain and get the response an so on, an extension can even open sockets, read/write files, so you see they are really mighty and you should really not install all extensions you think are just cool to have.

2.3 File Structure

Firefox Extensions have a more or less predefined file structure which you can find in nearly every extension you're looking at. Following you'll find the structure we'll need for this tutorial but in bigger extensions you'll find some more directories.

example.xpi (this is the extension itself)

```
/install.rdf  
/chrome.manifest  
/chrome/
```

(the following two directories will be packed in a *.jar archive)

```
/content/*  
/skin/*
```



I think this is now also a good time to take the magic from the mysterious *.xpi files. These files are just ZIP archives which contain the file structure described above and nothing more, so next time you download a Firefox extension, just unpack it and have a look what's in there and when you find a *.jar file, also just unpack it and you'll see that it's nothing special at all. So you see, there's no compilation or whatever, you just have your code, images and so on and all of this stuff will be packed into archives.

3 The Main Components

In this section I'd like to talk about the main technologies which can and will be used in Firefox extensions.

3.1 XML / XUL

At least in the last few years everyone who had to develop something which needs some kind of structured data (for example configuration files), based it on the so called Extensible Markup Language (XML).

Listing 1: A simple XML example of a user database

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <USERDATABASE>
3   <USER>
4     <ID>1</ID>
5     <NAME>Mickey Mouse</NAME>
6     <PASSWORD>aNeasIpW4u!</PASSWORD>
7   </USER>
8   <USER>
9     <ID>2</ID>
10    <NAME>Donald Duck</NAME>
11    <PASSWORD>s0m3thingS3cret!</PASSWORD>
12  </USER>
13 </USERDATABASE>
```

For Firefox extensions, we need also XML but not for having a place to store data to read them out of it as usual, here we use it in the form of the XML User Interface Language (XUL) which is used to describe the graphical user interface (GUI) for our extension. Later in this tutorial in the section "5 Getting started with XUL" at page 8, we will play around with XUL and believe me, afterwards you'll never create a GUI anymore with something else.

3.2 Javascript

Beside XUL, Javascript is the most important component we can use in an extension because it give us the possibility to create actions for our GUI and also



Sven Vetsch / Disenchant
<http://disenchant.ch>

creates all the events in the background. Javascript is the backend of every Firefox extension and everything we program, will be in this language. Against what many people think, for this extensions we don't use any high level programming languages like Java or C++, it's just Javascript all the time.

Listing 2: Just a Javascript example which displays a message box

```
1 <script type="text/javascript">
2   alert("Hello World!");
3 </script>
```

Also for Javascript we'll have a whole section ("6 Let's play with Javascript") in this tutorial at page 21, where we will have a look at the might of this scripting language.

3.3 CSS

Cascading Style Sheets or better known under the abbreviation "CSS" is a technology, which will be used mainly in the web for defining the layout of a website without having a direct connection to the site's content. You might ask now, why we should use CSS because we already have XUL which is responsible for the layout and the design of our extensions. This is a good question but there is a simple answer; It's because some stuff are much easier to realize with CSS than with XUL and there are also stuff we can't do with XUL. On the other hand, XUL has advantages which are not present in CSS, so for example only in XUL we have the possibility to interact with the Javascript because CSS is was never designed for having something to do with all other code which's used in the application the CSS is in. Overall we can say, that XUL defines the basic layout of the GUI and CSS does the fine tuning like colorize the background, defining fonts and so on.

3.4 Other Components

Here I'll show you some more components which can be used in Firefox extensions but we won't go too deep into this components to make this basic tutorial as easy as possible to understand.

- Resource Description Framework (RDF)

aaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa

- Cross Platform Component Object Model (XPCOM)

in the section "9.1 XPCOM" at page 28 aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa



```

aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa
aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa
aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa
aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa
aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa

```

4 The DOM Tree

Something we will work with all the time, when we work with Firefox extensions and also with many other things which have something to do with XML is the so called Document Object Model (DOM). This is just what we already know from the XML structure we've seen in the listing at page 6. In the mentioned example for example we have the so called "root element" which exists always only once. In the example, this root element is "USERDATABASE". After this root element we have other so called "nodes" in there. On the next deeper level we've got "USER" and again a level deeper in this tree structure we've got "ID", "NAME" and "PASSWORD". Now there are different ways to call a specific value out of this structure and I don't want to describe those now but basically because it's as I already wrote before a tree structure and so we can walk through this tree to our value. For example if we would like to get the name of the user with the value ID equal 1, our way is as described below:

```
USERDATABASE -> USER -> ID-> Mickey Mouse
```

Because you can't know who is the user with the ID equal 1, you have for example to check through all possible users until you find the id you're looking for. This might sound as a huge performance issue but it's not really because you have always to search for information in a data structure, independent if it's a database or a XML file. In a DOM tree we've got the advantage, that we don't have to go into all entries because if we have for example not only users but also cars in our XML file, but we're looking for a name of a user, then we can limit the data we've go through for our search already on the second level of the three and so we can become more specific at any level of the tree. Because for example XUL and XHTML are based on XML and so they're accessible over the DOM tree, we can get every value out of it and also change stuff on the fly. We'll see such stuff later in the section "6.4 Accessing the DOM Tree" at page 23, when we're diving into the Javascript stuff.

5 Getting started with XUL

Now I've talked too long anyway about all the things around this Firefox extensions, now let's go and start developing.



5.1 Comment in XUL

Even if XUL is not really a language, more another way of how to use XML, we've got the possibility also in XUL to comment our code. Because it's nearly similar to XHTML, you can comment your code simply by using:

```
<!-- your comment -->
```

5.2 XUL Basics

OK, let's go, below you'll see our first XUL file, just create a file with this content and call it something like *myXulFile.xul*

Listing 3: Our first XUL file

```
1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
3   Yes, this is our first XUL file.
4   <!-- and this is just a comment -->
5 </window>
```

Now you can give it a try and open it in your Firefox; You'll see nothing unless you're having a look into the source code, there you'll find everything we've written in our XUL file. Now you might ask why our text "Yes, this is our first XUL file." will not be displayed because you might have expected to see this part and the comment will be hidden but now you can see nothing else than a blank page. Think back when I wrote, that XUL is based on XML and as you might know, XML has to be well formed, which means that it has to be 100% correct. Don't worry, I didn't give you code which's not correct but in XML we don't have things which are not covered by a tag and so also this text needs to be in between of such tags. In XUL we've got the "description" tag for just displaying text, so change your source code to the one below:

Listing 4: Our XUL file with a description tag

```
1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
3   <description>
4     Yes, this is our first XUL file.
5   </description>
6   <!-- and this is just a comment -->
7 </window>
```

So, let's open it again in your Firefox and you'll see, that this time our text will be displayed. This was your first step on your way to developing Firefox extensions. Now have a look at the other two tags we've got in our code, first we've



got the XML tag. This part of the code is called "XML prologue" and covers information which will be used when the XML is used or better directly say parsed. In our example, there are not much information in it, for now we've got just the first attribute "version" which as you possibly expect stands for the XML version which was used in the XML document and the second one also doesn't need a big explanation, it's the encoding which was used and should be used when parsing the file. Then we also have the "window" tag which you might have already seen is at the beginning and the end of our file and so if we think back to the section "4 The DOM Tree" at page 8, we can see that the window tag is the root element. In this root element, we define the so called "XML namespace" trough the attribute "xmlns". Through this namespace definition, the parser knows that this file is a XUL document. Now you've learned the very basics on XUL and you're hopefully ready to dive deeper into it.

5.3 Add a first Style to you XUL Document

In our first XUL experiment we displayed a short they on a plank page and it looked very ugly, so let's make it at least a little bit better. Just add the following to our XUL file and open it once again in your Firefox web browser:

```
<?xml-stylesheet href="chrome://global/skin" type="text/css" ?>
```

Listing 5: XUL file with style

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
4 <description>
5   Yes, this is our first XUL file.
6 </description>
7 <!-- and this is just a comment -->
8 </window>
```

As you can see, it looks still ugly but we've added some style or to be more exact, we've added the default style of Firefox and as you can see from the attribute "type" it's a CSS file, which you can also access by navigate to `chrome://global/skin` with your Firefox.

5.4 Boxes and Textboxes

I've already told you in section "3.3 CSS" at page 7, that XUL is manly for two things, first for build an interface between the GUI and the Javascript code and secondly it will be used for create the basic layout. The second point is exactly what we're going to do now.



For example have a look at the following code and also try it out to see the result you'll get in your Firefox.

Listing 6: XUL file with a textbox

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
4   <description>
5     Text
6   </description>
7   <textbox id="textbox1" />
8 </window>

```

Here we've got once more a new tag which's called "textbox" and even if you didn't had a look at it in your web browser, it's clear what it is, it just creates a textbox. Unfortunately the textbox is much too big and how would it for example be possible, to place it at the right side of the description? So, now you really have to take care because right before I've talked about textboxes and now we come also to boxes but this boxes are really different from textboxes because now we're going to doing some basic layout stuff and this will be done in with two different box tags. OK, this is only a half-truth because this two tags are "hbox" and "vbox" and these are just abbreviations for `<box orient="horizontal">` and `<box orient="vertical">` so if we're correct, we've to say it's only one tag which's "box".

Of course we're lazy and we'll use the abbreviations, so lets start with one of the two box tags and let's do an example with it, based on our well known XUL file.

Listing 7: XUL file with box

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
4   <hbox>
5     <description>
6       Text
7     </description>
8     <textbox id="textbox1" />
9   </hbox>
10 </window>

```



As you can see in your Firefox, now the textbox is much smaller and they are also on the right side of the description text. This happens because the box element will group all elements which are in it and adjust them in this example horizontal because of the hbox. On default, everything is adjusted vertical but you can change this behavior globally in your XUL document by adding the attribute `orient="horizontal"` to the window tag.

5.5 Spacers

With the `box(es)` tag(s) we've done our first real layout stuff with XUL but at the moment everything is on the upper right corner of the window and as already mentioned it would be for example interesting to place our description and the textbox in the center of the window. This is exactly what we're going to do in this section, again with a new tag which's this time "spacer".

In the following example code I've placed two spacers, just have a look at it in your Firefox again.

Listing 8: XUL file with spacers

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
4 <spacer flex="1" />
5 <hbox>
6 <description>
7   Text
8 </description>
9 <textbox id="textbox1" />
10 </hbox>
11 <spacer flex="1" />
12 </window>

```

Now, let's play a little bit with the flex attribute before we really have a look at this spacers. For example let one of the both flexes equal to 1 and set the other to 2 or any other value you'd like.

As you've probably already seen, if both flex attributes in the spacer element are equal to 1 then the description and the textbox will be in the (as usual) vertical center of the screen and if you change one of these to 2 you can see that it's proportional, which means that if both flex attributes of the spacers are set to 1000 it's the same as when both are set to 1 but if one is set to 1 and the other to 2, the first one will use only 1/3 of the space which is not used by the visible elements (the description and the textbox) itself and the other spacer will take 2/3 of this space. For



example now you can as described in section "5.4 Boxes and Textboxes" at page 10, change the orient attribute of the window element to "horizontal" and you'll see that now you've got the same effect as before but horizontal and not vertical.

5.6 Buttons

The next step is now to add a button into our XUL file. Buttons are very important, because you'll use them most of the time when you need a visible element in the GUI for starting an action like sending data, call a function and so on. As you might expect, the element's tag is "button" and so let us directly add a button to our XUL file.

Listing 9: XUL file with a button

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
4   <spacer flex="1" />
5   <hbox>
6     <description>
7       Text
8     </description>
9     <textbox id="textbox1" />
10    <button label="Click Me" />
11  </hbox>
12 <spacer flex="1" />
13 </window>

```

As you can see, there's now a button on your screen which's labeled with "Click Me" trough the attribute "label". At the moment this button does nothing but later in section "7 Interacting with XUL" at page 27, we will add an action to this button so stay curious.

5.7 Tables

Now we come to a very important XUL element which's called "grid" and this element will create a table just as we know it from applications like Excel. Let's have again a look at an example before we start discussing about tables.

Listing 10: XUL file with a table

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">

```



```
4 <hbox>
5   <spacer flex="1" />
6   <grid>
7     <columns>
8       <column id="descriptions" />
9       <column id="textboxes" />
10    </columns>
11    <rows>
12      <row>
13        <description value="Text 1" />
14        <textbox id="textbox1" />
15      </row>
16      <row>
17        <description value="Text 2" />
18        <textbox id="textbox2" />
19      </row>
20    </rows>
21  </grid>
22 </hbox>
23 <button label="Click Me" />
24 </hbox>
25 <spacer flex="1" />
26 </hbox>
27 </window>
```

Looks much better now and this time we've got five new elements and not just one and I've also made some other small changes in the code, so let's have a look at it.

- **grid**
As I've already written before, the grid element is to set up a table. Everything between the grid opening and the grid closing tag should be in the form of a table. You'll see what I mean with having a look at the other four table elements.
- **columns**
This element just starts/ends the section in which you can define the used columns. You should take care about the name "columns" because there's also an element called "column" and this is something different, but it's described just below.
- **column**
The id attributes we've got here have nothing to do with what we can see afterwards in the GUI but in the column element we can give names to all of the columns for a better usability and to interact with the nodes easier but it has as already said nothing to do with the stuff we'll see afterwards in the web browser. Anyway, if you've defined some of these elements, you should have as many elements in every row.



- rows
This is the same as the columns element but this time it defines the start/end of the rows for the table.
- row
In here you can define what should be on each row in each column. In our example we've got on the first row in the first column a description with the value "Text 1" and in the second column we've got a textbox. Then there is a row closing tag and then there's once more a row element for defining the second row. You can put nearly everything in all this cells which are defined by column and row elements.

You might wonder why I've defined a second description and textbox, this is just to see the effect of the table. The I've also changed the description elements from the form `<description>Text</description>` to `<description value="Text" />`. This is exactly the same but it's easier to keep the overview and it's less to write anyway.

Now you can also define tables in XUL and make things looking at least a little bit better once more.

5.8 Groupboxes

Perhaps you've already seen some Firefox extensions, which have a configuration interface and there are different kind of options which are bordered and so it looks like one group of options. Now we'll do exactly that with the element "groupbox". As usual, first have a look at an example:

Listing 11: XUL file with a groupbox

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
4   <hbox>
5     <spacer flex="1" />
6     <groupbox>
7       <grid>
8         <columns>
9           <column id="descriptions" />
10          <column id="textboxes" />
11         </columns>
12        <rows>
13          <row>
14            <description value="Text 1" />
15            <textbox id="textbox1" />
16          </row>

```



```

17     <row>
18         <description value="Text 2" />
19         <textbox id="textbox2" />
20     </row>
21 </rows>
22 </grid>
23 <hbox>
24     <button label="Click Me" />
25 </hbox>
26 </groupbox>
27 <spacer flex="1" />
28 </hbox>
29 </window>

```

This does look much better than the examples before, doesn't it? I think I don't have to really say more about the groupbox element because you can see yourself what effect it has. It just makes a border around all the elements between the start/end of the groupbox section.

5.9 Overlays and Menus

We can see, that our code grows and grows and as you can expect, even in smaller project you'll running in the problem, that you'll lose the overview and because of this, XUL knows so called "overlays". With this overlays you can split your layout into different ones, so you can for example have one overlay for your right-click menu, one for your configuration interface, one for the main application and so on. It's not so easy to become familiar with overlays but this will really help you to write well arranged and structured XUL applications and Firefox extensions. Let's have once more a look at an example source code which contains this time a menu which's also something very important (we aware that we need two files this time).

This is our main XUL document.

Listing 12: XUL overlays / *menu.xul*

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
4 <?xul-overlay href="menuOverlay.xul"?>
5 <menubar>
6 <menu label="The Menu">
7 <menupopup>
8 <menu label="Menu1">
9 <menupopup>
10 <menuitem label="Menu1 - Item1" />

```



```

11     <menuitem label="Menu1 - Item2" />
12     </menupopup>
13 </menu>
14 <menuseparator />
15 <menu id="menu2">
16     </menu>
17 </menupopup>
18 </menu>
19 </menubar>
20 </window>

```

This is the overlay XUL document.

Listing 13: XUL overlays / *menuOverlay.xul*

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <overlay xmlns="http://www.mozilla.org/keymaster/
  gatekeeper/there.is.only.xul">
3   <menu id="menu2" label="More">
4     <menupopup>
5       <menu label="OS">
6         <menupopup>
7           <menuitem label="GNU Linux" />
8           <menuitem label="Mac OS" />
9           <menuitem label="Windows" />
10          </menupopup>
11        </menu>
12      </menupopup>
13    </menu>
14  </overlay>

```

Now start the file *menu.xul* in your Firefox web browser and you should get something like the following:



1: An overlay menu in action

So, let's dive into the source code and have a look at all the new elements we've got here.

- menubar

This is something like the root element for the whole document but this one



Sven Vetsch / Disenchant
<http://disenchant.ch>

is only for menus. Everything what has something to do with a menu has to be between a starting and an ending menubar tag so that the parser can see that it is a menu.

- menu
The menu element creates a new menu start entry, like in our example `<menu label="The Menu">`. If you put there a second menu element in the same menubar section, there will be a second start entry displayed directly on the right side of the first one. One such menu will be loaded out from the file *menuOverlay.xul* through an overlay but we'll discuss that later in this section.
- menupopup
As you can see, when you click on then menu in our example and the put your mouse cursor over the entry "Menu1" at "More", a new menu level will be displayed. This is the effect, the menupopup element has. You can also add more new menus through "menu" elements into a menupopup and so you can make menus as deep as you'd like.
- menuitem
Now here's the most important element for menus, with "menuitem" we finally add the entries we would like to have in our menu.
- menuseparator
This is just a separator which displays a horizontal line between the last and the next menuitem element.
- overlay
As already explained, an overlay document is a file which can be included in other XUL documents to split the layout into different files. There exist also overlays directly form Firefox, so that we can for example add our own menu entry into one of the standard menus. The overlay element is used to show the parser that everything between the opening and ending overlay element is part of it and has to be included in the element which has requested the include of the overlay.

So, now you know the new elements which I've used in the examples and we can go ahead. First, it's very important that you understand, that *menu.xul* includes *menuOverlay.xul* and because of that, *menuOverlay.xul* needs for example **no** window element because this is already open at the place where it will be included but instead of the window element, an "overlay" element is needed.

Let's have a better look on how overlay documents will be included. In *menu.xul* we've got `<?xul-overlay href="menuOverlay.xul"?>` which means that the document *menuOverlay.xul* has to be available for including as an overlay but at this time, **nothing** will be included. Now have a look at the place where our overlay



will be included. This happens at the element `<menu id="menu2"></menu>` but you might wonder why exactly there because we have no function or something which looks like one which will be used for including overlays. The simple trick is that you have the "id" attribute for all elements and they should be unique, not only for includes, also for working with the DOM tree, like we will do it in the section "6.4 Accessing the DOM Tree" starting at page 23. So when we have a look at the document *menuOverlay.xul*, we can find `<menu id="menu2" label="More">` and this element has just the same id as where we include it in *menu.xul*. So you see, just define which overlays should be ready for include them and then define an element just in your main document which is the same as in your overlay document and then it will be included there.

Hope you get the point on how to use overlays because it's really cool to work with such and it will make your life easier during developing Firefox extensions and depending on what you'd like to program, you have to use them anyway for interacting with the Elements you've go already in the web browser.

5.10 Combine XUL and CSS

In the section "3.3 CSS" at page 7, I've already described, what CSS is and now we're going to use it in combination with XUL for getting the possibility to really design the GUI for our extensions.

We will start again with a code example so we'll create again two files just like when we worked with the overlay before but this time it's for our main XUL document and for the CSS file.

Listing 14: XUL and CSS / *xulWithCss.xul*

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <?xml-stylesheet href="xulWithCss.css" type="text/css" ?>
4 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
5   <spacer flex="1" />
6   <hbox>
7     <spacer flex="1" />
8     <groupbox>
9       <description value="Text 1" class="test"/>
10      <textbox id="textbox1" class="test"/>
11    </groupbox>
12    <spacer flex="1" />
13  </hbox>
14  <spacer flex="1" />
15 </window>

```



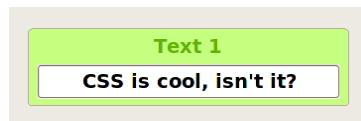
Listing 15: XUL and CSS / *xulWithCss.css*

```

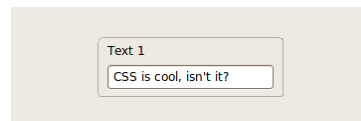
1 groupBox{
2   background-color: #C6FF80;
3 }
4
5 description{
6   color: #62B300;
7 }
8
9 .test{
10  font-size: 15pt;
11  text-align: center;
12  font-weight: bold;
13 }

```

When you've saved both files on your machine, open the main XUL document *xulWithCss.xul* in your Firefox web browser and have a look at it. What do you say? It's not that ugly anymore and to compare with the none CSS version, just rename or delete the CSS file *xulWithCss.css* or delete the line which includes it (`<?xml-stylesheet href="xulWithCss.css" type="text/css" ?>`) in *xulWithCss.xul*.



2: XUL with CSS



3: XUL without CSS

Now you've seen the difference between XUL in combination with CSS and without but because this is a tutorial on how to develop extensions for Firefox, I don't want to go deeper into the topic of CSS, even if it's interesting and it's also important that later if people are using your extension also think that these are looking good. I think out of the small example we've done here, you can expect that you can do much more with it and because of this I've got also some good references for you on CSS in the section "10 Further Reading" at page 28.

5.11 XUL without Firefox

At this point I'd like to give you the hint, that you can not only develop Firefox extensions with the stuff you'll learn and have already learned in this tutorial, you can also build (more or less) independent applications. For example let's start our small XUL document (*xulWithCss.xul*) we wrote in the last section "5.10 Combine XUL and CSS", from the command line without using the whole Firefox web browser.

Under Linux you can use:

```
firefox -chrome /path/to/your/file/myXulFile.xul
```



Sven Vetsch / Disenchant
<http://disenchant.ch>

Under Windows you can use:

aaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa aaaaaaaaaa

As you can see, you can't just build Firefox extensions with the things you'll learn and have already learned in this tutorial. You can also develop independent applications which run just in the Mozilla engine instead of the whole Firefox web browser. This will give us again much more possibilities on how to use all of that stuff.

6 Let's play with Javascript

Until now, we've just looked at the design and layout parts for developing Firefox extensions but of course this was not the most important part. Now we come to the backend part which isn't visible for the enduser, this is the Javascript code which defines all the actions and functionalities our extension will have got.

Javascript is a programming language which is really misunderstood at all by most developers. Most of them will say to you that Javascript is easy and if you can write some scripts with it you're not a programmer. For sure, you can do easy stuff with Javascript but there are much more possibilities that opening some popups, show alert boxes and so on. In this tutorial, for sure you will not learn how to become a Javascript guru but you will get an overview and afterwards you will be able to go ahead yourself in this topic.

6.1 Javascript Basics

So, let's start Javascript programming with a small "hello world" example like we've already seen one in the "3 The Main Components" section.

Listing 16: Hello World Javascript Example / *helloWorld.html*

```

1 <html>
2 <head>
3 <title>Hello World Javascript Example</title>
4 <script type="text/javascript">
5   alert("Hello World!");
6 </script>
7 </head>
8 <body>
9   This is some useless text.
10 </body>
11 </html>

```

When you made a file like in the listing above and open it in your Firefox web browser, you should see an alert box. This was now our first time of using Javascript,



even if it's not that useful yet. Here we've got a new Tag, which is the `<script>` element. This element is used to mark a section in your source as script and so the code there will be interpreted, in this case as Javascript because of the *type* attribute which shows what type of code is between the script tags.

In the example above, we've used also our first Javascript function which is `alert()`. This function just displays the string parameter we passed to the function, in our case this was "Hello World!". We'll learn how to deal with functions later in the section "6.3 Build your own Functions" at page 22.

By the way in every other in every code listing which we'll see later in this tutorial, there are no more HTML parts in Javascript examples because I think you understood the HTML stuff now and we should concentrate on the more important Javascript parts. You can see whenever it is just Javascript, by have a look at the file extension which is always in the title of the listing (it's *.js for Javascript documents).

6.2 Split you code into different files

In this part of the tutorial I just want to show you how you can split your Javascript into different files because this can become very important as soon as you're going to write bigger extensions and want to have a good structure in your code and also when you have to debug it, it's much better to have not one single big file with (X)HTML, XUL, Javascript, CSS and so on.

The following example shows you how to separate your code into different files.

Listing 17: Separate your Javascript code / *seperateYourJavascriptCode.html*

```
1 <html>
2 <head>
3 <title>Seperate your Javascript code</title>
4 <script type="text/javascript"
5     src="yourScript.js"></script>
6 </head>
7 <body>
8   This is again some useless text.
9 </body>
10 </html>
```

This listing will include the Javascript file `yourScript.js` which contains all the Javascript code you'd like to be executed at this point.

6.3 Build your own Functions

We already have seen the `alert()` function and there are of course much more functions which will be offered directly by the Javascript language. We will see some



of this built in functions later but to really learn Javascript, you should have a look in the section "10 Further Reading" at page 28 to know where you can get more information on that topic. Now to go ahead in this tutorial, you can of course write also your own functions and that's exactly what we're going to do now. As usual we'll start with an example.

Listing 18: Javascript functions / *javascriptFunctions.js*

```
1 function square(number) {  
2   var number = number*number;  
3   return number;  
4 }  
5  
6 alert(square(10));
```

So, let's have a look at the listing. We've got the keyword "function" which means, that you're going to define a new function in your Javascript. After the function keyword comes the name of the function which's in our case here "square" and in the brackets we've got the variable name (number) for the argument which will be passed to this function when it's called. Now in our square function the first thing which will be done is that the variable number will be multiplied by itself and will be again being stored in this variable. At the end of the function we've got the so called return value, which is as the name said the value which you get back from the function after calling it. After the function part, we've got an alert function but this time not a simple predefined value will be in the alert box, no this time it's the return value of the square function which's as you might expect 100 because you send the value 10 to the function and you get back the result of 10*10 which's 100. As you can see, it's really easy to define your own functions in Javascript and it's something you will definitely need to develop Firefox extensions then without it you can't do anything useful because you can't do something else than using the predefined functions by Javascript.

6.4 Accessing the DOM Tree

Do you remember the section "4 The DOM Tree" at page 8? There I've already explained the DOM tree but now we really want to start work with it through Javascript functions.

Once more we will have an easy example and afterwards we'll go through every new part we've got there. This time I've done no highlighting of new parts in the code because there is too much new stuff.

Listing 19: Accessing DOM Tree / *accessingDOMTree.html*

```
1 <html>  
2 <head>  
3 <script type="text/javascript">
```



```

4   window.onload = function () {
5
6       alert ( document.getElementById( " title " ).childNodes.length );
7       alert ( document.getElementById( " title " ).firstChild.nodeValue );
8       document.getElementById( " title " ).firstChild.nodeValue
          = " It works :) ";
9
10      var xyz = document.getElementsByTagName( " p " );
11
12      for ( var value in xyz ) {
13          alert ( xyz[ value ].childNodes[ 0 ].nodeValue );
14          xyz[ value ].childNodes[ 0 ].nodeValue = " A new
              value ";
15      }
16  };
17  </script>
18  </head>
19  <body>
20  <h1 id= " title " > A cool Title </h1>
21  <p id= " part1 " > This is Part 1 </p>
22  <p id= " part2 " > This is Part 2 </p>
23  <p id= " part3 " > This is Part 3 </p>
24  </body>
25  </html>

```

Now execute this HTML document in your Firefox and afterwards have a look at the first line in the Javascript section, here we've got a so called anonymous function because it has no name. Everything in this listing is in this anonymous function because of the part before it, the *window.onload*. This is very important because we'd like to access the DOM tree but when we directly load our script at the beginning, the DOM hasn't finished loading and so we wouldn't be able to work with it.

So, let's have a look at the two different methods of *document*, which we're using here. First we've got the *getElementById* method. To show you what we do here, I took the first line where we use this method:

```
document.getElementById("title").childNodes.length
```

In this example we give the string "title" to the method *getElementById* and as the name says, it gets the element with the id equal "title". In the next part, we get an array with all the child nodes of the title element and at the end we get the length of this array and because of there's only one element with the id equal "title", we get an alert box in the listing which shows us "1".

The we also have *document.getElementById(" title ").firstChild.nodeValue*, which gives us back the value of the first child node of the element with the id equal "ti-



tle”, so we get an alert box which says ”A cool Title”. Right after this part we something else with the same statement, we give it a new value, which is afterwards as you can see ”It works :)”.

Then we also have `getElementsByTagName(”p”)` (be aware of the additional ”s” at the end of Elements) which gets all the elements with a specific tag instead of an id. Also, until now we always just get the first child node but as I already mentioned, all the child nodes are stored in arrays, so we can get them by their index number. For this, we have a for-in loop, which runs once for every attribute in the array we give to it and because we do that for our *p* elements, it runs three times. Everytime it runs, it will first show the node value and then change it to ”A new value!”.

Of course we can do much more with Javascript on the DOM tree but just do show you the basics, that should be enough for the moment. Once more, if you’d like to learn for example how to modify attributes of tags, insert new tags and such stuff, go to the section ”10 Further Reading” at page 28 and read some of the mentioned stuff from there.

6.5 XMLHttpRequests

I think most people who’re reading this tutorial have already heard of AJAX which stands for Asynchronous Javascript and XML. In this section we’ll have a look at the core of what we call AJAX, this is the XMLHttpRequest (XHR). In the next listing, you can find an example of a XHR. Keep in mind, that I’m writing only about XHR in Firefox in this tutorial because extensions can just be used there, so to do an XHR for in Microsofts Internet Explorer you’ve to do some changes on the code and also be aware of the so called same origin policy which’s implemented in nearly every modern web browser (also Firefox of course). Because of this same origin policy it’s impossible (I don’t want to show you workarounds here because that’s not the topic) to access data from a domain which’s different from the one where the request comes from. This means, that the XHR example below won’t work if you start it on domain A which’s for example your localhost and you try to access data from domain B. This just as an information but because Firefox extensions run with higher privileges than normal websites, this will not be a problem later.

Listing 20: XMLHttpRequest / *xmlHttpRequest.js*

```
1 var XHR = new XMLHttpRequest ();
2 XHR.open(”GET”, ”./something.xml”, true);
3 XHR.onreadystatechange = function () {
4   if (XHR.readyState == 4) {
5     alert (XHR.responseText)
6   }
}
```



```

7 } ;
8 XHR.send( null )

```

So let's get into the code. First we create a new instance of a XMLHttpRequest which will then be stored into the variable *XHR*, from now on we've got a new object which represents our request. Now we start working with our request and with the *open* method, we define three things but there are two more arguments, one for a username and one for a password. Have a look at the three arguments we used here.

- Request type

The first thing we define is the type of the request we'd like to send, this can be GET,POST,PUT or HEAD. For more informations on the different HTTP headers you should have a look in the following "Request for Comments" (RFC) on HTTP.

<http://www.ietf.org/rfc/rfc2616.txt>

- Target

The only thing what's defined here is the target for our request. As soon as we send the request, it will be sent to predefined target here. Especially here, you should be aware of the same origin policy which I mentioned at the beginning of this section.

- (A)synchronous

This one I call AJAX argument most of the time because here we define if we like to send a request synchronous or asynchronous and the asynchronous is what is used for AJAX. So in our example we have an asynchronous request because of the boolean *true*, else (*false*) it would be synchronous of course.

Now we've got all the information we need to send our XHR. Because of this, we have a look at the *send* method before we go on with the *onreadystatechange* method. The *send* method right before the closing tag for the script part, this method once more just does what it's name says, it finally sends our request. So, let's go back one step to the *onreadystatechange* method. This method always reacts when the state of the request changes, for example when we get back a response and everything works normally, then we get back the status code 200. In the listing here we've got also the *readyState* method which returns the actual "ready state" and the check if it's equal four means nothing else the if it has a status code of 200. When we have *readyState* equal four then an alert function will be fired, which shows us a message box which contains the whole payload of the received HTTP package through the *responseText* method.

I think now you've got a basic understanding of Javascript, XUL and also CSS so that we can really start with developing Firefox extensions with this components.



7 Interacting with XUL

Now we've got the basic knowledge on XUL, CSS and Javascript but we didn't combine it yet and that's exactly what we'd like to do in this section of the tutorial, at least with Javascript and XUL but don't worry, because we already worked with all of that components, it's very easy to understand this stuff here. As usual we start with an example.

Listing 21: Interacting with XUL / *fileInteractingwithXul.xul*

```

1 <?xml version="1.0" encoding="iso-8859-1" ?>
2 <?xml-stylesheet href="chrome://global/skin"
   type="text/css" ?>
3 <window xmlns="http://www.mozilla.org/keymaster/
   gatekeeper/there.is.only.xul">
4 <button label="Click Me" oncommand="alert('You have
   clicked the button ')" />
5 </window>

```

As marked in the listing, you can see that the *oncommand* attribute is the new and important part here. The value of this attribute can be Javascript and so as in this example also directly a function call which's the normal case. Of course you can for example also include another Javascript document through the *script* element and the *src* attribute and call a function in there. There are also other attributes that *oncommand*, like *onchange*, *onload*, *onselect* and many more. I think it doesn't make sense to go deeper into this because in the next section we'll do a real Firefox extension and there we will not need more than this.

8 Build your Extension Package

OK, I hope you're ready now to really get into the main topic of developing Firefox extensions because in this section, we're going to develop our first working extension. It's going to be something easy but I'll try to pack as much of the learned stuff out of this tutorial as possible and put it into our extension. The goal will simply be, to have a possibility do request a random sentence out of of a XML file on a server in the Internet.

aaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa
 aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa
 aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa
 aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa aaaaaaaaaaaaa
 aaaaaaaaaaaaa aaaaaaaaaaaaa



9 Next Steps

In this section I'd like to show you what you can do at next and afterwards, you can go to the "10 Further Reading" section and get there the advanced stuff which we don't cover in this tutorial to make it as easy for you to start in the topic of developing Firefox extensions.

9.1 XPCOM

As I wrote in the "3.4 Other Components" section at page 7, I'll not go deep into the topic of XPCOM but I think it would be interesting to show you that there will also be some advanced techniques you can use during developing your extensions, so that you for example can access the local file system, open sockets and so on. We want to keep it easy and do only some basic stuff here.

aaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa
 aaaaaaaaaaaaaa aaaaaaaaaaaaaa

9.2 Analyzing other Extensions

Now you can develop basic Firefox extensions which means that you have some knowledge on Javascript and XUL which are the main components of this extensions. Beside the things mentioned in the "10 Further Reading" section at the end of this tutorial, you should also have a look at other extensions. You can find most of the available extensions at [aaaaaaaaaaaaa aaaaaaaaaaaaaa aaaaaaaaaaaaaa](#). Look for something which you think have some kind a similar function as you need for a project you'll start working on and download this extension to your harddisk. Now you can unpack the *.xpi file and afterwards eventually also the *.jar inside the *chrome* directory. Now just check out the source code and learn from others. Be aware of the licenses of different extensions because then you know under what circumstances you can copy something out of it and you find out who wrote the extension to give some credit to the developer(s) when their work was helpfully to you.

10 Further Reading

If you go the next step in developing Firefox extensions, you should have a look on the mentioned material in this section.



10.1 Books

- Programming Firefox: Building Rich Internet Applications with XUL
Kenneth C. Feldt / O'Reilly Media, Inc. (April 25, 2007)
- XUL
Jonathan Protzenko / Open Source Press (January 2007)
(sorry but this Book is only available in German)
- CSS: The Definitive Guide
Eric Meyer / O'Reilly Media, Inc. (November 7, 2006)
- JavaScript: The Definitive Guide
David Flanagan / O'Reilly Media (25. August 2006)

10.2 Websites

- <http://www.mozilla.org/projects/xul/>
The official XUL project website from Mozilla.
- http://developer.mozilla.org/en/docs/Introduction_to_XUL
An introduction to XUL from the mozilla developer center.
- <http://www.xulplanet.com/>
A library of XUL information, tutorials and downloads.
- <http://www.w3schools.com/css/>
CSS tutorial from the W3Schools.
- <http://www.html.net/tutorials/css/>
A bigger tutorial on CSS from HTML.net
- <http://www.w3schools.com/js/>
Javascript tutorial from the W3Schools.
- http://www.w3schools.com/xml/xml_http.asp
A good introduction on the XMLHttpRequest object from the W3Schools.



11 About the Author



4: Sven Vetsch

My name is Sven Vetsch and I'm also known under my nickname Disenchant which's also nearly my domain; it's <http://disenchant.ch>. Under this URL you'll find my blog, where I'm blogging mainly about web application security and web technology stuff at all, I also release some proof of concepts from time to time on my blog so you're kindly invited to have a look.

To have also a look at my profession, I'm actually a Security Tester, Analyst and Engineer at Dreamlab Technologies Ltd. (<http://dreamlab.net>) which's based in Bern (Switzerland) and I'm there the guy who's responsible for all web technologies security stuff. In the mid of September this year (2007) I'll start to study computer science at "Bern

University of Applied Sciences" (<http://www.bfh.ch>) and in my spare time I like to go to concerts, meeting friends and of course also do web hacking stuff of any kind.

For more information about me or if you've got any questions, don't hesitate to write me an e-mail to sven.vetsch@disenchant.ch



Listings

1	A simple XML example of a user database	6
2	Just a Javascript example which displays a message box	7
3	Our first XUL file	9
4	Our XUL file with a description tag	9
5	XUL file with style	10
6	XUL file with a textbox	11
7	XUL file with box	11
8	XUL file with spacers	12
9	XUL file with a button	13
10	XUL file with a table	13
11	XUL file with a groupbox	15
12	XUL overlays / <i>menu.xul</i>	16
13	XUL overlays / <i>menuOverlay.xul</i>	17
14	XUL and CSS / <i>xulWithCss.xul</i>	19
15	XUL and CSS / <i>xulWithCss.css</i>	20
16	Hello World Javascript Example / <i>helloWorld.html</i>	21
17	Seperate your Javascript code / <i>seperateYourJavascriptCode.html</i> .	22
18	Javascript functions / <i>javascriptFunctions.js</i>	23
19	Accessing DOM Tree / <i>accessingDOMTree.html</i>	23
20	XMLHttpRequest / <i>xmlHttpRequest.js</i>	25
21	Interacting with XUL / <i>fileInteractingwithXul.xul</i>	27

Images

1	Overlay Menu	17
2	XUL with CSS	20
3	XUL without CSS	20
4	Sven Vetsch	30

