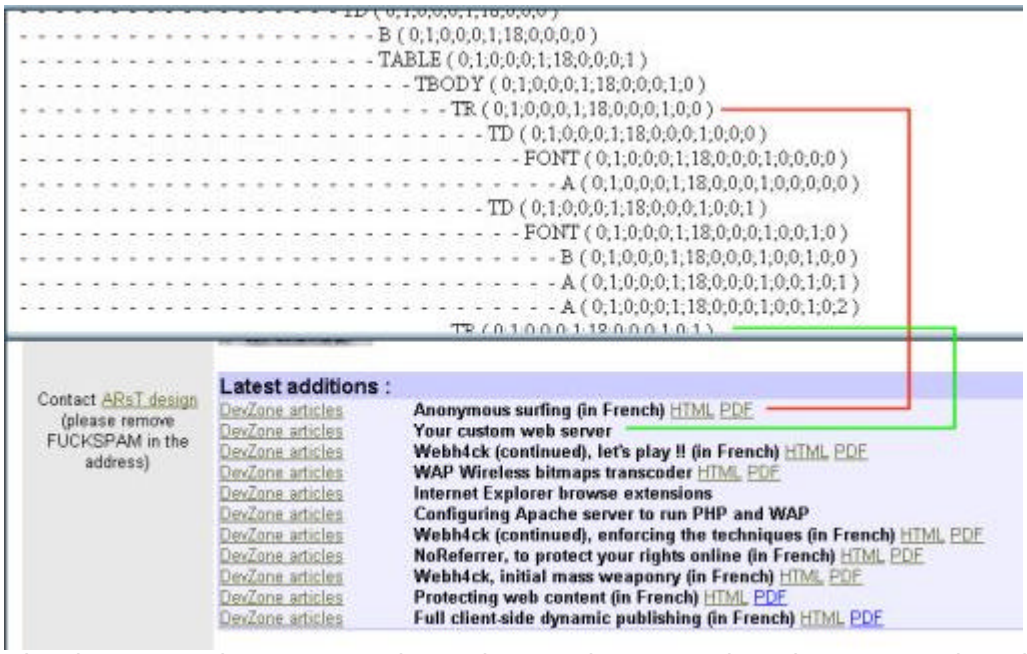
Broad HTML Analyser

Ce document répercute une étude portant sur l'analyse et l'extraction automatique d'éléments HTML. Le but étant d'être capable d'indexer du contenu et de le reconnaître dans une page Web, sans qu'il faille avoir recours à une ou plusieurs opérations manuelles.

Une des applications du procédé est le tracking de contenu sur le web. Le tracking de contenu permet en effet de récupérer du contenu à l'aide d'un automate et de constituer de l'aggrégation de contenu dans une page web. L'aggrégation de contenu est une application critique à l'heure où le web compte des milliers de source d'information, dans lesquelles il est très facile de se noyer.



Aggrégation de contenu obtenue par analyse puis extraction automatique d'une page web quelconque

1 - Analyse du contenu d'une page web

De quoi est constituée une page web ? d'un ensemble de tags HTML (norme du W3C, implémentations spécifiques de Microsoft et de Netscape). Ces tags HTML partagent une relation privilégiée. Deux tags qui se suivent sont soit parents l'un de l'autre, soit frères. Un tag parent comporte un ou plusieurs tags fils. Un tag frère est un tag au même niveau d'arborescence. Le mot-clef est bel et bien hiérarchie. Même si des effets de design permettent de mettre en œuvre des pages web très carrées ou très rondes, ceci n'est que le résultat visuel de l'application des tags, en jouant sur leurs attributs. Il n'est pas moins vrai que si l'on affiche le code source HTML on se rendra compte que les tags HTML sont articulés selon une hiérarchie.

Cette logique est due à SGML et est toujours valable dans XML. Il est donc raisonnable de croire que les relations privilégiées entre les tags HTML d'une page Web, d'une part sont déterministes, et d'autre part vont durer pendant de nombreuses années.

Cette hiérarchie est la clef de voute d'un système d'analyse très simple. En effet, un système hiérarchique peut être indexé. Voici un exemple d'indexation naturelle : si nous indexons chaque tag situé à un certain niveau par un indice démarrant à 0, et si nous modélisons la relation père fils par un point virgule suivi de l'indice du fils, alors nous ne faisons ni plus ni moins que mettre en œuvre un chemin d'accès de n'importe quel tag HTML à

partir du tag "root", le document HTML lui-même.

Ce chemin d'accès permet deux choses :

- indexer un tag dans une page web
- accéder à un tag d'une page web, pour lire son contenu, modifier son contenu

Exemple, soit la page HTML très simple suivante :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> New Document </TITLE>
<META NAME="Generator" CONTENT="EditPlus">
<META NAME="Author" CONTENT="">
<META NAME="Keywords" CONTENT="">
<META NAME="Description" CONTENT="">
</HEAD>

<BODY BGCOLOR="#FFFFFF">
         <TABLE WIDTH=87% BORDER=1 CELLPADDING=5 CELLSPACING=0>
                  <TR VALIGN=TOP BGCOLOR="#DDDDDD">
                           <TD><FONT FACE="VERDANA,ARIAL,HELVETICA" SIZE=2><B>Parameter</B></
                           </TD>
                           <TD><FONT FACE="VERDANA,ARIAL,HELVETICA" SIZE=2><B>Description</B>
                           </TD>
                  </TR>
                  <TR VALIGN=TOP>
                           <TD><FONT FACE="VERDANA,ARIAL,HELVETICA" SIZE=2><I>object</I></FON
                           </TD>
                           <TD><FONT FACE="VERDANA,ARIAL,HELVETICA" SIZE="2">The <A HREF="ine
                           </FONT>
                           </TD>
                  </TR>
                  <TR VALIGN=TOP>
                           <TD><FONT FACE="VERDANA,ARIAL,HELVETICA" SIZE=2>(<I>index</I>)
                           </FONT>
                           </TD>
                           <TD><FONT FACE="VERDANA,ARIAL,HELVETICA" SIZE="2">Optional. An int
            the index value of the element to retrieve. Integer indexes are zero-based, mean
            element in the collection has index 0. A string index is valid only if the strin
            identifier of at least one element in the document.
                           </FONT>
                           </TD>
                  </TR>
         </TABLE>
         <br>
</BODY>
</HTML>
```

Voici quelques chemin d'accès :

!DOCTYPE	0
HTML	1
HEAD	1;0
TITLE	1;0;0
BODY	1;1
TR (le premier après TABLE)	1;1;0;0;0

Le chemin d'accès de TR est 1;1;0;0;0 et non 1;1;0;0 contrairement à ce que l'on pourrait croire. Ce n'est pas une erreur, comme le montrera la suite.

Inversement, si je charge la page web et que je cherche le tag BODY, je sais par avance qu'il me suffit de

mettre en œuvre un algorithme de parcours de la hiérarchie à l'aide du chemin d'accès connu 1;1

Et les autres pages web ?

Bien sûr, chaque page web a sa propre hiérarchie. Impossible sans a priori d'exploiter les chemins d'accès de l'une dans l'autre.

Et si la page web évolue ?

Bien sûr, chaque page web est susceptible d'être modifiée légèrement ou substantiellement. Il faut bien comprendre la nuance ici entre légèrement et substantiellement.

- Légèrement, cela veut dire par exemple pour un site de news, que des news sont ajoutées par exemple quotidiennement. Cela change le chemin d'accès d'un certain nombre de tags mais d'une part les changements ne sont pas massifs, ils sont même assez facilement prédictibles. Il suffit pour cela d'analyser l'évolution du chemin d'accès d'un tag au fil des jours. Et d'autre part, cela ne change pas pas la structure du site. Le but de l'indexation est donc de faire ressortir la structure du site, une composante pour ainsi dire invariable pour de nombreux sites.
- Substantiellement, cela veut dire que le site, et en particulier la page principale, est "refondu". Les barrees de navigation sont transformées en d'autres, l'ajancement des blocs est différent. Bref, la structure est modifiée, et l'indexation est entièrement à refaire.

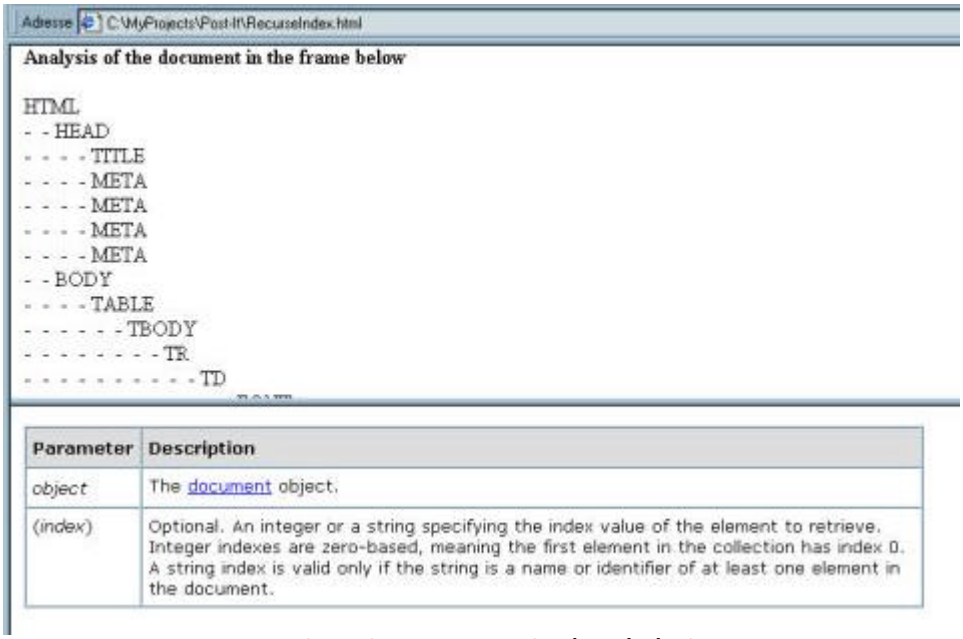
L'aggrégation de contenu exploite effectivement le fait que l'intégralité des sites de news n'opèrent que de légères modifications. De plus, lorsqu'un bloc liste les news au fil des jours, il est évident que les news peuvent être très facilement indexées : on peut savoir quel sera l'indice de la news du lendemain, et on peut savoir comment évoluent les indices des news déjà publiées.

Pour des sites très versatiles, c'est-à-dire pour ainsi dire aucun site de notre époque, ou éventuellement pour des sites mettant en jeu des modifications majeures de leur structure, un type d'analyse plus évolué, basé sur des motifs (*pattern matching* en anglais) permet de se substituer à notre approche, de façon à aggrémenter une dynamique à une certaine efficacité.

La première étape consiste à mettre en œuvre l'analyse des tags HTML.

De nombreux langages permettent d'atteindre cet objectif, mais qui peut le plus peut le moins ! Et Javascript comporte une interface de programmation appelée D.O.M. (Document Object Model). Le D.O.M. est normalisé par le W3C, et est supporté par Microsoft et Netscape. Ce DOM permet de traverser les tags selon besoin. Parfait pour ce qu'il nous faut.

Il est possible d'utiliser C++ ou Java ou un langage de script comme Python pour atteindre le même objectif. Mais il faudra soit développer complètement une interface de programmation, basée sur un analyseur lexical (LEX/YACC). Ou alors, sachant que le D.O.M. est accessible par une interface COM, il se trouve accessible via C++ notamment. On peut par ce biais adjoindre la puissance des interfaces COM à la puissance inhérente du C++ (performances, liberté, …). Ci-dessous, une approche Javascript :



Analyse d'une page web très générale

Cette page web est faite de deux cadres. Le cadre supérieur est un analyseur. Il indique l'arborescence de tags HTML de la page web, parfaitement quelconque, située dans le cadre inférieur. Pour bien rendre compte de la hiérarchie, on a introduit des tirets.

Plusieurs constats après quelques expérimentations :

- il n'y a absolument pas équivalence entre le code source HTML et la hiérarchie trouvée. Non seulement Dynamic HTML permet de rajouter des tags à la volée, ce qui distingue le D.O.M. en mémoire avec ce que l'on trouve en faisant Afficher source HTML. Mais de plus, l'analyseurs D.O.M. du navigateur web opère des transformations, déplacements, ajouts, suppression selon des choix que nous ne pouvons pas discuter, puisque les navigateurs web ne sont pas des produits fournis en OpenSource.
- un exemple d'absence d'équivalence : le tag TBODY entre le tag TABLE et le premier tag TR. Ce tag TBODY ne figure pas dans le code source HTML. A prendre en compte systématiquement.
- un tag SCRIPT mis avant le tag BODY se trouve nécessairement en tant que fils du tag HEAD, même si dans le code source HTML ce n'est pas le cas. Ceci est dû qu'à la base les tags SCRIPT sont effectivement censés se trouver dans la partie HEAD d'une page web. Mais au fil des années, les navigateurs ont permis de nombreux écarts, pour en arriver au fait qu'aujourd'hui il est possible d'insérer des tags SCRIPT absolument n'importe où. L'analyseur D.O.M. du navigateur web opère les transformations qu'il juge nécessaire. Attention, ici il peut y avoir des différences de comportement entre les navigateurs.

Le code source de l'analyseur :

```
<body bgcolor="#FFFFFF" onload="findAllElements()">
<p> </p>

<script language="Javascript">

function findAllElements()
{
	var framecontent = window.parent.frames["_2"];

	findElements(framecontent.document.all(1),0);
}

function findElements(e,nDepth)
{
	var i,obj,outputString;
```

```
    obj = e;

         outputString="";
         for (i=0;i<nDepth;i++)
                  outputString += " -  - ";

         outputString += obj.tagName+"<br>";

         document.all.myspan.insertAdjacentHTML("BeforeEnd",outputString);

         for (i=0; i<obj.children.length; i++)
         {
                  if (obj.children[i].id!="myspan")
                           findElements ( obj.children[i],nDepth+1 );
         }

}

</script>

<p><b>Analysis of the document in the frame below</b></p>

<SPAN id="myspan"></SPAN>

</body>
```

En deux mots, le cadre supérieur nommé _1 récupère un "pointeur" sur le document HTML du cadre inférieur nommé _2. Puis on obtient le premier tag par un simple **document.all(0)**. Ensuite une interface de programmation très simple basée sur l'ensemble **children**, défini pour chaque tag, permet d'accéder aux fils et aux frères par récurrence. Lorsqu'on est sur un tag, la propriété **tagName** indique le nom de ce tag, exemple HTML, alors que la propriété **id** renvoie un éventuel identifiant : elle renvoie myHTML si le tag HTML est décrit par <HTML id="myHTML" …>.

Pour chaque tag obtenu, le but du jeu est de construire visuellement la hiérarchie, grâce à l'indice de profondeur du tag courant. Une simple boucle FOR permet de construire une chaine de caractères proportionnellement grande à l'indice de profondeur.

L'astuce finale consiste à insérer du code HTML à la volée. Il nous suffit d'utiliser la méthode **insertAdjacentHTML** en référencent un tag qui sert de point d'insertion, ici **myspan**.

Bien prendre garde de ne pas insérer cette chaîne de caractères dans la page HTML que l'on analyse, car le code HTML que nous insérons à chaque étape comporte un tag HTML
, qui est un nouveau tag, donc une étape supplémentaire dans le processus d'analyse, et ainsi de suite. Au final, la boucle serait sans fin.

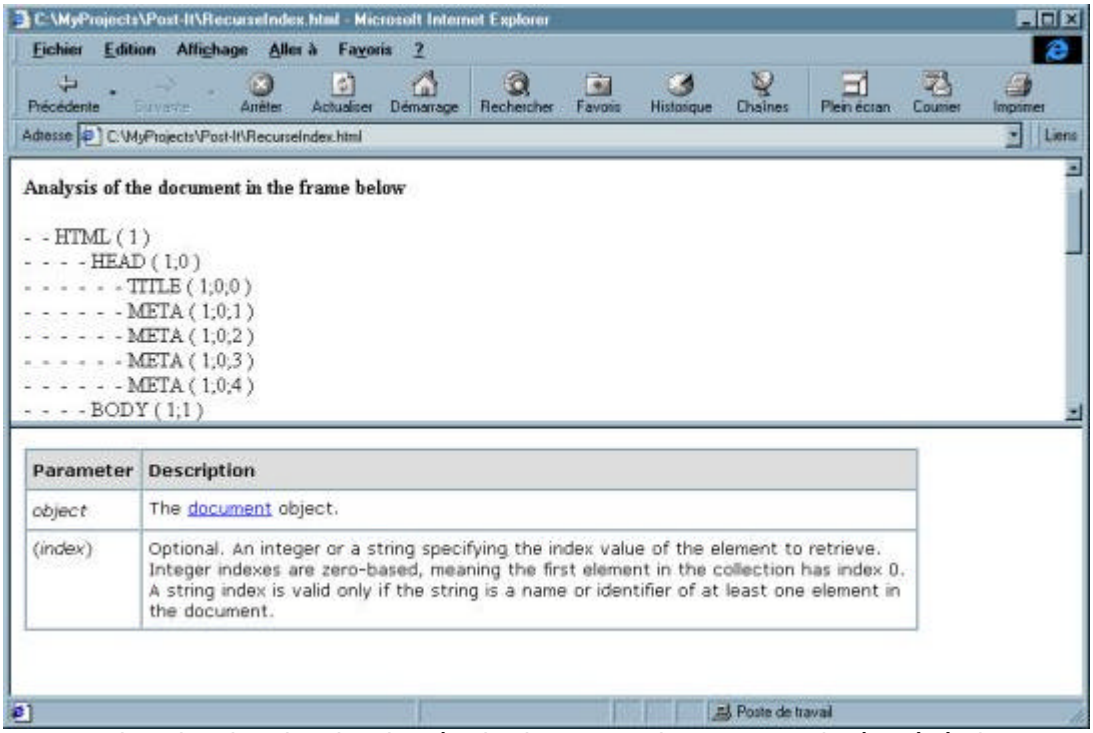
document.all(0) et pas **document.all(1)** ? C'est la question que l'on peut se poser. Il se trouve en l'occurrence que le tag <HTML> de la page analysée n'est pas le premier tag du code source. On trouve le tag <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">, un tag facultatif indiquant la grammaire HTML utilisée (cf norme SGML). Or il se trouve que l'on part du tag père de tous les tags. Analyser la hiérarchie à partir du tag <!DOCTYPE> n'analyse pas la page web car ce tag n'a aucun fils. Le tag <HTML> est frère du tag <!DOCTYPE>.

Solution : Il nous faut donc être capable dans un premier temps de lister tous les tags qui ont pour père le document lui-même, et lancer un processus récursif pour chacun d'eux.

Une autre particularité du web de nos jours est qu'il est possible pour une page web d'afficher du contenu sans qu'il n'y ait le moindre tag <HTML> ou <BODY>. De plus, il est également possible de concaténer plusieurs mini-pages HTML, c'est-à-dire plusieurs tags <HTML> avec pour chacun un contenu web.

Concrètement, le document étant lui-même en quelque sorte un élément de la hiérarchie, il est le seul à ne pas avoir de parent, c'est-à-dire que la propriété **.parentElement** renvoie **null** lorsqu'elle est appliquée sur le document. Le document n'est autre que le premier tag <HTML> rencontré. Partant de l'objet document, nous listons tous ces fils à l'aide de document.all(i). Mais comme nous allons par ce biais à la fois obtenir les fils directs, et les fils indirects, il ne faut lancer un processus d'analyse que lorsque cet élément est fils direct du document, c'est-à-dire lorsque **document.all(i).parentElement==null**.

Lorsqu'une page web comporte plusieurs tags <HTML>, le premier constitue le document, et les autres tags HTML sont des fils.



Obtention des chemins d'accès de chaque tag d'une page web très générale

Le code Javascript pour obtenir ce qui est ci-dessus est :

```
function findAllElements()
{
	var framecontent = window.parent.frames["_2"];

	for (j=0; j<framecontent.document.all.length; j++)
	{
		if (framecontent.document.all(j).parentElement==null)
			findElements(framecontent.document.all(j),j,j+";");
	}
}

function findElements(e, nDepth, szPath)
{
	var i,obj,outputString;

    obj = e;

	// create a depth
	outputString="";
	for (i=0;i<nDepth;i++)
		outputString += " -  - ";

	// remove trailing ';' from szPath
	szPathTemp = szPath;
	if (szPath.charAt(szPath.length-1)==";")
		szPathTemp = szPath.substring(0,szPath.length-1);

	// output the string
	outputString += obj.tagName+" ( "+szPathTemp+" )"+"<br>";

	document.all.myspan.insertAdjacentHTML("BeforeEnd",outputString);


	// go on with children
```

```
        //
        for (i=0; i<obj.children.length; i++)
        {
                if (obj.children[i].id!="myspan")
                        findElements ( obj.children[i],nDepth+1, szPath+i+";" );
        }

}
```

Un détail technique, on supprime le point-virgule final de façon à obtenir un chemin d'accès facile à analyser. De sorte qu'on peut modéliser un chemin d'accès le plus général par :

$$PATH = INDEX\ (\ ';'\ INDEX)^*$$

Où INDEX est un indice réel dans le D.O.M., raison pour laquelle dans notre exemple le PATH du tag <HTML> est 1.

2 - Accès à un élément connaissant son chemin d'accès

La finalité du chemin d'accès est de l'extraire et de le sauver dans une base de données, pour pouvoir ensuite utiliser l'élément référencé.

Ici encore, l'interface de programmation du D.O.M. est parfaitement adaptée.

Par exemple, pour changer la couleur de fond du tag <BODY>, sachant que le chemin d'accès est connu et est : **1;1**, il suffit de trouver l'objet puis de lui appliquer **obj.bgcolor="#FF0000";**

```
function updateElement(szPath,doc)
{
        if (szPath==null)
                return;

        var szPathTemp = szPath;

        var obj = doc;
        var objIndex;
        var i = 0;
        var j = 0;
        var bError = 0;

        // parse first index (dedicated implementation because of .all behavior versus .chi
        j = szPathTemp.indexOf(";",i);
        if (j==-1)
                j=szPathTemp.length;

        s = szPathTemp.substring(0,j);
        objIndex = Number( s.valueOf() );
        i = j;

        obj = obj.all( objIndex );

        // parse the remainder of the path
        //
```

```
        if (j<szPathTemp.length)
        {
                szPathTemp = szPathTemp.substring(i+1,szPathTemp.length);

                // parse and get all indexes
                while ( szPathTemp.indexOf(";",i)>-1 )
                {
                        j = szPathTemp.indexOf(";",i);
                        s = szPathTemp.substring(0,j);
                        objIndex = Number( s.valueOf() );
                        i = j;

                        // get children
                        if (obj!=null)
                                obj = obj.children( objIndex );
                        else
                                bError = 1;

                        // next element in the path
                        szPathTemp = szPathTemp.substring(i+1,szPathTemp.length);

                }
                // last index
                objIndex = Number( szPathTemp.valueOf() );
                if (obj!=null)
                        obj = obj.children( objIndex );
                else
                        bError = 1;
        }

        if (bError>0 || obj==null)
        {
                alert("Wrong path");
                return;
        }

        // process something with the element
        if (obj!=null)
        {
                alert(obj.tagName);
                obj.bgColor="#FF0000";
        }

}
```

Et pour utiliser cette fonction, il suffit de l'appeler par :

```
updateElement("1;1", framecontent.document);
```

(l'objet framecontent est window.parent.frames["_2"]; et permet d'obtenir l'accès au cadre inférieur).

3 - Collecte intéractive du chemin d'accès d'un élément

Ce qui a été vu en 1- et en 2- permet à la fois d'analyser de façon automatique une page web et d'accéder après coup à un élément, mais cela ne préjuge en rien de méthodes d'extraction de chemins d'accès plus intéractives.

A quoi cela sert ? il suffit d'imaginer que l'on sélectionne à la souris un bloc de contenu d'une page web, que l'on clique ensuite sur un bouton et que la partie sélectionnée se trouve copiée, agrégée, bref capturée et disposée ailleurs, dans une autre page web ou une autre application. Oui, il s'agit bien d'un outil d'aggrégation

de contenu. C'est unique. Et les perspectives sont phénoménales car cela ouvre la voie à plusieurs choses :

- pour le backoffice, cela permet l'émergence d'outils d'aggrégation de contenu, capables d'extirper pour votre compte des blocs de news préalablement sélectionnés une fois pour toutes. Toute l'intelligence ici est la capacité à appréhender les nouvelles news, et les aggréger sur un site tierce, rendant de fait inutile d'aller soi-même sur un ou plusieurs sites pour y cueillir les news.
- pour l'utilisateur, être capable d'aggréger du contenu permet de construire son propre web. On parle toujours du web comme d'une interconnection de réseaux. C'est vrai, mais cela veut aussi dire que cela ne nous appartient pas. Le web, c'est les autres. Mais en étant capable d'extirper du contenu en provenance de n'importe quel site web, ce qui est créé, c'est votre web. Pas moins.

Le D.O.M. nous fournit une fois de plus l'identifiant de l'élément sélectionné à la souris. Sans cette fonctionnalité nous ne pourrions rien faire sans développer de toutes pièces un nouveau navigateur web !

L'API minimale est :

- **document.elementFromPoint(x,y)** : qui retourne l'élément sous la position courante (x,y) de la souris
- **document.activeElement**
- **document.selection** : qui retourne la sélection courante à la souris. A partir de cette sélection, on peut appeler une méthode **createTextRange()**, qui à son tour crée un objet TextRange. A partir de l'objet TextRange, il est possible notamment d'obtenir l'élément l'englobant par un simple **.parentElement()**.

La fonction Javascript suivante permet de remonter à l'écran la sélection courante, c'est-à-dire le premier tag englobant la sélection souris courante. Comme pour les scripts précedents, l'accès à l'objet document est basé sur l'accès au cadre inférieur d'une page comportant deux cadres.

```
function showParent()
{
	var framecontent = window.parent.frames["_2"];
	var sel = framecontent.document.selection;
	var rng = sel.createRange();
	var obj = rng.parentElement();

	if (obj!=null)
		alert(obj.tagName);
	else
		alert("No object selected");
}
```

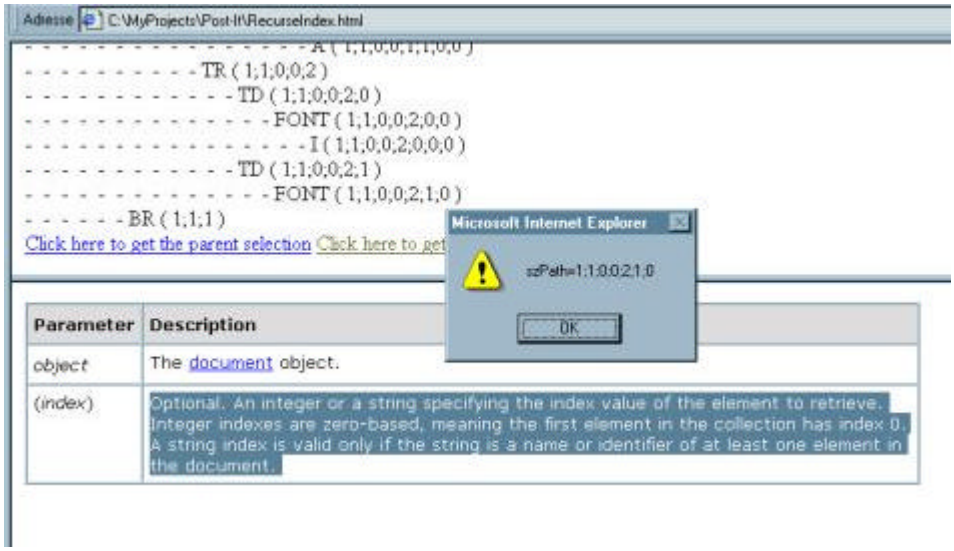
Pour obtenir le chemin d'accès, il faut construire le cheminement qui mène de la racine à cet élément. Mais le D.O.M. ne nous fournit pas directement ces informations. Il va falloir les construire de proche en proche. L'interface de programmation ne fournit pas l'inverse de **document.all(i)**, c'est-à-dire l'indice d'un élément lorsqu'on connaît celui-ci.

Nous allons devoir récupérer l'élément père puis parcourir tous les fils et chercher l'élément que nous avons déjà en main. L'utilisation d'un compteur permettra d'obtenir l'information cherchée. De proche en proche, nous allons être capable de construire le chemin vers la racine.

Mis à part écrire cette fonction qui, semble-t-il met en jeu une certaine forme de récurrence, il faut trouver un moyen de particulariser l'élément courant que nous connaissons, de façon à ce que lorsque nous parcourons l'ensemble des éléments nous puissions à coup sûr mettre la main dessus.

A cet effet, il semble assez logique d'utiliser un attribut. Pour éviter de détruire la logique initiale de la page web, nous utilisons un nouvel attribut, **identifier**, puis nous lui affectons une valeur très particulière et peu courante, comme **_Lookup456**. L'interface de programmation du D.O.M. nous permet d'accéder en lecture et en écriture à cet attribut à l'aide de getAttribute(name) et setAttribute(name, value). Nous procédons au parcours des éléments. Une fois l'indice compteur obtenu, nous redonnons à l'élément sa valeur d'attribut initial,sachant que dans la réalité cet attribut est custom et a donc peu de chances d'être utilisé dans une page web sur un site.

En sélectionnant une zone, et en déclenchant le script, nous obtenons :



Récupération intéractive à la souris d'un identifiant de bloc de page Web

Voici le code correspondant :

```
// find the ID (in the element hierarchy) of the current selection
function findIndex()
{
	var framecontent = window.parent.frames["_2"];
	var sel = framecontent.document.selection;
	var rng = sel.createRange();
	var obj = rng.parentElement();

	if (obj==null)
	{
		alert("No object selected");
		return;
	}

	//alert(obj.tagName);
	szId = "identifier";
	szNewName = "_Lookup456";

	var szPath="";
	while (obj!=null)
	{
		szOldName = obj.getAttribute(szId);
		obj.setAttribute(szId,szNewName);

		id = lookupElement(framecontent.document,obj,szNewName);

		obj.setAttribute(szId,szOldName);

		//alert("id="+id);

		// construct the path incrementally, from right to left (from bottom to ro
		szPath = id+szPath; // the empty string is used to convert the int to a st

		obj = obj.parentElement;

		if (obj!=null)
			szPath = ";" + szPath;
	}

	alert ("szPath="+szPath);
}

// find a given element in a hierarchy
```

```
function lookupElement(doc, element, szName)
{
	var i = 0;

	if (doc==null || element==null)
		return -1;

	var parent = element.parentElement;

	if (parent==null)
	{
		// perform child search using the document object
		while ( (i<doc.all.length) && (doc.all(i).getAttribute(szId) != szName) )
		{
			i++;
		}

		if (i==document.all.length) // not found, return -1
			i = -1;
	}
	else
	{
		// perform child search using the parent element
		while ( (i<parent.children.length) && (parent.children(i).getAttribute(szI
		{
			i++;
		}

		if (i==parent.children.length) // not found, return -1
			i = -1;
	}

	return i;
}
```

4 - Extension à un groupe d'éléments

Des algorithmes peuvent être créés pour grouper des éléments par blocs. On entend par bloc toute zone compacte d'informations dans une page web.

Dans notre cas de figure, si nous sélectionnons le texte d'une cellule d'un tableau, tout se passe comme si nous avions sélectionné le tag <FONT> qui "encapsule" le texte. Mais au besoin nous pouvons décider de capturer le tableau entier. Comment faire ? Simple, il suffit de remonter d'un cran, pour tomber sur le tag <TD>, puis remonter sur le tag <TR> puis sur le tag <TBODY> (n'apparaît jamais dans le code source HTML) puis sur le tag <TABLE>.

5 - Extraction des données

Ce paragraphe indique la méthode pour extraire le contenu d'une sélection dans une page web. Ceci est mis en jeu dans le cas d'une application d'aggrégation de contenu.

Là encore, l'interface de programmation du D.O.M. nous rend tous les services possibles et imaginables.

La sélection doit être transformée en objet TextRange, et les méthodes de TextRange à utiliser sont les suivantes :

- **TextRange.text** : extraie le texte au format brut
- **TextRange.htmlText** : extraie le texte au format HTML
- **TextRange.expand(mode)** : agrandit la sélection. Modes : **word**, **sentence**, **textedit**
- **TextRange.pasteHTML(text)** : recopie la sélection ailleurs

Ainsi que des méthodes pour changer le début et la fin de la sélection.

Bref, tout est possible. Le code suivant sélectionne le mot lorsque vous cliquez sur celui-ci :

```
<HTML><HEAD>
<TITLE>moveToPoint Example</TITLE>
<script>
  function selectMe()
  {
    var r=document.body.createTextRange();
    r.moveToPoint(window.event.x, window.event.y);
    r.expand("word");
    r.select();
  }
</script>
</HEAD>

<BODY>

<H1 id="myH1" onclick="selectMe()">Click on a word and it will highlight</H1>

</BODY>
</HTML>
```

Il faut savoir par ailleurs qu'au titre de **TextRange** on peut assimiler tout contenu de tag, ce qui permet dans la pratique de combiner ce que l'on sait faire pour un simple texte pour capturer des blocs d'information complets.

Pour rappel, chaque tag HTML est composé d'un tag d'ouverture, d'un certain nombre d'attributs, éventuellement 0, d'un contenu, et éventuellement d'un tag de fermeture :

<htmltag [attribute1="value1"] [attribute2="value2"] …>

[content]

[</htmltag>]

6 - Sauvegarde sur disque.

Une fois les données extraites, il suffit de créer une requête sur un serveur et de les y transférer. Ceci se fait très bien en forçant une requête GET (construire une URL et passer le texte en paramètre) ou POST (simuler l'envoi des données d'un formulaire invisible) toujours par du script.

Mais il est aussi possible à ce niveau, et suivant l'application, de sauvegarder sur disque et en local les données qui nous avons en main. Pour ce faire, sous Internet Explorer, nous pouvons faire appel à un objet ActiveX FileSystem :

```
var fs = new ActiveXObject("Scripting.FileSystemObject");
var a = fs.CreateTextFile("c:\testfile.txt", true);
a.WriteLine("This is a test.");
a.Close();
```

Pour que ce script s'exécute, il faut que l'exécution dite d'Active Scripting soit autorisée. Active Scripting n'est autre que l'autorisation de l'exécution de composants ActiveX via du script Jscript ou VBScript. C'est une option du navigateur web. Cette solution ne fonctionne qu'avec Internet Explorer. Internet Explorer 4+ installe le composant ActiveX FileSystem. Pour le vérifier, il suffit d'aller dans la base de registres et de chercher la ligne HKEY_CLASSES_ROOT \ Scripting.FileSystemObject.