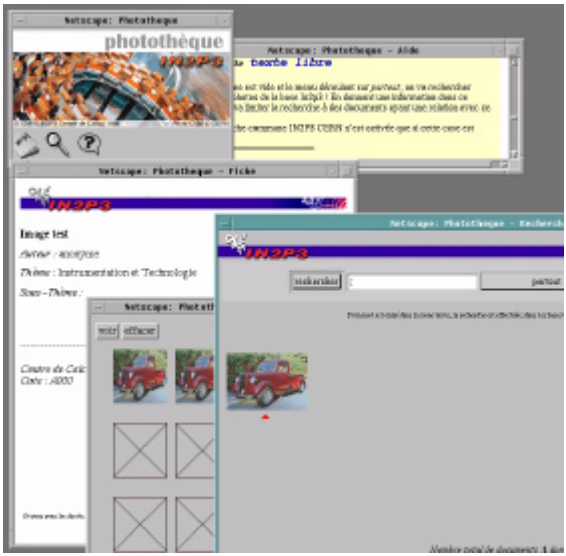
L'environnement multi fenêtré



Une application s'exécutant dans le cadre d'un browser Web démarre généralement depuis un lien hypertexte ou un signet stocké dans le *bookmark*. La première opération que devra réaliser cette application sera donc de se créer sa propre fenêtre. Celle ci sera généralement la fenêtre principale et aura comme durée de vie toute la durée de vie de l'application. La création d'un nouveau *browser* entièrement dédié présente de nombreux avantage. Le concepteur de l'application a ainsi la maîtrise complète de l'environnement qu'il met en œuvre ; il peut par exemple gérer la navigation à sa guise : le bouton *back* par exemple, peut être sinon catastrophique, tout au moins très gênant dans certains enchaînements de formulaires.
Cette fenêtre principale, ne servant plus à *surfer* sur le réseau, sera donc construite avec un environnement complètement géré par le programmeur donc adaptée aux strictes besoins de l'application.
La fermeture de cette fenêtre principale terminera l'application et on prendra soins de programmer la clôture de l'ensembles des fenêtres qui auront pu être créées. En effet il ne serait pas normal de considérer que la fin de

l'application soit liée à l'action "quitter" du *browser*. Le *browser* a servi à lancer l'application, la prise en charge de tout l'environnement revient alors à l'application qui laissera, lorsqu'elle s'arrêtera l'environnement de travail (le bureau) dans son état initial.

contexte

La notion de fenêtre principale

Cette fenêtre est le premier contact entre l'utilisateur et l'application. L'information présentée devra être suffisante pour que l'utilisateur puisse commencer à travailler, et on lui facilitera l'identification rapide des différentes fonctions mises à sa disposition.

Si on ne gère pas un accès personnalisé, c'est à dire l'enregistrement d'un contexte préféré de l'utilisateur, il faut alors garder à l'esprit que l'application présente une apparence identique pour les utilisateurs qui la connaisse déjà et pour ceux qui la découvre. L'accès à l'aide ou à l'initiation doit donc être facilement repérable par le nouveau venu mais doit être le plus discret possible pour celui qui maîtrise déjà le logiciel. On peut imaginer au moins deux genres distincts pour cette fenêtre :

- La fenêtre principale est le lieu ou s'effectue le travail ; par exemple le traitement de texte Microsoft Word démarre avec un document vide et depuis cette fenêtre principale on va déclencher diverses fenêtres : format des paragraphes, choix de couleurs, aide, etc.
- La fenêtre principale n'est qu'une barre de menu comportant les boutons pour accéder aux divers fonctions : ouverture d'un document, création d'un nouveau document ou aide en ligne pour le traitement de texte FrameMaker.

Le choix de l'une ou l'autre des techniques dépend essentiellement des fonctions apportées par l'application, de l'ergonomie souhaitée et du cahier des charges défini pour l'utilisateur final.

fonctionnalité

La fenêtre principale aura d'un point de vue fonctionnel plusieurs rôles:

- Stocker les scripts servant à la création des fenêtres auxiliaires et par la même en conserver leurs références ; ceci permettra leurs fermetures lors de la fin de l'application.
- Conserver un maximum de scripts : si les scripts sont stockés dans

la fenêtre principale il vont pouvoir être réutilisés par toutes les fenêtres de l'application. Ceci évitera entre autre leur duplication.

- Stocker des variables globales qui seront ainsi connue par l'ensemble des scripts de l'application

Si la fenêtre principale est le lieu de dépôt des fonctions, des références à certains objets ou des variables, il importe que cette fenêtre puisse être référencée par sa descendance. Rappelons qu'une fenêtre est connue depuis le document HTML par un nom, et qu'elle est connue par l'interpréteur JavaScript, donc par l'ensemble des fonctions JavasScript, par une référence. Le nom au sens HTML est utilisé en relation avec l'attribut *target* des différentes balises (*forms*, *area*,...). La référence, qui n'est autre que le nom de la variable utilisée dans l'instruction *window.open* va être utilisé par les programmes JavaScripts.

le fichier de lancement de l'application exécute essentiellement l'instruction de création de la fenêtre et repositionne le browser principale sur l'URL ou il se trouvait. L'application démarre dans une fenêtre dénuée du décor habituel du *browser* (pas de boutons, ni de *status,* ni d'affichage de l'URL)

```
<html><head><title>l'Appli</title></head>

<script>
    // creation de la fenetre principale de l'application
    fenetrePrincipale =  window.open
                                    ("menu.html",
                                     "laFenetrePrincipale",
                                     "width=322,height=200, scrolling=no, toolbar=no,
menubar=no, location=no, directories=no,status=no");

    // repositionnement a l'URL
    history.back();
</script>

</html>
```

Exemple de code 11

Dans l'exemple de code11, *fenetrePrincipale* est la référence au sens JavaScript et sera utilisé dans les codes et fonctions des scripts; en revanche *laFenetrePrincipale* est le nom de la fenêtre au sens

HTML et servira essentiellement avec l'attribut *target* des balises <A>, <FORM>, <MAP>, etc.
Afin de simplifier les programmes et pour que le programmeur n'ait à se souvenir que d'un seul nom, nous recommandons fortement d'utiliser le même nom pour le nommage HTML et JavaScript. En aucun cas l'utilisation d'un même nom ne sera source de confusion ou de problèmes.

Le référencement d'objets entre fenêtres

Si plusieurs fenêtres coopèrent entre elles, il va vraisemblablement être nécessaire que chacune d'elle puisse utiliser des objets contenus dans les fenêtres voisines. Par exemple en appuyant sur un bouton d'une fenêtre on va modifier le contenu d'une liste se situant dans une autre fenêtre.

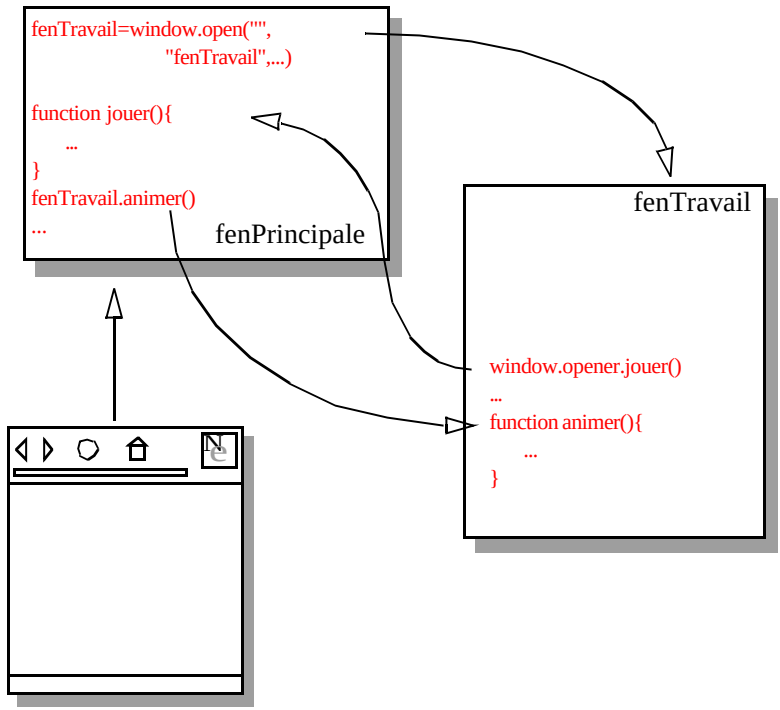


Figure 12 - multi fenêtrage

Dans l'exemple de la figure12, l'application lancée par le browser, vraisemblablement par un script identique à l'exemple de code11, se compose d'une fenêtre principale (*fenPrincipale*) contenant le script *jouer* et d'une fenêtre de travail (*fenTravail*) contenant le script *animer.*

Comportement

Lorsque depuis la fenêtre *fenTravail* on veut exécuter un script situé dans la fenêtre *fenPrincipale* il n'est pas possible de référencer le chemin vers ce script en indiquant le nom *fenPrincipale*. En effet, *fenTravail* est la fenêtre fille de *fenPrincipale* et depuis cette fenêtre

fille on ne connaît pas le nom, la référence, de la fenêtre parent. Pour invoquer un objet situé dans son ascendance, comme son créateur, on utilise le mot *opener*. Le référencement des objets dans un système multi fenêtré peut être comparé à la navigation dans un système hiérarchique de fichier de type Unix. : on ne peut nommer que les répertoires qui son au dessous de sois. Si on veut remonter d'un niveau on utilise la syntaxe .. (*opener* en JavaScript) de deux niveaux ../.. (*opener.opener*) et ainsi de suite. Pour aller dans une branche de l'arborescence de même niveau, on remonte d'un niveau puis on nomme explicitement le niveau à atteindre.

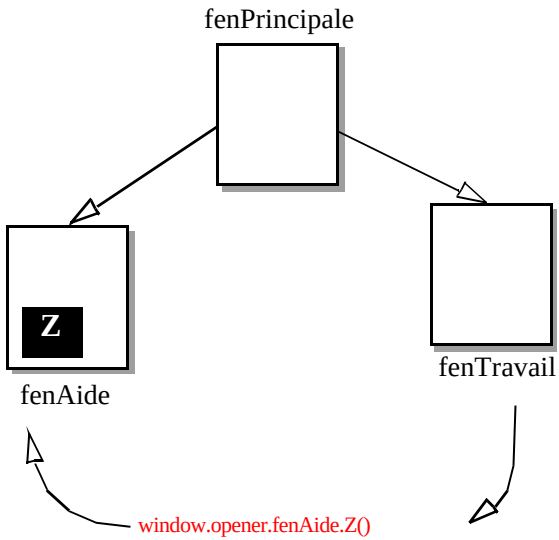


Figure 13 - références inter fenêtre

La figure13 illustre un système où depuis *fenPrincipale* on a ouvert deux fenêtres. Si un script contenu dans le fenêtre *fenTravail* doit accéder un script (Z) contenu dans la fenêtre *fenAide* située hiérarchiquement au même niveau, il utilisera le nommage *window.opener.fenAide.Z*.
window.opener renvoie au niveau parent *fenPrincipal* qui lui à une visibilité sur sa descendance, *fenAide*.
Une référence de type *window.opener.opener.fenSaisie* est tout à fait

légale.

Tester l'existence d'une fenêtre

Avant d'agir ou d'interroger un objet (script, variable, élément d'un formulaire, etc.) contenu dans une fenêtre il est nécessaire de vérifier avant tout l'accessibilité vers cette celle -ci. En effet toute référence à une fenêtre ou à son contenu qui serait inacessible se traduit par un message d'erreur émis par l'interpréteur.

Pour effectuer ce test on va analyser le type (*typeof*) de la variable censée référencer cette fenêtre. Si la fenêtre n'a jamais été créée et la variable jamais utilisée par ailleurs, elle sera de type indéfini (*undefined*) ; en revanche, le type de la variable sera *window* pour peu que la fenêtre ait existé à un moment donné et que la variable n'ait pas été réaffectée entre temps à un autre usage..

Tester le type de la variable n'est donc pas totalement suffisant mais néanmoins nécessaire puisqu'avant de vérifier l'état de la fenêtre on s'assure qu'il s'agit d'un objet fenêtre.

```
if ((typeof(fenAide)=="window"){
    // la fenetre existe (ou a existé...)
    if (fenAide.close == false) {
        // la fenetre existe reellement
    }
        ...
} else {
    // la fenetre n'a (peut etre) jamais ete creee
        ...
}
```

Exemple de code 14

Fermer les fenêtres

Dans une application on prendra soin d'ouvrir toutes les fenêtres depuis la fenêtre principale de l'application et on gardera la référence à ces fenêtres dans des variables globales. Ainsi lors de la fin de cette application on appellera une fonction qui fermera toute les fenêtres qui sont encore ouvertes. Cette fonction peut être positionnée par exemple sur l'événement *onUnLoad* survenant lorsque la fenêtre principale est close ou encore être déclenchée par un clic sur un bouton ou un lien hypertexte. Il est nécessaire, avant d'appeler la métho-

de fermeture d'une fenêtre, de s'assurer que la fenêtre existe ou a existé. En effet appeler la méthode *close* sur une fenêtre qui a été fermée provoque un message d'erreur JavaScript se traduisant par une fenêtre modale d'erreur sur Internet Explorer, et un message dans la console JavaScript de Netscape.

Un fenêtre a pu être ouverte à un moment, donc l'objet window a été créé, puis avoir été ensuite fermée ; cette fermeture laisse l'objet window avec la propriété *closed*

```
function cloreFenetres (); {
    if ((typeof(fenAide)!="undefined")&&(fenAide.closed==false))
        fenAide.close();
    if ((typeof(fenMenu)!="undefined")&&(fenMenu.closed==false))
        fenMenu.close();
    ...
    self.close
}
```

Exemple de code 15

Dans l'exemple de code15, la dernière instruction (self.close) est utile dans le cas ou l'on a prévu de terminer l'application sur un événement différent de l'événement *onUnload*. Si on implémente un bouton "quitter l'application," dans la fenêtre principale, il faut donc aussi fermer celle-ci.

Le nommage si des frames sont présentes

Pour un browser une frame ou une fenêtre

Les fichiers de test

```
<!-- fichier index.html -->
<html>
<head><title>lien vers l'application</title></head>
<body link=red vlink=red alink=black>
<h3><a href=startAppli.html>Lancer l'application</a></h3>
</body>
</html>

<!-- Fichier startAppli.html -->
<html>
```

```
<script>
    // creation de la fenetre principale de l'application
    fenetrePrincipale =  window.open
                                    ("appli.html",
                                     "laFenetrePrincipale",
                                     "width=322,height=200, scrolling=no, toolbar=no,
menubar=no, location=no, directories=no,status=no");
    // repositionnement a l'URL d'origine
    history.back();
</script>
</html>

<!-- fichier appli.html -->
<html>
    <head><title>l'Appli</title>
        <script>

            function tester() {
                alert ('Hello...');
            }

            function creerFenTravail() {
                fenTravail = window.open
                                    ("travail.html",
                                     "fenTravail",
                                     "width=500,height=350, scrolling=no, toolbar=no,
menubar=no, location=no, directories=no,status=no");
            }

            function creerFenAide() {
                fenAide = window.open
                                ("aide.html",
                                 "fenAide",
                                 "width=400,height=100, scrolling=no, toolbar=no, menubar=no,
location=no, directories=no,status=no");
            }

            function cloreFenetres() {
                if((typeof(fenTravail)!="undefined")
                    &&(fenTravail.closed==false))fenTravail.close();
                if((typeof(fenAide)!="undefined")
                    &&(fenAide.closed==false))fenAide.close();
                self.close();
            }
```

```
        </script>
    </head>
    <body onUnload=cloreFenetres()>
    <h3>Fen&ecirc;tre principale</h3>
    <ul>
        <li><a href=javascript:void(0) onClick=creerFenTravail();> ouvrir une fen&ecirc;tre de
travail</a>
        <li><a href=javascript:void(0) onClick=cloreFenetres();> quitter l'application</a>
    </ul>
    </body>
</html>
```

<!-- Fichier travail.html -->

```
<html>
<head><title>Fenetre de travail</title></head>
<body>
<h3>Fen&ecirc;tre de travail</h3>
<ul>
<li>
<a href=javascript:void(0); onClick=window.opener.tester()> Appeler un script de la fen&ecirc;tre
principale</a>
<li>
<a href=javascript:void(0);onClick=window.opener.creerFenAide()> Appeler l'aide</a>
</ul>
</body>
</html>
```