Les applets

La balise APPLET

les applets sont chargées à partir d'une page HTML contenant la balise :

```
<APPLET ...>
...
</APPLET>
```

Syntaxe générale

```
'<' 'APPLET'
      ['CODEBASE' '=' repURL]
      'CODE' '=' fichier_applet.class
      ['ARCHIVE' '=' liste_de_fichiers_archive]
      ['ALT' '=' texte_de_remplacement]
      ['NAME' '=' nom_de_l_applet]
      'WIDTH' '=' pixels 'HEIGHT' '=' pixels
      ['ALIGN' '=' alignement]
      ['VSPACE' '=' pixels] ['HSPACE' '=' pixels]
'>'

['<' 'PARAM' 'NAME' '=' param1 'VALUE' '=' value '>']
['<' 'PARAM' 'NAME' '=' param2 'VALUE' '=' value '>']
...
[HTML_de_remplacement]
'</APPLET>'
```

Signification des champs

'CODEBASE' '=' repURL (optionnel).
Cet attribut optionnel indique l'URL du répertoire où se trouve le byte code. Lorsque que cet attribut n'est pas mentionné, c'est l'URL du répertoire de la page HTML courante.

'CODE' '=' fichier_applet.class
indique le nom du fichier où se trouve le byte code. Son nom est relatif à CODEBASE.

'ALT' '=' texte_de_remplacement (optionnel).
indique un texte de remplacement à afficher pour les browsers comprenant la balise applet mais ne pouvant exécuter Java.

'NAME' '=' nom_de_l_applet (optionnel).
indique le nom de l'applet. Utilisé pour faire communiquer 2 applets d'une même page HTML.

'WIDTH' '=' pixels 'HEIGHT' '=' pixels
indique la taille de la zone de dessin initiale dans laquelle sera exécutée l'applet dans la page HTML.

Signification des champs (suite)

'ALIGN' '=' alignment (optionnel).
indique l'alignement de la zone de dessin de l'applet dans la page HTML. Les valeurs possibles sont celles de l'option ALIGN de la balise IMG (left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom).

'VSPACE' '=' pixels 'HSPACE' '=' pixels
(optionnel).
indique le nombre de pixels à laisser dessus et dessous (VSPACE), à gauche et à droite (HSPACE) pour la zone dessin de l'applet dans la page HTML.

'<' 'PARAM' 'NAME' '=' param_i 'VALUE' '=' value '>'
. . .
(optionnels). Ce sont les paramètres que passent la page HTML à l'applet. Celle ci les récupère grâce à la méthode getParameter().

HTML_de_remplacement
est affiché par les browsers ne comprenant pas la balise <APPLET>.

Exemples

```
<applet code="MyApplet.class"
width=100 height=140>
</applet>
```

indique au browser d'afficher l'applet dans une zone de 100 par 140 pixels. Le byte code se trouve dans le même répertoire que la page HTML dans le fichier `MyApplet.class`.

exemple 2:

```
<applet
codebase="http://www.javasoft.com/applets/applets/NervousText"
code="NervousText.class" width=400 height=75
align=center >
<param name="text" value="This is the Applet Viewer.">
<blockquote>
<hr>If you were using a Java-enabled browser, you would see dancing text instead of this paragraph.<hr>
</blockquote>
</applet>
```

indique au browser de charger le byte code `http://www.javasoft.com/applets/applets/NervousText/NervousText.class`

La taille initiale de la zone d'affichage est 400 par 75 pixels. Cette zone est alignée au centre de la fenêtre du browser.

Le code de l'applet s'attend à recevoir un argument de nom `text`. La valeur passée à cet argument est : `This is the Applet Viewer.`

Un browser ne pouvant pas exécuter Java mais comprenant la balise APPLET affichera le texte HTML entre `<blockquote>` et `</blockquote>`

La balise ARCHIVE

L'obtention des ressources d'une applet (une nouvelle classe à charger, une image, un fichier son, ...) est faite à l'exécution, au moment où l'applet en a besoin (évaluation paresseuse). Pour chaque ressource de l'applet, une connexion HTTP est nécessaire et ceci est préjudiciable.

À partir de la version 1.1, on peut regrouper plusieurs ressources dans un fichier compressé qui est chargé par une seule connexion HTTP. Un tel fichier est un archive compressée (par le même algorithme que les fichiers `.zip`) : un `.jar`.

L'outil `jar` de construction d'archive compressée est donné avec le jdk :

```
% jar cf Neko.jar Neko.class images/*.gif
```

Le fichier HTML indique ce fichier archive ainsi que la classe de l'applet (attribut `CODE`).

```
<APPLET ARCHIVE="Neko.jar" CODE="Neko.class"
WIDTH=400 HEIGHT=150>
</APPLET>
```

Plusieurs fichiers archive peuvent être référencés par l'attribut `ARCHIVE`. Ils sont alors séparés par des virgules.
Si un fichier recherché n'est pas dans ces archives, ce fichier est recherché comme habituellement (compatibilité ascendante) par une nouvelle connexion HTTP.

Sécurité et applets

Charger du code "inconnu" est un risque d'insécurité. De ce fait, les browsers lisant Java ou les applet viewers ont implantés une sous classe de la classe `java.lang.SecurityManager` de sorte à interdire à une applet :

- de lire ou d'écrire dans des fichiers locaux
- de détruire des fichiers locaux (soit en se servant de `File.delete()` soit en lançant une commande système `rm` ou `del`)
- de renommer des fichiers locaux (soit en se servant de `File.renameTo()` soit en lançant une commande système `mv` ou `rename`)
- de créer un répertoire sur le système local (soit en se servant de `File.mkdir()` soit ou `File.mkdirs()` en lançant une commande système `mkdir`)
- lister le contenu d'un répertoire local.
- vérifier l'existence, la taille, le type, la date de dernière modification d'un fichier local
- créer une connexion vers un ordinateur autre que celui d'où vient l'applet
- écouter, accepter des connexions d'un port du système local.
- créer une fenêtre "toplevel" sans indiquer qu'elle provient d'une applet.
- obtenir le nom, le répertoire de connexion de l'utilisateur

Sécurité et applets (suite)

- lancer un programme local (par `Runtime.exec()`, `Runtime.exit()`, `System.exit()`
- charger des librairies locales

Les seuls paquetages locaux qu'une applet peut utiliser sont :
`java.applet`, `java.awt`, `java.awt.image`, `java.awt.peer`, `java.io`, `java.lang`, `java.net`, `java.util`.
Certaines de ces "interdictions" peuvent être levées par certains browsers.

De même si ces applets proviennent de certaines parties du réseau (Intranet, firewall, ...) : cf. `hotjava`.

Plus précisément, toute application Java implante sa propre sécurité à l'aide d'un objet chargé une et une seule fois au début du lancement de l'application : un objet d'une sous classe de la classe `SecurityManager`. C'est le cas des browsers. Cet objet est unique à l'intérieur d'une application Java, ne peut pas être remplacé et pour les browsers, les applets ne peuvent pas le référencer.

applet = ?

Une applet est une classe chargée par le browser Web. En local l'interpréteur Java instancie un objet de cette classe et lance la méthode `init()` sur cet objet. Parfois c'est cet objet qui est appelé une applet.
Cette classe est obligatoirement une classe dérivée `public` de la classe `Applet`. On doit donc écrire :

```
public class MonApplet extends java.applet.Applet {
...
}
```

Une applet est aussi un conteneur d'objets graphiques à l'intérieur du browser. On obtient cela grâce à l'héritage :
`Object` -> `Component` -> `Container` -> `Panel` -> `Applet`.

Méthodes fondamentales d'une applet

Ce sont :

```
public void init() { ... }
```

lancée au moment où l'applet est chargée par le browser

```
public void start() { ... }
```

lancée après `init()` et aussi lorsque l'applet est "visitée" i.e. lorsque le browser revient sur la page HTML contenant l'applet (par appui sur le bouton "back" du navigateur ou bien après désiconification du navigateur)

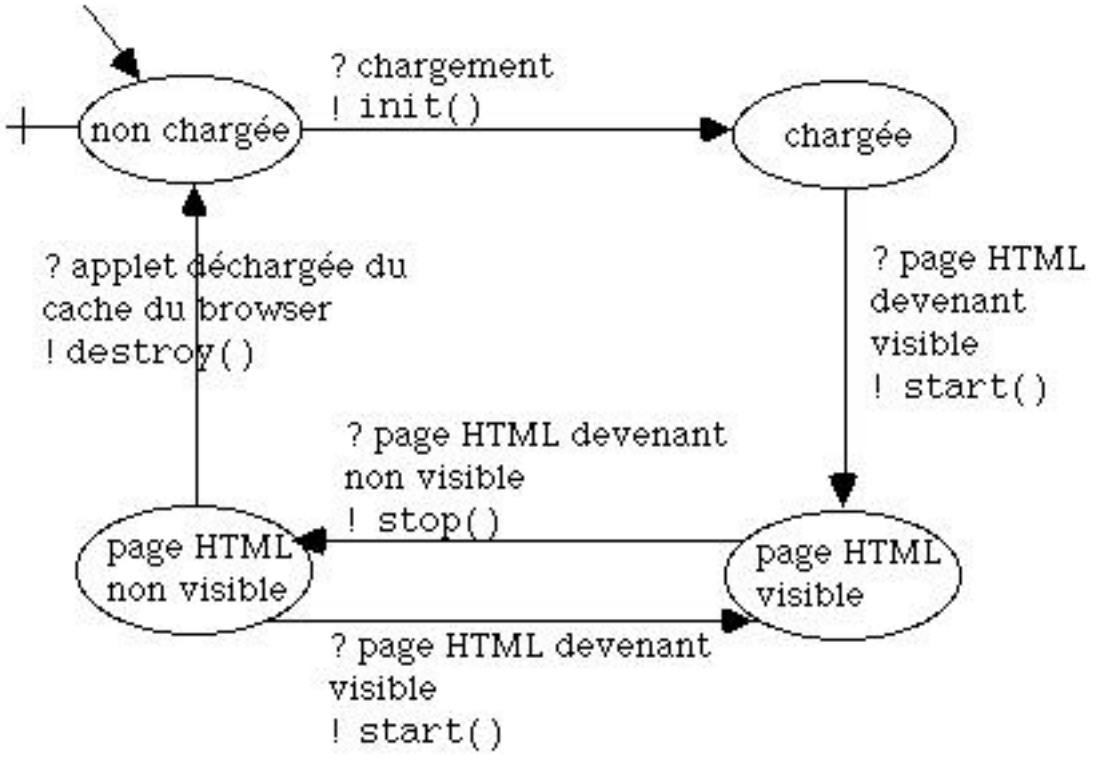
```
public void stop() { ... }
```

lancée lorsque l'applet n'est plus à l'écran : par exemple quand le browser quitte la page HTML contenant l'applet (ou lorsque le browser est iconifié).

```
public void destroy() { ... }
```

lancée lorsque le browser "décharge" l'applet (ou que l'exécution du browser se termine).

Automate des états d'une applet



Méthodes fondamentales d'une applet (suite)

En général on met :

dans `init()` :

création d'objets, chargement d'images, de polices, initialisation de champ globaux à la classe

dans `start()` :

lancement de tache (thread)

dans `stop()` :

l'arrêt de thread.

dans `destroy()` :

libération de ressources de l'applet (images, polices, ...)

en résumé :

```
init()
    start()
    stop()
destroy()
```

Exemple

```
import java.applet.*;
import java.awt.*;

public class monApplet extends Applet {
    static final String message = "Hello World";
    private Font font;

    public void init() {
        font = new Font("Helvetica", Font.BOLD, 48);
    }

    public void paint(Graphics g) {
        // Un ovale plein rose
        g.setColor(Color.pink);
        g.fillOval(10, 10, 330, 100);

        // Un contour ovale rouge épais. Le contour
// java sont seulement d'épaisseur un pixel. Il
// faut donc en dessiner plusieurs.
        g.setColor(Color.red);
        g.drawOval(10,10, 330, 100);
        g.drawOval(9, 9, 332, 102);
        g.drawOval(8, 8, 334, 104);
        g.drawOval(7, 7, 336, 106);

        // le texte
        g.setColor(Color.black);
        g.setFont(font);
        g.drawString(message, 40, 75);
    }
}
```

Exemple (suite)

avec le fichier HTML :

fichier second.html

```
<html>
<head><title>Seconde applet</title></head>

<body>
<h1>Bienvenue sur cette nouvelle page</h1>
Ex&eacute;cution de cette seconde applet :<P>

<applet code="monApplet.class" WIDTH=400
HEIGHT=200>
Si vous voyez ce texte c'est que votre browser ne
supporte pas la balise applet
</applet>
</body>
</html>
```

donne l'affichage dans netscape :

Exemple (fin)

Un outil de développement permet de ne visionner que la partie `<APPLET ...> </APPLET>` d'un fichier html. Ce mini-browser plus rapide que netscape est toujours livré avec les versions du JDK correspondantes (c'est d'ailleurs une application indépendante). Cet outil est `appletviewer` et il est encouragé de l'utiliser.

Le fichier html ci dessus exécuté par `appletviewer` donne :

```
% appletviewer second.html
```

Récupérer les paramêtres du .html dans l'applet

On utilise la méthode `getParameter()` de la classe `Applet`. Par exemple soit le fichier html :

```
<html>
<head><title>Hello World Parametre
</title></head>

<body>

<applet code="HwParam.class" width=400
height=200>
<param name="msg" value="Bonjour a tous">
</applet>
</body>
</html>
```

On veut, dans l'applet, récupérer la valeur du message à afficher.

Récupérer les paramêtres du .html dans l'applet (suite)

Voici l'applet :

```
iimport java.applet.*;
import java.awt.*;

public class HwParam extends Applet {
    private String message;
    private Font font;

    public void init() {
        font = new Font("Helvetica", Font.BOLD, 48);
        message= getParameter("msg");
    }

    public void paint(Graphics g) {
        // Un ovale plein rose
        g.setColor(Color.pink);
        g.fillOval(10, 10, 330, 100);

        // Un contour ovale rouge épais.
        g.setColor(Color.red);
        g.drawOval(10,10, 330, 100);
        g.drawOval(9, 9, 332, 102);
        g.drawOval(8, 8, 334, 104);
        g.drawOval(7, 7, 336, 106);

        // le texte
        g.setColor(Color.black);
        g.setFont(font);
        g.drawString(message, 40, 75);
    }
}
```

Exemples (suite)

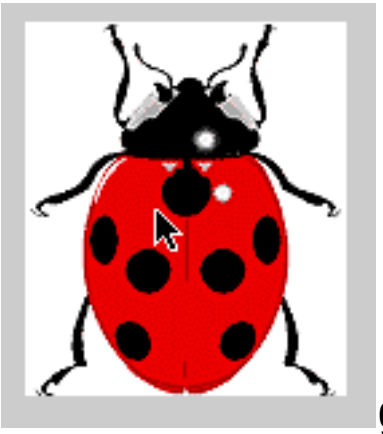
les images

Java lit les formats `.gif` et les affichent sans problème. Si le fichier `ladybug.gif` se trouve dans le répertoire `images`, sous-répertoire du répertoire du fichier de l'applet, le code :

```
import java.awt.Graphics;
import java.awt.Image;

public class LadyBug extends java.applet.Applet
{
Image bugimg;
  public void init() {
    bugimg = getImage(getCodeBase(),
"images/ladybug.gif");
}
public void paint(Graphics g) {
    g.drawImage(bugimg,10,10,this);
}
}
```

donne :



`getCodeBase()` retourne l'URL du répertoire de l'applet. `getImage()` supprime en fait la concaténation avec son second argument.

Les images dans les applications Java

Il y a 2 méthodes `getImage()` en Java. L'une appartient à la classe `Applet` et on l'a utilisé ci dessus. L'autre est de la classe `Toolkit` et c'est la seule utilisable dans une application Java autonome. Une application Java similable à l'applet ci dessus est :

```
import java.awt.*;

public class LadyBug extends Frame {
Image bugimg;

  public static void main(String args[ ]) {
      Toolkit tk = Toolkit.getDefaultToolkit();
      Frame fr = new LadyBug(tk, "Coccinelle");
      fr.setSize(200, 200);
      fr.show();
    }

public LadyBug (Toolkit tk, String st) {
        super(st);
        bugimg = tk.getImage("ladybug.gif");
    }

public void paint(Graphics g) {
            g.drawImage(bugimg,10,10,this);
    }
}
```

Passer d'une application à une applet

en général c'est un code "simplifié". En effet, dans une applet :
- la taille de l'applet a été fixée par le fichier HTML (donc plus besoin de `setSize()`)
- l'affichage est fait par le browser (donc plus besoin de `setVisible(true)`)
- la création du premier objet (l'applet) est faite par le browser (donc plus besoin de `Frame fr = new MonAppli();`)

Passer d'une applet à une application

La technique est de construire un objet de la classe applet, puis de mettre dans la `Frame` de l'application cette applet et de lancer ensuite `init()` et `start()`. On a donc :

```
class MaClasseApplet extends Applet {
• • •
}
```

et on veut à partir de cela construire une application indépendante. On écrit alors un code comme :

```
class MonAppli extends Frame {
• • •
   public MonAppli() {
Applet monApplet = new MaClasseApplet();
monApplet .init();
monApplet .start();
add("Center", monApplet);
pack();
setVisible(true);
   }

   public static void main(String args[ ]) {
new MonAppli();
  }
• • •
}
```

Beaucoup de fonctionnalités des applets ne sont pas implantées dans les applications indépendantes : exécution de sons, récupération de paramêtres du fichier HTML (il n'y a plus de fichier HTML!!), méthodes trop spécifiques aux applets et qui nécessitent le browser (`getParameter()`, `showStatus()`, `getImage()`, ...).

L'applet "élastique" Java

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Event;
import java.awt.Point;
import java.awt.event.*;

public class Lines extends java.applet.Applet
implements MouseListener,
MouseMotionListener  {

Point starts[] = new Point[10]; // les points de  //
départ
Point ends[] = new Point[10];  // les points
// d'arrivée
Point anchor;  // le début de la ligne courante
int currline = 0; // le nombre de lignes
Point currentpoint; // la fin de ligne

public void init() {
setBackground(Color.white);
addMouseListener(this);
addMouseMotionListener(this);
}

public void mouseClicked(MouseEvent e){}

public void mouseEntered(MouseEvent e){}

public void mouseExited(MouseEvent e){}

public void mousePressed(MouseEvent e){
int x = e.getX();
int y = e.getY();
anchor = new Point(x,y);
}
```

```
public void mouseReleased(MouseEvent e){
int x = e.getX();
int y = e.getY();
       addline(x,y);
}

public void mouseDragged(MouseEvent e){
int x = e.getX();
int y = e.getY();
       currentpoint = new Point(x,y);
       repaint();
}

public void mouseMoved(MouseEvent e){}

void addline(int x,int y)
{
    Point stmp[];
    Point etmp[];

    // toutes les 10 lignes, accroître les tableaux
    if (currline % 10 == 0) {
      stmp = new Point[currline + 10];
      etmp = new Point[currline + 10];
      System.arraycopy(starts, 0, stmp, 0,
starts.length);
      System.arraycopy(ends, 0, etmp, 0,
ends.length);
      starts = stmp;
      ends = etmp;
}
    starts[currline] = anchor;
    ends[currline] = new Point(x,y);
    currline++;
    currentpoint = null;
    repaint();
}
```

```
public void paint(Graphics g)
{
    // dessine toutes les lignes
    for (int i = 0; i < currline; i++) {
      g.drawLine(starts[i].x, starts[i].y,
        ends[i].x, ends[i].y);
    }

    // dessine la ligne courante en bleu
    g.setColor(Color.blue);
    if (currentpoint != null)
g.drawLine(anchor.x,anchor.y,currentpoint.x,curr
entpoint.y);
  }
}
```

Les Animations

Bien que ce sujet ne soit pas spécifique aux applets, beaucoup d'applets accessibles par le Web sont des animations. En voici une.

On considère l'applet :

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;

public class ColorSwirl extends
java.applet.Applet implements Runnable {

  Font f = new
Font("TimesRoman",Font.BOLD,48);
  Color colors[] = new Color[50];

  Thread runThread;

  public void start() {
    if (runThread == null) {
      runThread = new Thread(this);
      runThread.start();
    }
  }

  public void stop() {
    if (runThread != null) {
      runThread.stop();
      runThread = null;
    }
  }
```

```
  public void run() {

    // initialise le tableau de couleurs
    float c = 0;
    for (int i = 0; i < colors.length; i++) {
      colors[i] = Color.getHSBColor(c,
(float)1.0,(float)1.0);
      c += .02;
    }

    // cycle through the colors
    int i = 0;
    while (true) {
      setForeground(colors[i]);
      repaint();
      i++;
      try { Thread.currentThread().sleep(50); }
      catch (InterruptedException e) { }
      if (i == (colors.length)) i = 0;
    }
  }

  public void paint(Graphics g) {
    g.setFont(f);
    g.drawString("Diverses couleurs", 15, 50);
  }
}
```

Une telle animation même si elle fonctionne fait apparaître des tremblements. Ceci est du à la gestion du rafraîchissement en Java.

Les tremblements dans les animations

repaint(), update(), paint()

repaint() appelle, dès que cela est possible, la méthode update() qui appelle ensuite paint().
update() dessine le fond d'écran, puis appelle paint() : c'est ce qui produit les tremblements. D'ailleurs update() de la classe Component est :

```
public void update(Graphics g) {
    g.setColor(getBackground());
    g.fillRect(0, 0, width, height);
    g.setColor(getForeground());
    paint(g);
}
```

Première solution : redéfinir update()

on écrit dans update()le seul appel à paint().
Dans le code ci dessus on ajoute simplement :

```
public void update(Graphics g) {
paint(g);
}
```

et il n'y a plus de tremblements

Les tremblements dans les animations (suite)

Optimisation : préciser dans update() la zone de cliping

zone de cliping = zone sensible de dessin (i.e. à l'extérieur rien n'est redessiné). On écrit alors :

```
public void update(Graphics g) {
g.clipRect(x1, y1, x2, y2);
paint(g);
}
```

Les tremblements dans les animations (suite)

Seconde solution : le double-buffering

On prépare tout le dessin à afficher à l'extérieur de l'écran (dans un buffer annexe). Lorsque ce second buffer est prêt, on le fait afficher à l'écran. L'écran a ainsi deux buffers (=> double buffering).
Pour cela on utilise :
1°) deux objets un de la classe `Image`, l'autre de la classe `Graphics`, qu'on initialise dans la méthode `init()`. Les deux objets sont associés.
2°) les dessins se font dans l'instance `Graphics`.
3°) quand tout est dessiné, on associe l'image au contexte graphique de l'applet.

Le corps de `update()` doit être :

```
public void update(Graphics g) {
paint(g);
}
```

Les tremblements dans les animations (suite)

Seconde solution : le double-buffering

Syntaxe

1°) les initialisations sont :

```
Image bufImg;
Graphics bufgc;

public void init() {
bufImg = createImage(this.getSize().width,
this.getSize().height);
bufgc = bufImg.getGraphics();
}
```

2°) et 3°) les dessins sont faits dans le buffer `Graphics` qu'on finit par associer à l'écran. Par exemple:

```
public void paint(Graphics g) {
bufgc.setColor(Color.Black);
bufgc.fillOval(20, 60, 100, 100);
...
// paint() doit obligatoirement se terminer par :
g.drawImage(bufImg, 0, 0, this);
}
```

Java Plug In

est un plug in à mettre dans un navigateur Web (Netscape Navigator ou Internet Explorer) pour utiliser la MV de SUN au lieu de celle du navigateur.

Gros avantages :
les MV de SUN sont à jour vis à vis des versions courantes de Java.

Java plug-in à chercher à :
http://www.javasoft.com/products/plugin/

Disponible aussi pour Linux voir à
http://www.blackdown.org/activator/index.html

anciennement appelé Java Activator.
Technique disponible depuis le 30 Avril 1998.
gratuit.

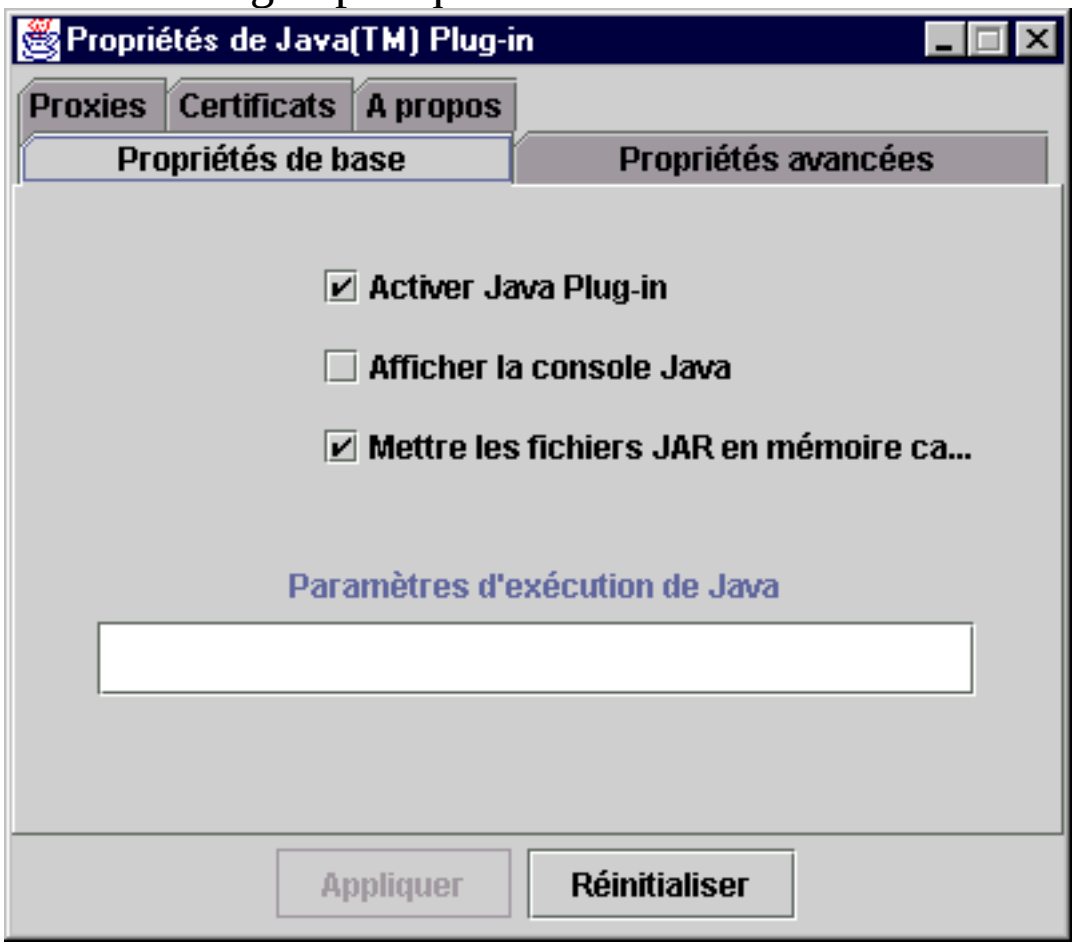
Lors d'une première connexion demande à charger le plug-in

Le plug-in pour Java 1.2 fait environ 5 Mo octets !!

Avec ce plug-in on peut alors :
- exécuter des applets 100% pure Java de la version Java courante
- exécuter des Java Beans

Java Plug In : configuration

Après installation, on peut avoir des renseignements sur ce plug-in à l'aide de l'interface graphique :



Java plug in : plateformes et browsers supportés

	Internet Explorer 3.02	Internet Explorer 4.0	Navigator 3.0	Navigator 4.0
Windows 95	X	X	X	X
Windows 98		X	X	X
Windows NT	X	X	X	X
Solaris/SPARC			X	
Solaris/x86			X	

Le fonctionnement : HTML Converter

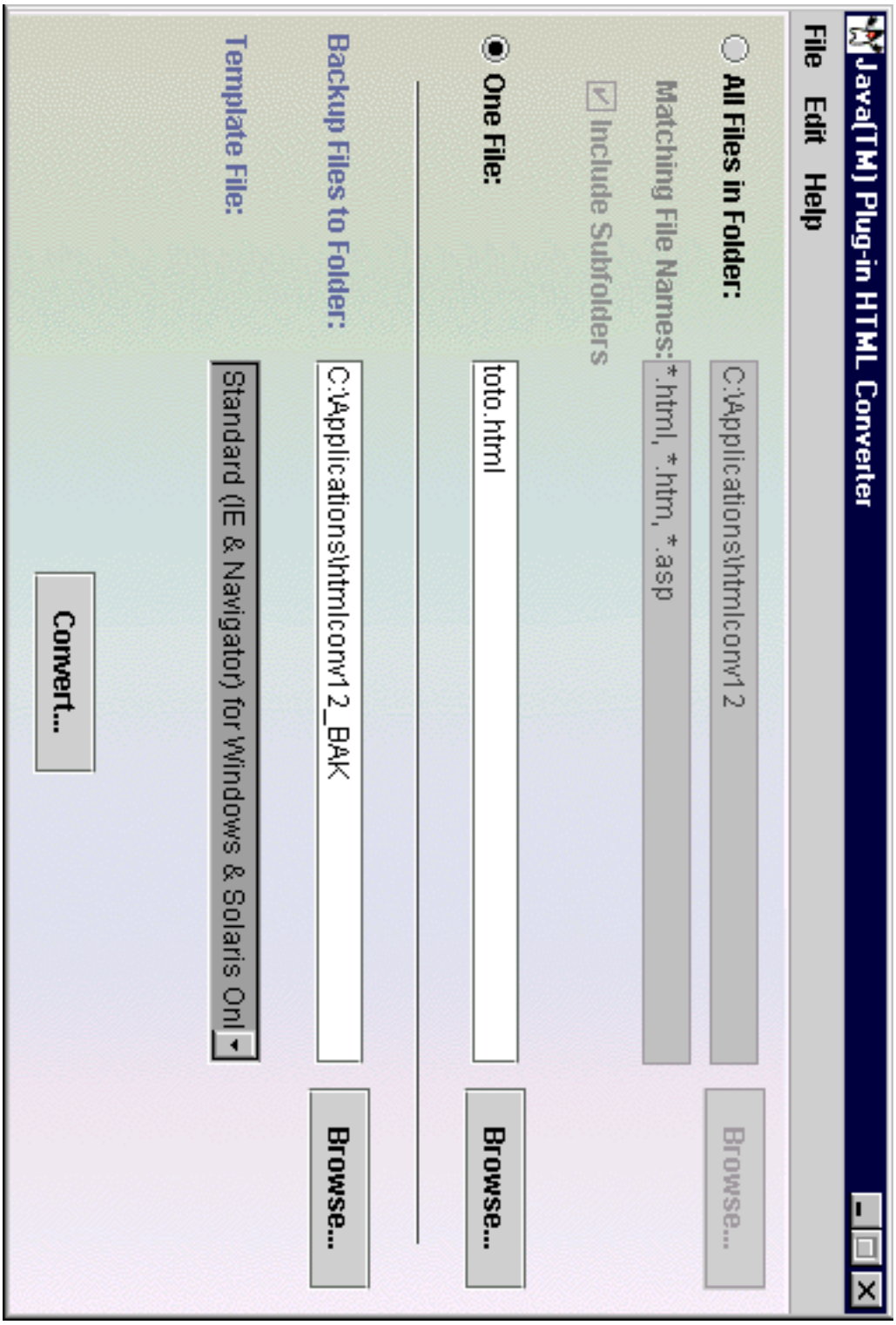
Il faut utiliser une autre balise que la balise APPLET pour utiliser la MV de SUN.
Ce sont les balises :
OBJECT dans le cas de Explorer.
EMBED dans le cas de Netscape.

Il faut donc changer les pages HTML. Un outil est proposé par SUN : HTML Converter (pour 300 Ko environ en .zip pour Java 1.2)

Exemples Java 1.1 à tester à :
http://204.160.241.19/products/plugin/1.1.1/demo/applets.html

HTML Converter

On utilise HTML Converter à l'aide de son interface graphique :



De APPLET à OBJECT

voir documentation à :
http://java.sun.com/products/plugin/1.1.1/docs/tags.html
Pour les plate-formes Windows (95, 98, NT).

remarque

La balise OBJECT est une balise HTML normalisé.

À partir d'une balise comme :

```
<APPLET code="XYZApp.class" codebase="html/" width="200"
height="200">
<PARAM NAME="model"
VALUE="models/HyaluronicAcid.xyz">
</APPLET>
```

Il faut avoir la balise OBJECT suivante :

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-
00805F499D93"
width="200" height="200"
codebase="http://java.sun.com/products/plugin/1.1/jinstall-11-
win32.cab#Version=1,1,0,0">
<PARAM NAME="code" VALUE="XYZApp.class">
<PARAM NAME="codebase" VALUE="html/">
<PARAM NAME="type" VALUE="application/x-java-
applet;version=1.1">
<PARAM NAME="model"
VALUE="models/HyaluronicAcid.xyz">
 </OBJECT>
```

classid est l'identificateur de classe du Java plug-in. Sa valeur est celle ci dessus. Les balises code (et codebase) ne sont pas directement conservées du passage de APPLET à OBJECT : dans OBJECT on utilise la sous-balise PARAM.

De APPLET à OBJECT (fin)

L'attribut codebase de la balise OBJECT a une autre signification : elle indique l'URL de chargement du plug-in Java au cas où ce plug-in ne se trouverait pas sur la machine locale.

le champ type de PARAM indique au Java plug-in le type de programme Java à exécuter i.e. une applet ou un bean. C'est un type MIME.

remarque
les 3 types de PARAM code, codebase et type ne doivent pas apparaître dans les PARAM de la balise APPLET.

De APPLET à EMBED

Pour Netscape Navigator 3 ou 4 sur Windows (95, 98, NT 4.0), Solaris.

La balise :

```
<APPLET code="XYZApp.class" codebase="html/" width="200"
height="200">
<PARAM NAME="model"
VALUE="models/HyaluronicAcid.xyz">
</APPLET>
```

est transformée en :

```
<EMBED type="application/x-java-applet;version=1.1"
width="200" height="200" code="XYZApp.class"
codebase="html/" model="models/HyaluronicAcid.xyz"
pluginspage="http://java.sun.com/products/plugin/1.1/plugin-
install.html">
<NOEMBED>
</NOEMBED>
</EMBED>
```

Contrairement à OBJECT toutes les informations sont des attributs de la balise EMBED => plus de balise PARAM.

Entre autre, les balises PARAM de APPLET (par exemple `<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">`) deviennent des attributs de EMBED (par exemple `model="models/HyaluronicAcid.xyz"`).

type indique le type de programme Java à exécuter : applet ou bean.

pluginspage est utilisé si le Java plug-in n'est pas déjà chargé. Sa valeur indique une URL de chargement de ce Java plug-in.

De APPLET à EMBED (suite)

Entre <NOEMBED> et </NOEMBED>, est mis un texte de remplacement au cas où le browser ne supporte pas la balise EMBED.

Java plug-in dans MSIE et NN

Une balise classique comme :

```
<APPLET code="XYZApp.class" codebase="html/"
align="baseline" width="200" height="200">
<PARAM NAME="model"
VALUE="models/HyaluronicAcid.xyz">

No JDK 1.1 support for APPLET!!
</APPLET>
```

devient, pour être lue par Java plug-in dans MSIE et NN :

Java plug-in dans MSIE et NN (suite)

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-
00805F499D93" width="200" height="200" align="baseline"
codebase="http://java.sun.com/products/plugin/1.1/jinstall-11-
win32.cab#Version=1,1,0,0">
 <PARAM NAME="code" VALUE="XYZApp.class">
<PARAM NAME="codebase" VALUE="html/">
<PARAM NAME="type" VALUE="application/x-java-
applet;version=1.1">
<PARAM NAME="model"
VALUE="models/HyaluronicAcid.xyz">
<COMMENT>
<EMBED type="application/x-java-applet;version=1.1"
width="200" height="200" align="baseline" code="XYZApp.class"
codebase="html/" model="models/HyaluronicAcid.xyz"
pluginspage="http://java.sun.com/products/plugin/1.1/plugin-
install.html">
<NOEMBED>
                No JDK 1.1 support for APPLET!!
</NOEMBED></EMBED></COMMENT>
</OBJECT>
```

Les balises <COMMENT> et </COMMENT> ne sont comprises que par MSIE et ce navigator ignore le contenu entre ces 2 balises. On obtient alors code OBJECT correct pour MSIE. NN ne comprend pas les balises OBJECT et COMMENT et lit :

```
<EMBED type="application/x-java-applet;version=1.1"
width="200" height="200" align="baseline" code="XYZApp.class"
codebase="html/" model="models/HyaluronicAcid.xyz"
pluginspage="http://java.sun.com/products/plugin/1.1/plugin-
install.html">
<NOEMBED>
        No JDK 1.1 support for APPLET!!
</NOEMBED></EMBED>
```

ce qui est le résultat recherché.

Appendice

Communications entre applets

On peut avoir dans une page html deux applets qui s'exécutent et communiquent entre elles. Pour cela :
on écrit un fichier html dans lequel il y a 2 applets :

```
<head><title> Communication entre les applets
</title></head>

<body>
<h1>Communication entre les applets</h1>
<P>

<applet code="applet1.class" width=500
height=100 name="app1">
</applet>


<P>Ci dessus la premiere applet.
<HR><P>
En dessous la seconde. <P>

<applet code="applet2.class" width=500
height=100 name="app2">
</applet>
</html>
```

C'est par l'intermédiaire de l'attribut name qu'elles vont pouvoir communiquer.

Communications entre applets (suite)

Puis on écrit la première applet dont le byte code sera mis dans `applet1.class` :

```
import java.awt.*;
import java.applet.*;

interface intapplet2 {
   public void affiche(String chaine);
}

public class applet1 extends Applet {

public void paint(Graphics g) {
   g.setColor(Color.blue);
   g.drawString ("appuyer avec la souris dans
cette zone",40, 40);
  }

public boolean mouseDown(Event evt, int x, int y)
{
   intapplet2 consommateur = (intapplet2)
getAppletContext().getApplet("app2");
   consommateur.affiche("Coucou");
   return true;
}
}
```

Cette applet ne fait qu'afficher :
"appuyer avec la souris dans cette zone"

Communications entre applets (suite)

Enfin on écrit la seconde applet donc le byte code sera mis dans `applet2.class` :

```
import java.awt.*;
import java.applet.*;

interface intapplet2 {
   public void affiche(String chaine);
}

public class applet2 extends Applet implements
intapplet2 {
String blabla= "Au debut";

public void affiche(String chaine) {
   blabla = chaine;
   repaint();
}

public void start() {
   affiche(blabla);
}

public void paint(Graphics g) {
   g.setColor(Color.red);
   g.drawString ("Ici on recoit ce qui est envoye par
l'autre applet",60, 40);
  g.drawString (blabla, 20,40);
   }
}
```

Au départ on a :

après clic dans la zone "sensible", on a :