

Introduction au MIDlet Pascal 2.02

par Darryl KPIZINGUI ([site](#)) Jules AKAKPO-TOULAN ([site](#))

Date de publication : 5 Janvier 2010

Dernière mise à jour : 5 Janvier 2010

Cet article présente MIDlet Pascal 2.02 et montre comment le prendre en main. MIDlet Pascal utilise un langage intermédiaire au Pascal standard tout en alliant quelques spécificités de J2ME (Java 2, Micro Edition).

I - Introduction.....	3
II - Installation de MIDlet Pascal 2.02.....	3
II-A - Installation sous Windows.....	3
II-B - Installation sous Linux.....	7
III - Présentation de l'interface.....	8
III-A - Les menus.....	8
III-B - La barre d'outils.....	11
III-C - La console.....	12
IV - Premier programme.....	12
V - Compilation et exécution d'un projet.....	12
V-A - Compilation du projet.....	12
V-A-1 - Théorie sur la compilation.....	12
V-A-2 - Pratique de la compilation.....	13
V-A-3 - Fichiers jad et jar générés lors de la compilation.....	15
V-B - Exécution du projet.....	15
V-B-1 - Exécution du projet dans un émulateur.....	15
V-B-2 - Déploiement et exécution sur téléphone portable.....	16
VI - Création et utilisation des unités.....	17
VI-A - Théorie.....	17
VI-B - Pratique.....	17
VI-C - Utilisation de l'unité par d'autres programmes.....	19
VI-D - Importation d'unités.....	19
VII - Ressources.....	19
VII-A - Utilisation des images.....	20
VII-B - Exemple de code.....	20
VIII - Conseils.....	24
IX - Conclusion.....	25

I - Introduction

MIDlet Pascal existe sous 2 licences depuis sa sortie en 2006. Nous n'aborderons que les points de la version gratuite disponible.

Le compilateur intégré à l'IDE produit du bytecode J2ME exécutable sur tout téléphone mobile supportant minimum MIDP 1.0 et la plateforme CLDC 1.0. Les fichiers jad et jar produits peuvent également être exécutés avec des émulateurs dont en voici les principaux:

- **Java Wireless Toolkit**
- **Motorola SDK for J2ME**
- **Siemens Mobility Toolkit**
- **SonyEricsson SDK for J2ME**
- **Nokia J2ME SDK's**

Il est facile de programmer sous MIDlet Pascal une fois les bases du Pascal standard acquises. La version que nous abordons dans ce tutoriel est l'édition **personnelle** et donc à utiliser pour des applications à but non commercial.



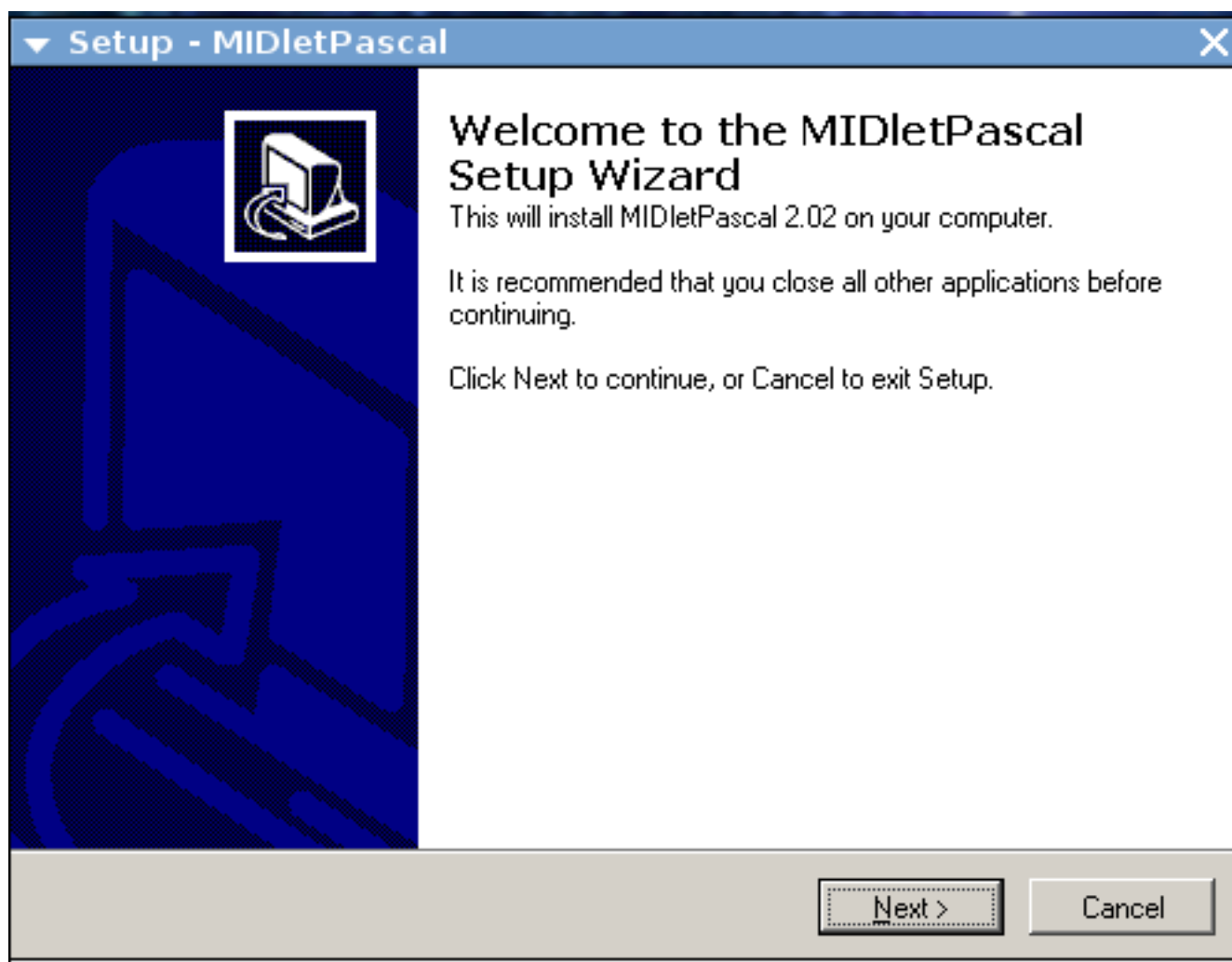
II - Installation de MIDlet Pascal 2.02

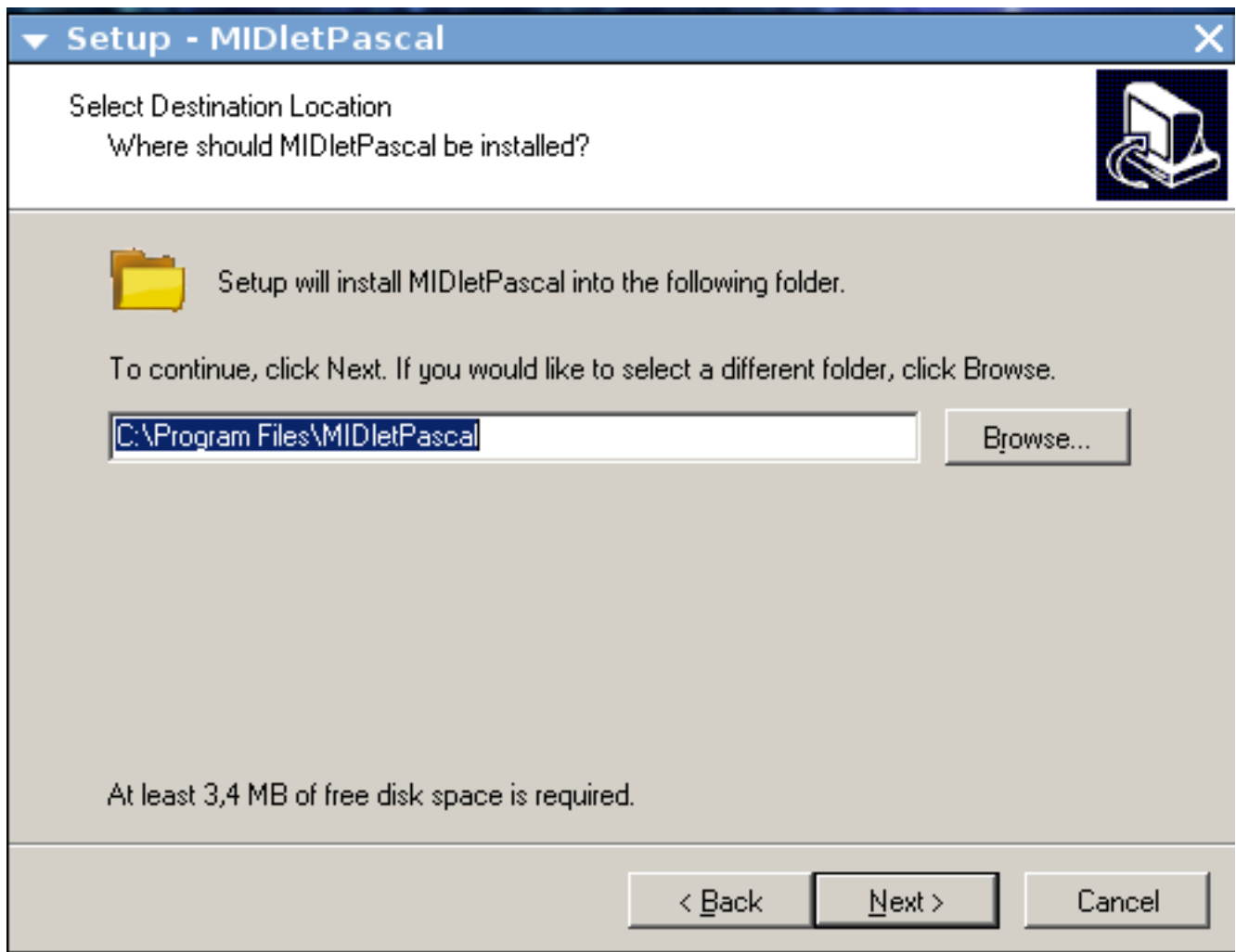
MIDlet Pascal est fait pour n'être utilisé que sous Windows, mais il peut l'être également sous Linux avec l'émulateur wine.

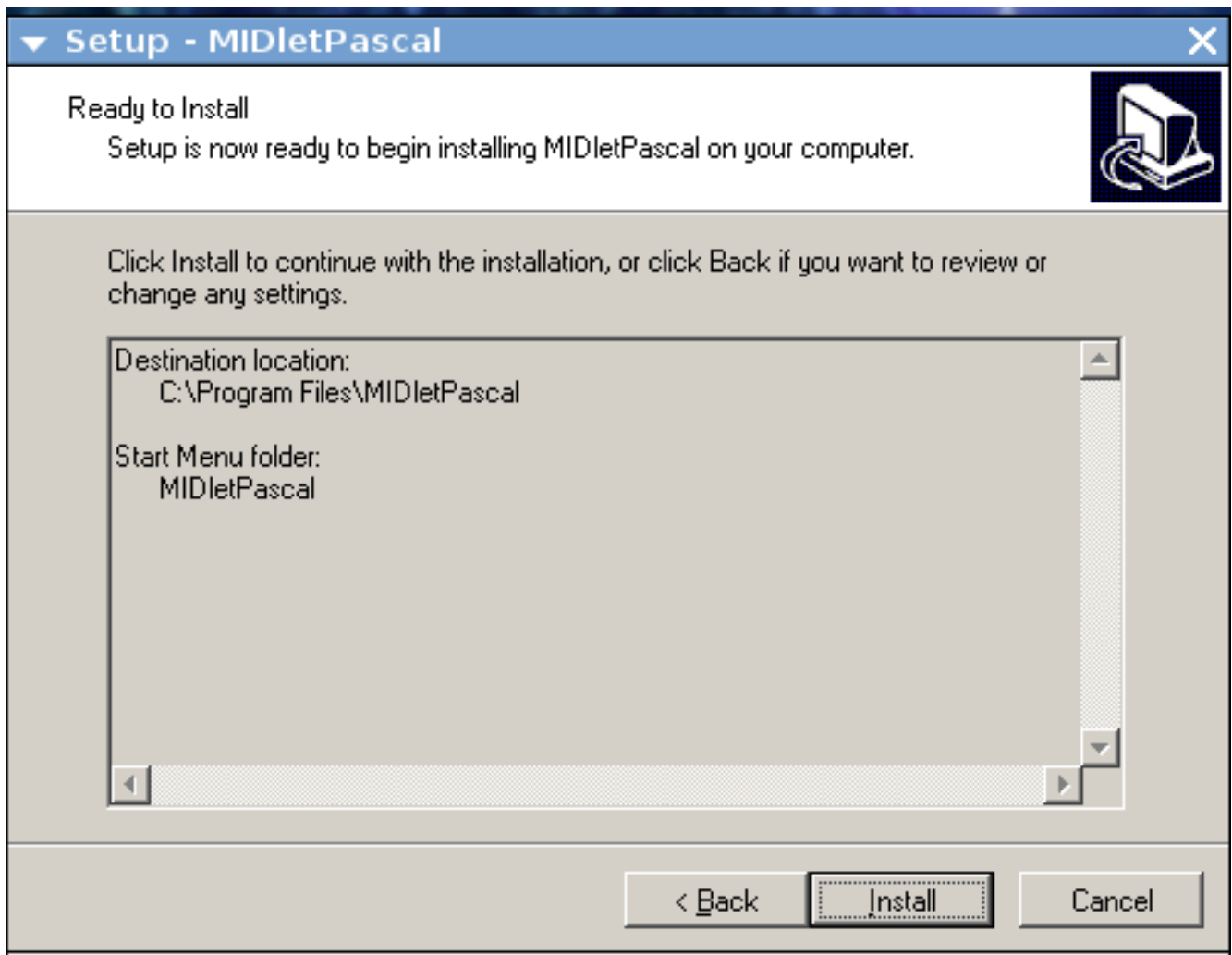
II-A - Installation sous Windows

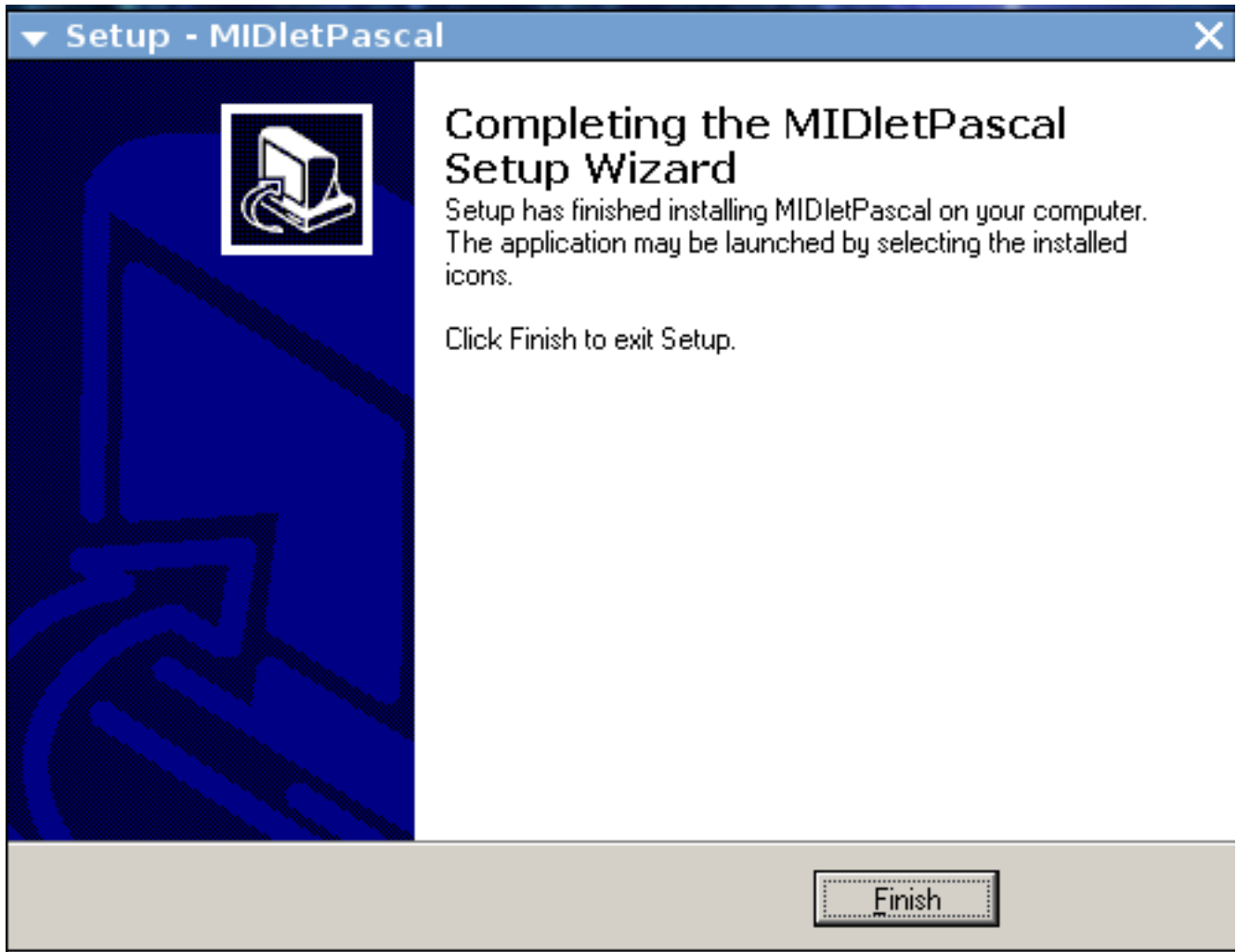
MIDlet Pascal est disponible en téléchargement sur www.midletpascal.com ou sur www.developpez.com.

Une fois téléchargé, double-cliquez sur **MPInstall202.exe**. Ensuite suivez les instructions jusqu'à la fin de l'installation.









L'installation est finie; à présent, vous pouvez lancer l'outil.

 **MIDlet Pascal peut s'installer sur toutes les versions de Windows**

II-B - Installation sous Linux

L'installation sous Linux se fait en deux temps si vous ne disposez pas de wine ([site officiel](#)).

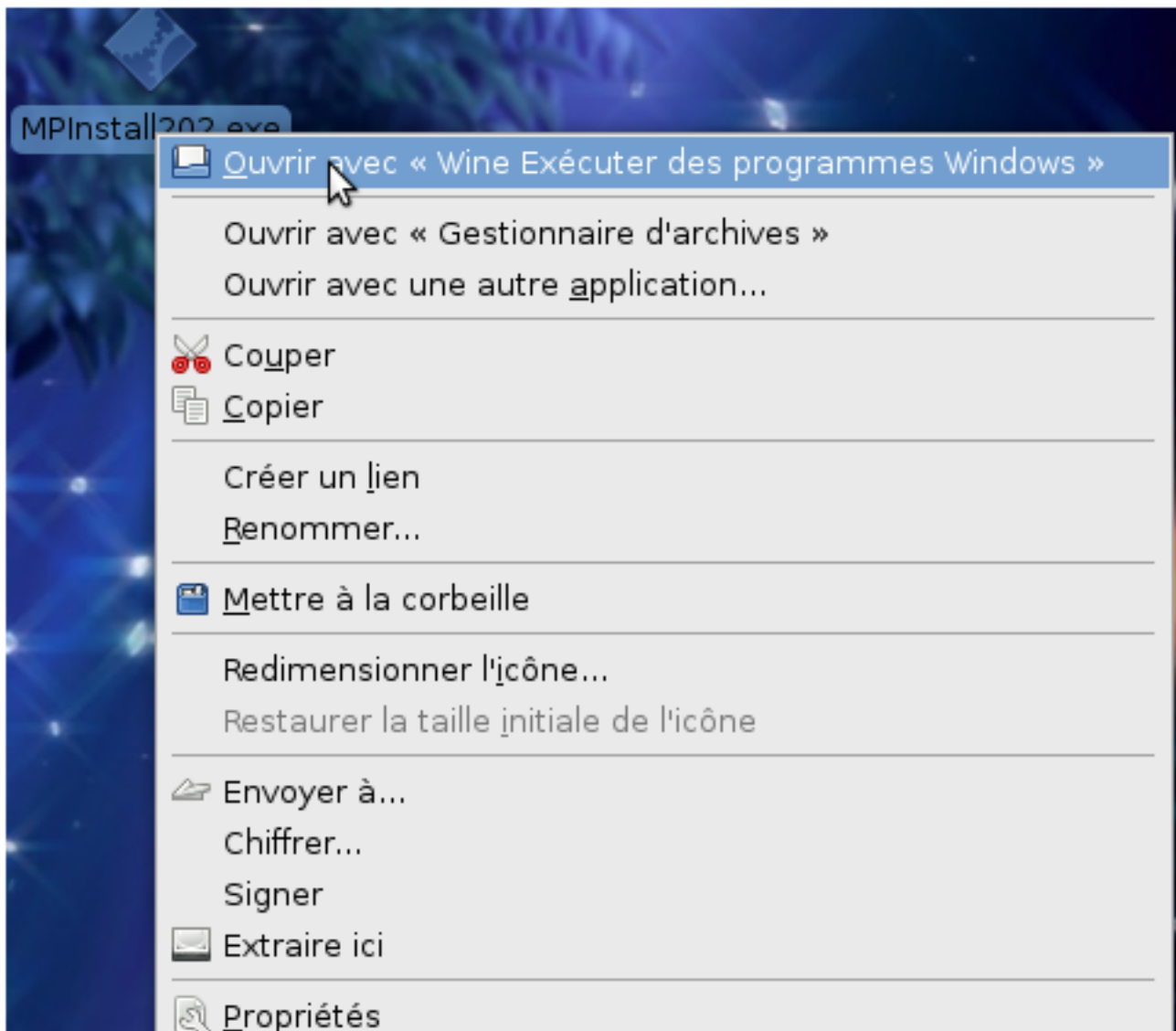
- Etape 1 : Installation de wine
Si vous avez déjà wine installé vous pouvez passer à la seconde étape

Vous pouvez installer *wine* sur votre distribution à l'aide du gestionnaire de paquets ou vous pouvez aller le télécharger sur le site officiel pour être sûr de bénéficier d'une version récente.

- Etape 2: Installation de MIDlet Pascal 2.02

L'installation sous Linux est identique à celle sous Windows, à la différence que vous avez besoin de wine pour l'installer.

Une fois téléchargé, exécutez-le avec l'émulateur wine.

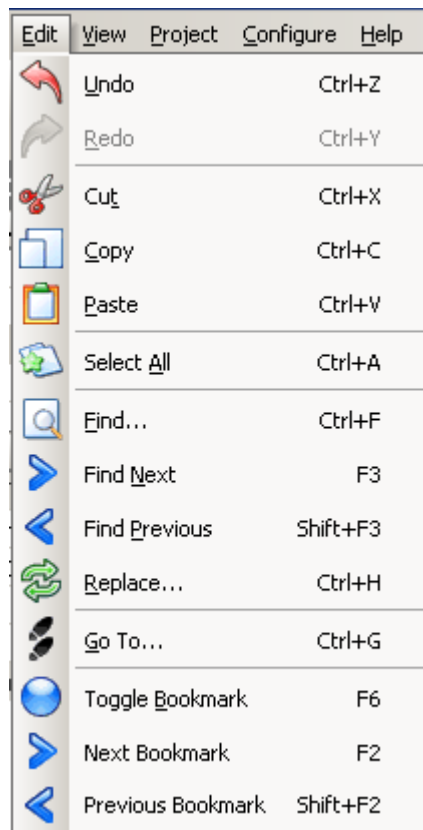


III - Présentation de l'interface

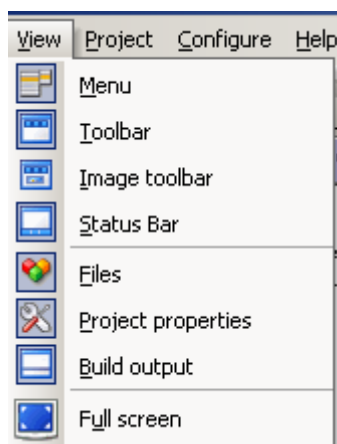
III-A - Les menus

Nous présenterons seulement quelques menus de l'interface, les autres étant assez intuitifs.

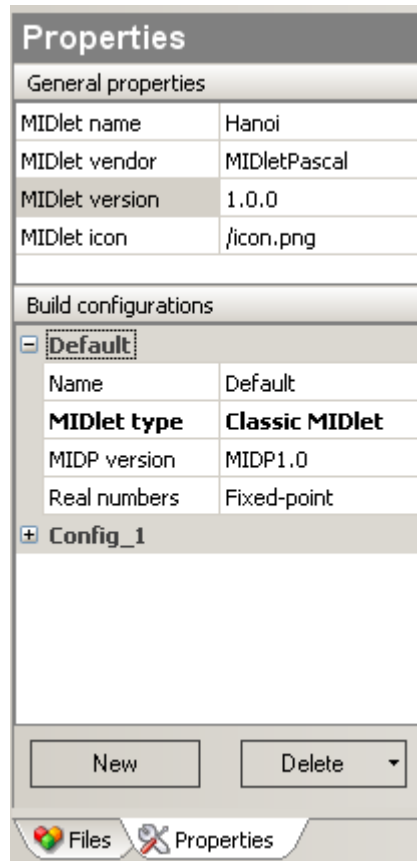
- **L'outil d'édition et de gestion**



- **Gestion de la vue:** Activer ou désactiver les vues sur certaines parties de l'interface



- **La gestion des éléments de projets**



Pour créer une unité pour votre projet, cliquez sur **New Source File**. Ceci ouvre une petite fenêtre où vous entrerez le nom de l'unité.


Après validation, vous obtiendrez une nouvelle fenêtre avec ce contenu:

```
unit votre_unite;

interface
{ add public declarations here }

implementation
{ add unit functions and procedures here }

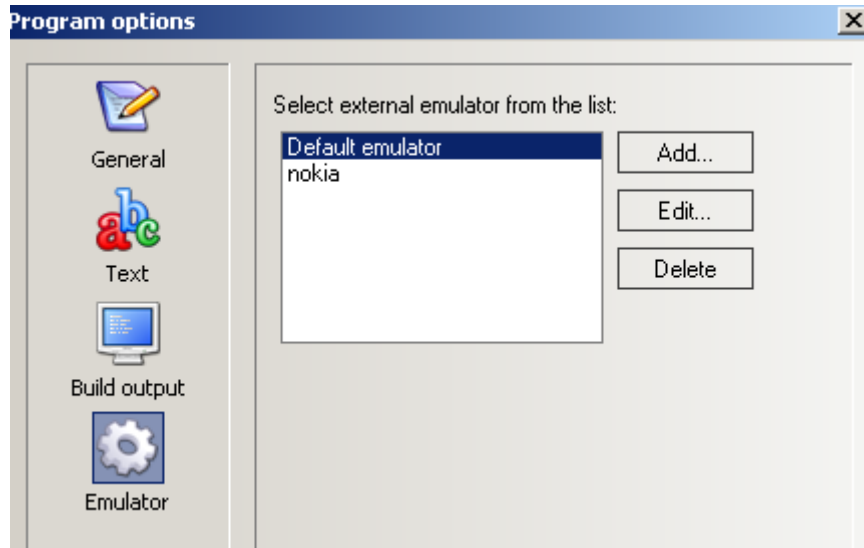
initialization
{ add initialization code here }
end.
```

 Vous avez la possibilité de créer directement de petites images sous MIDlet Pascal en cliquant sur **New Image resource**. Ces images peuvent principalement être de petites images textuelles ou des icônes.

*Vous pouvez également importer vos propres images (**format png**) ou des sons (**format MIDI**).*

- **Les options**

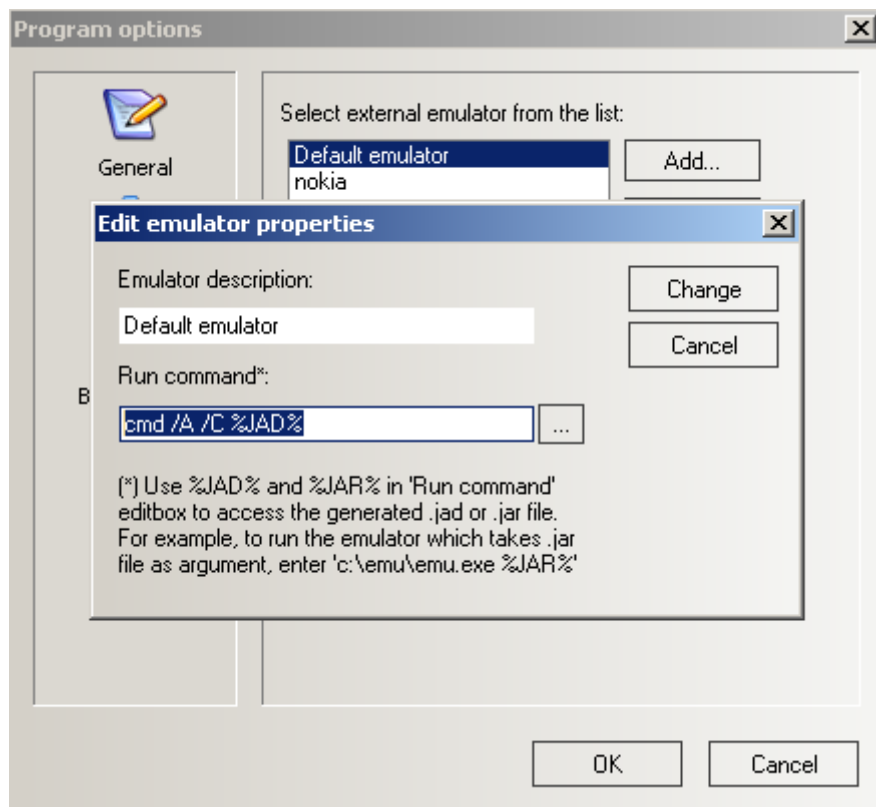
La partie essentielle est sans doute celle pour choisir ou ajouter un émulateur pour l'exécution.



Sous Windows l'émulateur par défaut choisi permet de lancer votre application. Sous Linux, il faut entrer la bonne commande pour que l'émulateur puisse être lancé.

Lors de l'exécution de votre application sous Windows c'est cette commande qui est exécutée.

```
cmd /A /C %JAD%
```



Il suffit donc de sélectionner le bon path vers l'exé sous Linux.

III-B - La barre d'outils

Elle ne contient que des raccourcis vers certains éléments des menus ci-dessus



III-C - La console

En compilant votre programme, vous verrez apparaître sur la console ce qu'a produit le compilateur. Exemple: en compilant le source **projet_test.mpsrc**(mpsrc = midlet pascal source).

```
Build output
Build configuration: Default (MIDP 1.0, classic MIDlet)
Checking for unit dependencies...

Compiling 'projet_test.mpsrc'...
Done - 0 error(s), 0 warning(s)

Creating JAR file...
JAR file created
Creating JAD file...
JAD file created
Build successfull (configuration: Default)
```

IV - Premier programme

Par défaut, la création d'un nouveau projet fait apparaître dans la fenêtre:

```
program projet_test;
begin
  drawText('Hello world!', 0, 0);
  repaint;
  delay(2000);
end.
```

Nous verrons dans la suite les prochaines étapes.

V - Compilation et exécution d'un projet

MIDlet Pascal utilise le langage Pascal pour produire du binaire Java. Le langage Pascal est le Pascal standard, avec quelques spécificités du J2ME. Mais le support du Pascal n'est pas encore complet : le passage par adresse, la structure case et les procédures internes, par exemple, ne sont pas encore supportés.

Voyons maintenant les détails de la compilation, jusqu'à l'exécution d'un projet sous MIDlet Pascal

V-A - Compilation du projet

V-A-1 - Théorie sur la compilation

La compilation sous MIDlet Pascal est du moins semblable à la compilation d'une application sous Turbo Pascal. Mais avec MIDlet Pascal, le programme, une fois compilé, est traduit en binaire Java qui peut être directement déployé et exécuté sur les téléphones portables dotés d'une machine virtuelle Java (*JVM*).

La compilation génère deux fichiers : un fichier .jar et un fichier .jad. Le fichier jar contient le programme exécutable, et le fichier jad donne des informations sur le fichier jar et l'application en général. Voici par exemple, le contenu d'un fichier jad :

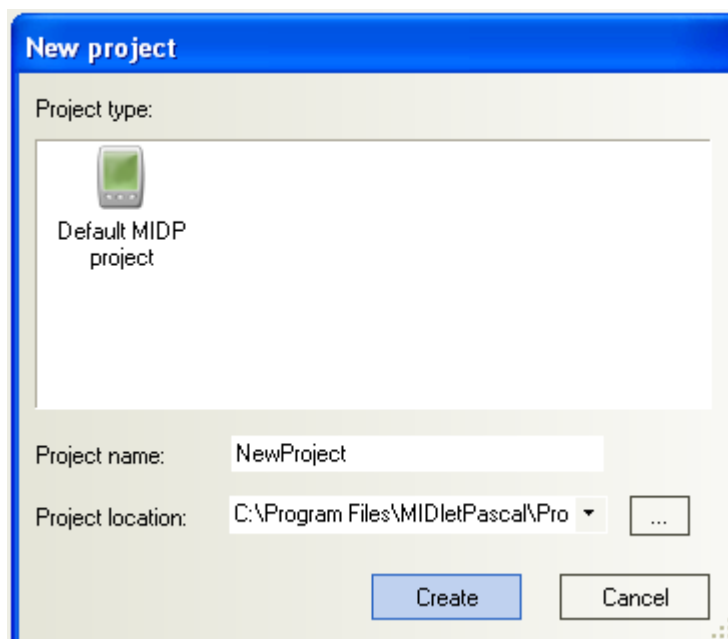
```
MIDlet-1: test, /icon.png, main
```

```
MIDlet-Jar-Size: 33610
MIDlet-Jar-URL: test.jar
MIDlet-Name: test
MIDlet-Vendor: darrylsite
MIDlet-Icon: /icon.png
MIDlet-Version: 1.0.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

Nous pouvons par exemple lire que le chemin du programme exécutable est test.jar, la taille du fichier jar est 33610 octets et la version de l'application est 1.0.0.

V-A-2 - Pratique de la compilation

Nous allons maintenant compiler en pratique notre premier code sous MIDlet Pascal. Lancez MIDlet Pascal. Dans le menu fichier choisissez « New Project » (nouveau projet).



Dans la fenêtre qui apparaît, cliquer sur l'icône « **Default MIDP project** » qui est la configuration par défaut, et c'est ce que nous allons utiliser. Ensuite indiquer le nom de votre projet, puis l'emplacement de ce dernier sur le disque. Cliquer sur « Create » pour finir la création du projet. Nous allons choisir « hello » comme nom pour notre premier projet.

Le programme que nous allons compiler est un programme qui fait défiler un texte sur l'écran. Le texte sera bien sûr le "Hello World" bien plus que classique.

Le programme utilise les instructions standard du Pascal. Les principales fonctions propres à MIDlet Pascal que nous avons utilisées sont :

- **setFont()**, qui définit la police, le style et la taille des textes affichés sur l'écran
- **getWidth()** et **getStringWidth()**, qui respectivement donnent la largeur de l'écran et la largeur de la chaîne de caractères passée en paramètre une fois affichée sur l'écran
- **drawText()**, affiche la chaîne de caractères passée en paramètre à l'écran
- **GetKeyClicked**, donne la dernière touchée appuyée sur le clavier
- **Repaint**, redessine l'écran. Toute modification concernant l'affichage n'est visible à l'écran qu'après un appel de cette fonction.

Maintenant, copiez et collez le code suivant dans la fenêtre de l'éditeur qui s'affiche :

```
program hello;

//fait rotation des caractères d'une phrase

function rotation(tex: string) :string;
var c, d: char;
    i :integer;
begin
c:=getChar(tex,0);
for i:=length(tex)-1 downto 0 do
begin
d:=getChar(tex,i);
tex:=setChar(tex, c, i);
c:=d;
end;
rotation:=tex;
end;
{-----}

{cree l'effet de texte défilant et l'affiche sur l'ecran}
procedure animation;
const texte='Hello world - MidletPascal - developpez.com *** ';
    dim=20;
var rot, aux : string;      a, b : integer;

begin

    rot:=texte;
    //SetFont(FONT_FACE_SYSTEM,FONT_STYLE_BOLD,FONT_SIZE_LARGE);

    //couleur bleue

    setColor(0, 0, 250);
    rot:=texte;
    //on cherche a centrer le texte sur l'ecran

    b:=(getHeight div 2) - (getStringHeight(copy(rot,0, dim)) div 2);
    a:=(getWidth div 2) - (getStringWidth(copy(rot,0, dim)) div 2);

    repeat

        setColor(50, 114, 184);
        aux:=copy(rot,0, dim);
        drawText(aux, a, b);
        rot:=rotation(rot);
        repaint;

        delay(250);

        SetColor(255, 255, 255);
        FillRect(a, b, getStringWidth(aux)+20, getStringHeight(aux));

    until GetKeyClicked <> KE_NONE;

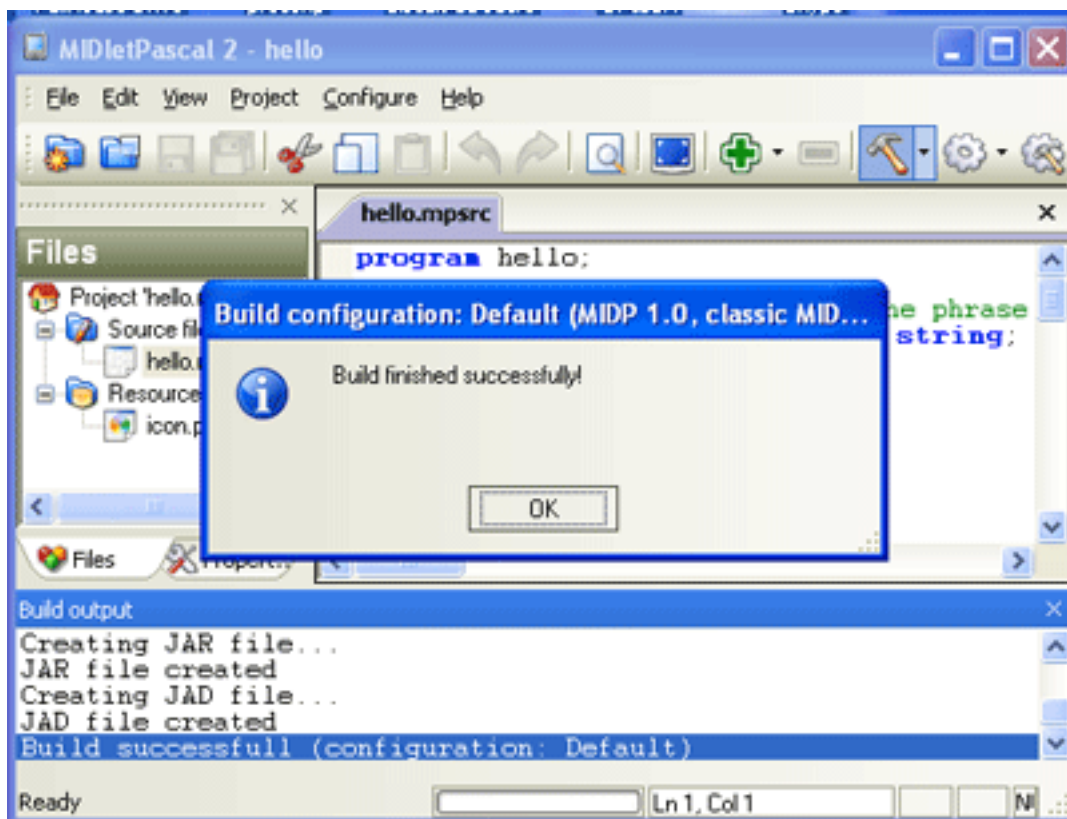
end;

BEGIN

animation;

END.
```

Dans la barre de menu, choisissez « Project » ; puis cliquez sur « Build project » pour compiler le projet. Si la compilation se passe bien, vous aurez la fenêtre suivante :



V-A-3 - Fichiers jad et jar générés lors de la compilation

Allez dans le dossier de votre projet, chez moi c'est « c:\midletPascal\hello ». Ouvrez le dossier bin\ et vous verrez les deux fichiers hello.jar et hello.jad créés lors de la compilation. Vous pouvez par curiosité ouvrir le fichier hello.jad avec un simple éditeur de texte comme **pspad** ou **notepad** afin de voir son contenu.

V-B - Exécution du projet

Voyons comment exécuter notre projet dans un émulateur, et ensuite comment déployer l'application et l'installer sur les téléphones portables.

V-B-1 - Exécution du projet dans un émulateur

Avant d'aller plus loin, nous allons apprendre comment configurer MIDlet Pascal pour exécuter le programme dans un émulateur. Nous allons utiliser l'émulateur de Sun que vous pouvez télécharger depuis le lien suivant : **Java Wireless Toolkit**

Une fois l'émulateur installé, on peut exécuter le programme sur le poste de travail. Allez dans le menu « projet » puis cliquez sur « Run midlet ». Aucune configuration n'est requise puisque l'émulateur installé devient le logiciel par défaut qui exécute les fichiers jad. Mais si vous désirez utiliser un autre émulateur, allez dans le menu « Configurer », puis choisissez « Program options ». Choisissez ensuite l'onglet « Emulator » pour configurer l'émulateur.

Pour exécuter le projet, dans le menu « Project » choisissez « Run Midlet ». Exécution du programme dans l'émulateur donne :



V-B-2 - Déploiement et exécution sur téléphone portable

Il y a deux moyens possibles de transférer les applications créées avec MIDlet Pascal sur les téléphones portables : il est possible de connecter le téléphone à un ordinateur ou d'utiliser internet afin d'effectuer ce transfert.

La connexion avec un ordinateur est possible en utilisant des câbles fournis avec le téléphone, l'infrarouge, □ Une fois la connexion établie, les deux fichiers jar et jad doivent être transférés vers le téléphone. Certains téléphones n'ont besoin que du fichier jar pour installer l'application.

Le moyen le plus utilisé est sans doute internet. Vous pouvez en utilisant internet distribuer vos applications à un grand public. Pour ce faire, il faut charger sur un serveur web ou wap les deux fichiers jar et jad générés lors de la compilation. Le fichier à télécharger est le jad qui contient le chemin vers le fichier jar qui va ensuite être téléchargé automatiquement par le téléphone.

Si vous vous souvenez encore du contenu du fichier jad que nous avons présenté plus haut,

```

MIDlet-1: test, /icon.png, main
MIDlet-Jar-Size: 33610
MIDlet-Jar-URL: test.jar
MIDlet-Name: test
MIDlet-Vendor: darrylsite
MIDlet-Icon: /icon.png
MIDlet-Version: 1.0.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
    
```

La ligne « MIDlet-Jar-URL: test.jar » indique le chemin pour accéder au fichier jar. Si vous voulez mettre le jar et le jad dans des répertoires différents, vous devez modifier cette ligne de sorte à indiquer le chemin du fichier jar. Par exemple :

```

MIDlet-Jar-URL: http://darrylsite.developpez.com/midlet/test.jar
    
```


VI - Création et utilisation des unités

Quand nous faisons un gros programme, il devient souvent difficile de maintenir le tout dans un même fichier source. Il convient alors de le diviser en plusieurs fichiers facilement maniables afin de faciliter la maintenance et le suivi du projet.

Il arrive souvent, aussi, que nous devions coder un ensemble de fonctions qu'on aura à utiliser dans plusieurs projets, ou distribuer afin que d'autres les utilisent.

Pour répondre à ces divers besoins, on utilise fréquemment en Pascal des unités.

Une unité est une liste de fonctions, de variables et de procédures traitant d'un même thème mises ensemble dans un fichier source. Et ces procédures et fonctions sont par la suite réutilisables dans d'autres programmes ou unités.

VI-A - Théorie

Une unité sous MIDlet Pascal comprend trois parties :

- Interface
- Implémentation
- Initialisation

Dans l'interface, on définit l'entête des fonctions que d'autres programmes pourront utiliser. C'est dans la partie implémentation que ces fonctions seront déclarées entièrement avec leur code. On peut aussi déclarer dans les entêtes des fonctions qui n'ont pas été définies dans la partie implémentation. La dernière partie -initialisation- est réservée à l'initialisation des variables globales de l'unité.

Avec MIDlet Pascal, on ne peut définir qu'une unité par fichier, et le nom du fichier doit correspondre au nom de l'unité.

Format d'une unité sous MIDlet Pascal : PremierUnite.mpsrc

```
unit premierUnite;

interface
{ procedure et fonctions publiques }

implementation
{ codes des procedures et fonctions}

initialization
{ initializations des variables}
end.
```

VI-B - Pratique

Lancez MIDlet Pascal, et créez un nouveau projet. Nommons-le hello2. Allez dans le menu projet et choisissez « New source file ». Dans la boîte de dialogue qui s'affiche, tapez le nom de l'unité « transform ».

Nous allons transformer notre programme précédent en un programme principal et une unité. Voyons le code de l'unité :

```
unit transform;

interface

  procedure animation; //affiche le texte defilant

  procedure setTexte(s : String); //definit le texte a faire defiler

implementation
```

```

var texte : String; //le texte a faire defiler

    dim : integer; // nombre de caractere a afficher

procedure setTexte(s : String);
begin
    texte:=s;
end;

//fait rotation des caractere d'une phrase

function rotation(tex: string) :string;
var c, d: char;
    i :integer;
begin
c:=getChar(tex,0);
for i:=length(tex)-1 downto 0 do
begin
    d:=getChar(tex,i);
    tex:=setChar(tex, c, i);
    c:=d;
end;
rotation:=tex;
end;

{cree l'effet de texte defilant et l'affiche}
procedure animation;

var rot, aux : string;
    a, b : integer;

begin
    //on affiche le texte defilant

    rot:=texte;
    setColor(0, 0, 250);
    rot:=texte;
    //position du texte à l'ecran

    //on affiche le texte centre sur l'ecran

    b:=(getHeight div 2) - (getStringHeight(copy(rot,0, dim)) div 2);
    a:=(getWidth div 2) - (getStringWidth(copy(rot,0, dim)) div 2);

    repeat
        setColor(50, 114, 184);
        aux:=copy(rot,0, dim);
        drawText(aux, a, b);
        rot:=rotation(rot);
        repaint;
        delay(250);
        SetColor(255, 255, 255);
        FillRect(a, b, getStringWidth(aux)+20, getStringHeight(aux));
    until GetKeyClicked <> KE_NONE;
end;

initialization

dim:=20;

end.

```

Vous pouvez voir qu'il n'y a pas de modification importante par rapport au code précédent. Nous avons mis les deux fonctions **setText()** et **animation()** dans la partie interface. Dans la partie **implementation**, nous avons défini le

code de ces deux fonctions, et aussi d'autres fonctions comme `rotation()`, qui est utilisée par `animation()`, mais qui ne pourra être appelée par un autre programme ou unité utilisant l'unité `transform` puisque son entête ne figure pas dans la partie `Interface`.


Dans la dernière partie, à savoir `initialization`, nous initialisons `dim` à 20, qui est le nombre de caractères à afficher sur l'écran.

VI-C - Utilisation de l'unité par d'autres programmes

Pour utiliser l'unité dans un programme, il suffit d'ajouter après le nom du programme la clause `uses` suivie du nom de l'unité. Si vous voulez utiliser une unité depuis une autre unité, la clause « `uses` » doit être placée immédiatement après la clause « `implementation` ».

Une fois la déclaration faite, toutes les procédures et variables disponibles dans l'interface de l'unité peuvent être utilisées.

Voyons le code du programme principal de notre programme utilisant l'unité `transform`.



```
program hello;
//appel de l'unite

uses transform;

BEGIN
setTexte(' hello world -- darrylsite -- developpez.com --');
animation;

END.
```

Lorsque que plusieurs unités ont les mêmes noms pour des identificateurs, on peut précéder le nom de l'identificateur du nom de l'unité qui le contient, suivi d'un point. Et ceci est indispensable si on veut utiliser des variables, des types ou des constantes déclarées dans la partie `interface` d'une unité.

```
program hello;
//appel de l'unite

uses transform;

BEGIN
Transform.setTexte(' hello world -- darrylsite -- developpez.com --');
Transform.animation;

END.
```

VI-D - Importation d'unités

Il nous arrive fréquemment de réutiliser des unités provenant d'autres projets. Avec certains compilateurs comme Turbo Pascal ou Free Pascal par exemple, il suffit de mettre l'unité dans le même dossier puis de lancer la compilation. Avec MIDlet Pascal, il faut créer une nouvelle unité dans le projet en cours, puis copier/coller le code de l'ancienne unité dans celle que vous venez de créer.

VII - Ressources

Les ressources MIDlet Pascal sont des fichiers que nous pouvons importer et utiliser dans un projet. Ces fichiers peuvent être des images, de la musique, ou tout autre type de fichier.

Nous allons donc voir comment importer et utiliser dans images dans un projet. Puis nous examinerons un exemple de code qui simule une horloge analogique.

VII-A - Utilisation des images


Créez un nouveau projet, nommons-le hello3.

Dans le menu « Project », choisissez « Import ressource file ». Choisissez l'image à ajouter puis cliquez sur ouvrir. Vous pouvez voir dans l'onglet « Files » à gauche le fichier image ajouté.

Vous pouvez aussi choisir de dessiner l'image à l'aide de l'éditeur d'image de MIDlet Pascal. Pour cela, dans le menu « Project », choisissez « New image ressource ». Indiquez la taille de l'image et son nom puis cliquez sur « Ok ».

Voyons un petit programme utilisant les images.

Téléchargez le fichier image suivant puis ajoutez-le à notre projet hello3.

 *Concernant les images, seul le format **png** est supporté.*




Nous allons utiliser le type « image » disponible dans MIDlet Pascal pour manipuler les ressources images.

Le code suivant charge l'image contenue dans le fichier image.png précédemment ajouté à notre projet, et l'affiche à l'écran.

```
Var monImage : Image ;

Begin
  monImage :=loadImage('/horloge.png') ;
  drawImage(monImage, 10, 10);
  repeat until getKeyPressed<>KE_NONE;
End;
```

 *Le chemin de l'image doit être précédé du **slash** « / », sinon la compilation se passera bien, mais à l'exécution on aura une erreur et le programme se bloquera.*

VII-B - Exemple de code

Nous voilà maintenant bien armés pour faire un petit programme affichant une horloge analogique.

Nous allons utiliser le fichier horloge.png que vous avez précédemment téléchargé pour nous servir de cadran.

Créez un nouveau projet, nommons-le « horloge ». Ajoutez l'image horloge.png à votre projet.

Nous allons diviser notre programme en une unité *uHorloge* en plus du programme principal.

L'unité *uHorloge* se contentera de dessiner les aiguilles de notre l'horloge en se basant sur l'heure de notre téléphone.

Voyons le code de l'unité *uHorloge* :

```
unit uhorloge;

interface

  //(x, y) position du centre de l'horloge
```

```
//r le rayon du cadran

procedure init(x, y, r : integer);
procedure dessiner;

implementation

const PI=3.141592653;

var
    centre : record // le centre de l'horloge
        x, y : integer;
    end;

    rayon : integer; // le rayon de l'horloge.

    // On considere que l'horloge est un cercle

procedure init(x, y, r : integer);
begin
    centre.x:=x;
    centre.y:=y;
    rayon:=r;
end;

//arrondie un reel positif a l'entier le plus proche

function Round(a : real):integer;
var c: integer;
    b : real;
begin
    c:=trunc(a);
    b:=a-c;
    if (b<0.5) then
        Round:=c
    else
        Round:=c+1;
    end;

//convertit les radians en degres

function radToDegre(a: real) : integer;
begin
    radToDegre:=round(180*a/pi);
end;

//dessine un cercle de centre (x, y) et de rayon r

procedure drawCircle(x, y, r : integer);
begin
    drawEllipse(x-r, y-r, r*2, r*2);
end;

//dessine les aiguille de l'horloge

procedure dessiner;
var h, m, s, time : integer;
    hPos, mPos, sPos : record
        x, y : integer;
    end;

    a, b, c :real;
    img : Image;
begin
    //on obtient l'heure actuelle

    time := GetCurrentTime;
    h:=getHour(time);
```

```

m:=getMinute(time);
s:=getSecond(time);

//on calcule la position des aiguilles

//les secondes
a:=s * PI / 30 - PI / 2;
sPos.x := round( (cos(s * PI / 30 - PI / 2) * (rayon-5) + Centre.x));
sPos.y := round(sin(s * PI / 30 - PI / 2) * (rayon-5) + Centre.y);
//minutes
b:=m * PI / 30 - PI / 2;
mPos.x:= round((cos(m * PI / 30 - PI / 2) * (rayon-10) + Centre.x));
mPos.y:= round( (sin(m * PI / 30 - PI / 2) * (rayon-10) + Centre.y));
//heures
c:=(h*30 + m / 2) * PI / 180 - PI / 2;
hPos.x := round((cos((h*30 + m / 2) * PI / 180 - PI / 2) * (rayon-20)+ Centre.x));
hPos.y := round((sin((h*30 + m / 2) * PI / 180 - PI / 2) * (rayon-20)+ Centre.y));

//on dessine la scene

//seconde
setColor(204, 0, 0);
drawLine(centre.x, centre.y, sPos.x, sPos.y);

//minute
setColor(150, 50, 210);
drawLine(centre.x, centre.y+1, mPos.x, mPos.y);
drawLine(centre.x, centre.y-1, mPos.x, mPos.y);
drawLine(centre.x, centre.y, mPos.x, mPos.y);

//heure
setColor(0, 0, 0);
drawLine(centre.x, centre.y, hPos.x, hPos.y);
drawLine(centre.x, centre.y+1, hPos.x, hPos.y);
drawLine(centre.x, centre.y-1, hPos.x, hPos.y);
drawLine(centre.x, centre.y+2, hPos.x, hPos.y);
drawLine(centre.x, centre.y-2, hPos.x, hPos.y);
end;

initialization

end.

```

Dans l'interface, nous plaçons les procédures `init(x, y, r)`, qui indique la position du centre de l'horloge et le rayon de celle-ci, et `dessiner()`, qui dessine les aiguilles à l'écran.

Le code suivant permet d'obtenir l'heure de notre téléphone :

```

//time, h, m et s etant des Integer

time := GetCurrentTime;
h:=getHour(time);
m:=getMinute(time);
s:=getSecond(time);

```

Connaissant la position des aiguilles, nous les dessinons avec la procédure `drawLine()`. La couleur des aiguilles étant fixée avec la procédure `setColor(r : integer, g: integer, b:integer)`.

Le programme principal est très court, puisqu'il fait appel à l'unité `uHorloge` qui fait tout le gros travail. Le travail qui est confié est de charger l'image depuis le fichier `horloge.png`, d'appeler les fonctions définies dans `uHorloge` qui initialisent la position de l'horloge, et dans une boucle dessine les aiguilles de l'horloge.



```
program horloge;

uses uHorloge;

var img : Image;
    taille, cad : record
        x, y : integer;
    end;
    temps : integer;

begin
    //on charge l'image de notre cadran

    img:=LoadImage('/horloge.png');
    taille.x:=getImageWidth(img);
    taille.y:=getImageHeight(img);

    // on cherche a centrer l'horloge sur l'ecran

    cad.x:=(getWidth div 2)-(taille.x div 2);
    cad.y:=(getHeight div 2)-(taille.y div 2);

    uHorloge.init(cad.x+taille.x div 2, cad.y+taille.y div 2, taille.x div 2-20);

    repeat
        drawImage(img, cad.x, cad.y);
        uHorloge .dessiner;
        Repaint;
        delay(100);
        setColor(255, 255, 255);
        fillRect(0, 0, getWidth, getHeight);
    until getKeyPressed<>KE_NONE;

end.
```

La procédure fillRect dessine un rectangle rempli avec la couleur spécifiée dans setColor(), qui est la couleur blanche. De son côté, la procédure delay() fait dormir l'application suivant le nombre de milisecondes qui lui est passé en paramètre.

L'exécution du programme dans l'émulateur donne :



VIII - Conseils

Il est important de tenir compte des performances des applications que vous créez sous MIDlet Pascal. Parce qu'il s'agit d'applications destinés aux téléphones mobiles, il faut donc préférer les programmes optimisés. Il est donc souhaité de tester aussi bien votre application sur les émulateurs mais également directement sur des téléphones mobiles.

L'utilisation de données comme tableau (array), enregistrement (record) etc, ou encore l'utilisation de **repaint**, nécessitent de grandes ressources; nous vous conseillons de ne les utiliser qu'en cas de nécessité.

Dans la programmation conventionnelle, il est conseillé de préférer l'utilisation de variables locales aux variables globales, ce qui n'est pas le cas ici pour des raisons de performance.

En prenant comme exemples les deux exemples ci-dessous, vous remarquerez que le second, en terme de performance, est plus rapide:

Exemple 1

```
program projet_test;
begin
  drawText('Hello world une fois!', 0, 0);
  repaint;
  drawText('Hello world deux fois!', 20, 10);
  repaint;
  delay(2000);
end.
```

Exemple 2

```
program projet_test;
```



```
begin
drawText('Hello world une fois!', 0, 0);
drawText('Hello world deux fois!', 20, 10);
repaint;
delay(2000);
end.
```

Notez que dans l'exemple 1, le premier **repaint** n'est pas nécessaire et utilise des ressources pour rien.

IX - Conclusion

MIDlet Pascal est un environnement de développement et un compilateur qui utilise le langage Pascal pour produire du binaire Java. Aucune connaissance du J2ME n'est nécessaire puisque MIDlet Pascal gère tout de manière transparente. Il permet de faire des programmes utilisant le graphisme ou les formulaires. Plusieurs fonctions et types intégrés permettent l'envoi des sms, la lecture de la musique et le support du protocole http.

Puissant et simple d'utilisation, MIDlet Pascal est donc un outil pour débutant et professionnel aidant à produire du binaire Java sans s'embrouiller avec les détails du J2ME.

Nous remercions **Alcatiz** pour sa relecture et ses conseils.