Format jps : l'environnement 'picture'

par Jean-Paul Vignault
Groupe des Utilisateurs de Linux Poitevins (GULP)
(**jpv@melusine.eu.org**)
26 Janvier 2005

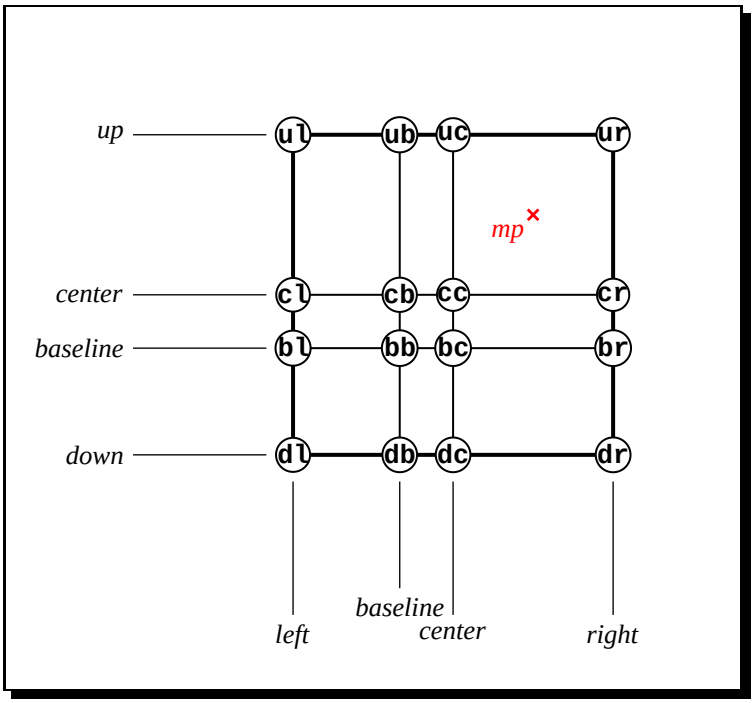
1. L'environnement 'picture'

Le but de cet environnement est de proposer un ensemble de commandes standardisées pour le positionnement des objets dans le repère *jps*. C'est en particulier lui qui gère le positionnement du texte ou des labels TEX.

L'utilisateur peut définir ses propres objets et utiliser toutes les facilités de placement ou d'encadrement proposées par l'environnement.

1.1 - Les points de référence

Chaque objet de l'environnement 'picture' est contenu dans un cadre rectangulaire : sa BoundingBox. Ce cadre définit 2 autres lignes : les médianes du rectangle. Pour finir, on considère la *baseline* sur chaque axe. Au total 4 lignes horizontales et 4 lignes verticales; leurs intersections nous donnent les 16 points de référence de l'objet considéré.



En plus de ces 16 points, vous voyez apparaître un point particulier sur le schéma ci-dessus. Il s'agit d'un point *spécial*, qui portera le nom **/mp** (*mon point*) pour les exemples qui vont suivre. Un objet peut comporter aucun ou plusieurs points spéciaux, et l'utilisateur peut en ajouter à des objets existants.

Si l'objet considéré est un texte, la baseline verticale sera souvent confondue avec le bord gauche du cadre (ligne *left*).

1.2 - Les commandes de positionnement

Les 17 commandes de positionnement se terminent toutes par **pict**, seul le préfixe change.

Les 4 préfixes *bb*, *bc*, *cb* et *cc* fonctionnent de la même façon et désignent le point de référence. Ainsi la commande

```
A (mon_objet) bbpict
```

signifie « dessiner l'objet *mon_objet* de telle sorte que son point *bb* soit exactement au point *A* ».

Les 12 préfixes standards qui restent désignent non plus les points de références, mais des directions. Ainsi la commande

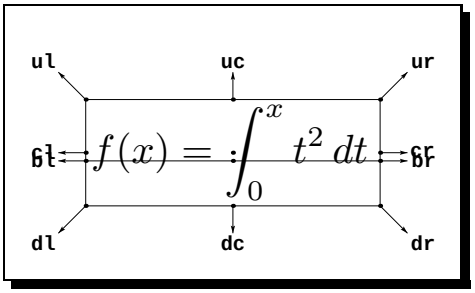
```
A (mon_objet) ucpict
```

signifie « dessiner l'objet *mon_objet* dans la direction *uc* par rapport au point *A* ». La différence par rapport à précédemment, c'est que le format a tendance à rajouter un petit déplacement dans la direction adéquate. Ainsi,

l'exemple précédent aura pour effet de placer le point *dc* de l'objet exactement au point *A*, puis d'ajouter un décalage vertical avant de tracer l'objet demandé.

Les composantes du décalage sont stockées dans les variables *had just* (décalage horizontal) et *vad just*. Elles sont initialisées à 3, 75 par défaut et représentent des dimensions en points postscript.

La figure ci-dessous montre les directions de déplacement en fonction des préfixes, dans le cas particulier où l'objet est un label TEX.



Pour finir la commande **spict** permet de placer un point spécial. Ainsi la commande

```
A (mon_objet) /mp spict
```

signifie « dessiner l'objet *mon_objet* de telle sorte que son point *mp* soit exactement au point *A* ».

Si le point spécial n'est pas nommé, on peut utiliser la syntaxe

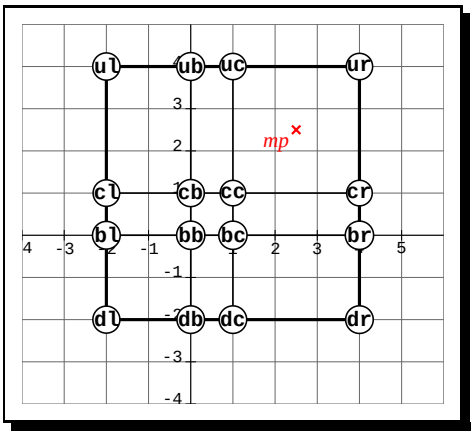
```
A (mon_objet) x y spict
```

où (x, y) représente les coordonnées du point spécial dans un repère *jps* dont le point *bb* de l'objet serait l'origine.

Ainsi, lorsque l'on reprend l'objet précédent, et qu'on le trace dans un repère gradué avec la commande **0 0 (mon_objet) bbpict**, on voit que le point *mp* a pour coordonnées $(2,5; 2,5)$. Les commandes

```
A (mon_objet) /mp spict
```
et
```
A (mon_objet) 2.5 2.5 spict
```

sont alors équivalentes.



1.3 - Les options

Chacune des commandes précédentes permet en option de faire subir à l'objet un décalage supplémentaire, un agrandissement/réduction et une rotation.

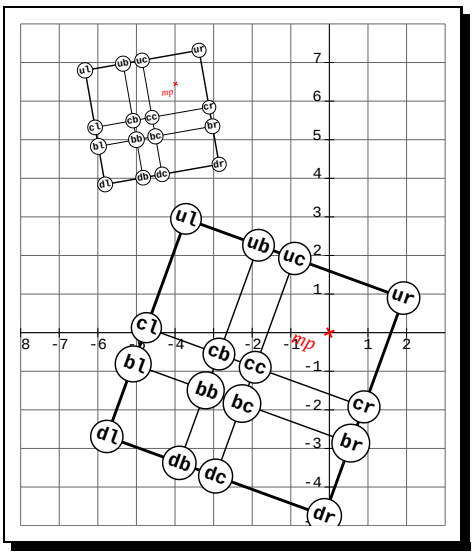
La syntaxe générale est

A (*dx dy*) [$scale_x$ $scale_y$] {α} **(mon_objet) bbpict**

où (dx, dy) est le déplacement en points postscript à rajouter au déplacement initial, $(scale_x, scale_y)$ est le facteur d'agrandissement, et α l'angle de la rotation autour du point *A*. Il est également possible de donner la chaîne vide **()** pour le déplacement. Dans ce cas, le format annule tout déplacement; par exemple, la commande **A () (mon_objet) urpict** va placer le point *dl* de *mon_objet exactement* au point *A*.

Par exemple, la figure ci-dessous est obtenue avec les commandes

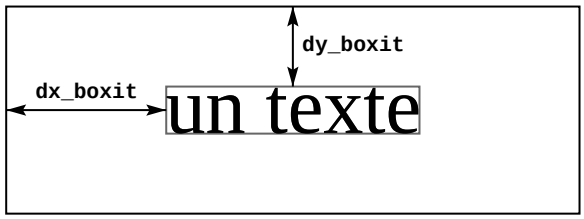
```
-5 5 [.5 dup] {10} (objet) bbpict
```
et
```
O {-20} (objet) /mp spict
```



1.4 - Mises en boites ou en cercles

Tous les objets affichés avec la famille de commandes **`urpict`**, **`ucpict`**, etc. . . peuvent être encadrés ou encerclés de différentes manières.

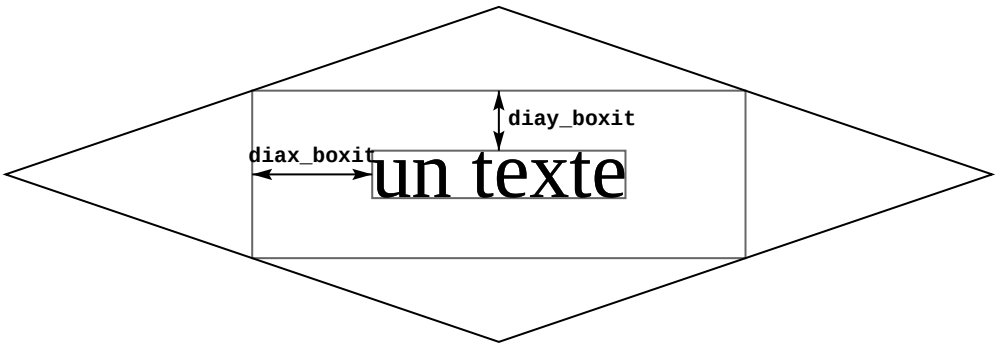
La commande **`boxit`** provoquera l'encadrement par un rectangle du prochain objet affiché. L'espace laissé entre le cadre et la boîte contenant l'objet est géré par les variables *dx_boxit* et *dy_boxit* comme l'indique le dessin ci-dessous :



La commande **`circleit`** tracera un cercle passant par les 4 sommets du rectangle défini ci-dessus. L'espace laissé entre le cercle et la boite contenant l'objet est donc là encore géré par les variables **`dx_boxit`** et **`dy_boxit`** (dimensions en points postscript) initialisées à 0 par défaut.

La commande **`Circleit`** tracera un cercle de centre le centre de la boite contenant l'objet, et de rayon le contenu de la variable *Circleradius*. Cette commande permet ainsi d'encercler divers objets avec des cercles d'un rayon fixé.

La commande **`diaboxit`** provoquera l'encadrement par un losange du prochain objet affiché. L'espace laissé entre le cadre et la boîte contenant l'objet est géré par les variables *diax_boxit* et *diay_boxit* comme l'indique le dessin ci-dessous :



Le losange est le losange d'aire minimale contenant la boite rectangulaire définie ci-dessus.

La commande **`ovalit`** tracera un ovale (une ellipse) passant par les 4 sommets du losange défini ci-dessus. L'espace laissé entre la boite incluse dans l'ellipse et la boite contenant l'objet est donc là encore géré par les variables **`diax_boxit`** et **`diay_boxit`** (dimensions en points postscript) initialisées à 0 par défaut.

La commande **boxit_all** (resp. **circleit_all**, **Circleit_all**, **diaboxit_all**, **ovalit_all**) provoquera l'encadrement de tous les objets suivants, et elle sera annulée par la commande **boxit_none** (resp. **circleit_none**, **Circleit_none**, **diaboxit_none**, **ovalit_none**).

Pour ces encadrements, le format utilise les versions étoilées des commandes **cercle**, **ellipse** et **frame**, ce qui autorise l'utilisation de **fillstyle** pour des cadres non transparents.

1.5 - Pour aller plus loin

L'environnement 'picture' dispose d'un dictionnaire spécifique (le dictionnaire *Pictdict*), dans lequel il stocke les coordonnées des différents points de référence de l'objet affiché. Ces dernières sont réactualisées à chaque nouvel affichage.

Chaque objet affiché comporte au moins les 16 points de référence classiques, nommés *ulpictpoint*, *ubpictpoint*, etc. . . (tous les noms sont suffixés par *pictpoint*, seul le préfixe change).

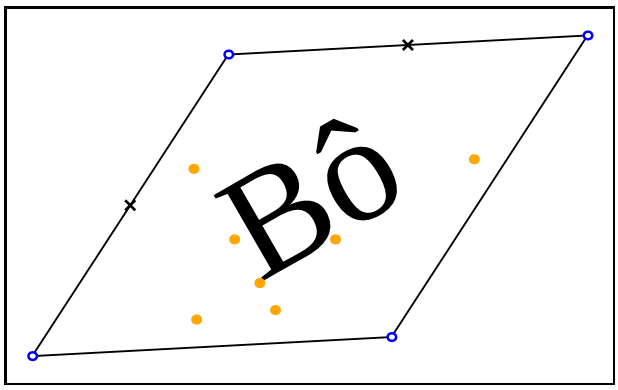
Si l'objet a été encadré par un losange ou une ellipse (une « *diabox* » ou une « *ovalbox* »), alors il y a 4 points de référence supplémentaires *udiapictpoint*, *ddiapictpoint*, *rdiapictpoint* et *ldiapictpoint* qui correspondent aux quatre sommets du losange ou de l'ellipse considérée (les 16 autres points de référence correspondant à ceux de la boite rectangulaire incluse dansle losange ou l'ellipse).

La commande **pictget** permet d'accéder à ces points. Sa syntaxe est la suivante

> *name* **pictdict** *x y* $\longrightarrow$ dépose sur la pile les coordonnées associées au nom *name* dans le dictionnaire *Pictdict*

le fichier jps

```
autocrop
setTimes
/diax_boxit 5 def
/diay_boxit 2 def
diaboxit
(Bô) 0 0 [6 dup] {30} cctext

/ulpictpoint pictget times2 %% croix en ul
/urpictpoint pictget times2 %% croix en ur
bleu
[  %% ronds sur les sommets du losange
   /rdiapictpoint /ldiapictpoint
   /udiapictpoint /ddiapictpoint
] {pictget circ} apply
orange
/dotscale {.8 dup}
[  %% points sur les 2 baseline
   /blpictpoint /bbpictpoint /bcpictpoint /brpictpoint
   /ubpictpoint /dbpictpoint /cbpictpoint
] {pictget dot} apply
```



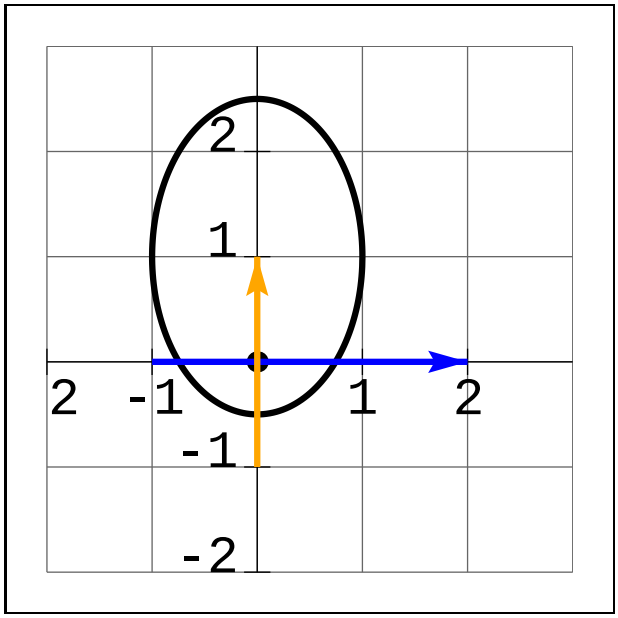
1.6 - Mettre un objet dans l'environnement 'picture'

1.6.1 - Création d'un nouvel objet

On commence par faire un beau dessin que l'on veut réutiliser. Pour l'exemple, on le trace avec des coordonnées absolues dans le repère jps. Ici le dessin sera constitué d'une ellipse, de deux flèches (une bleue et une orange), et un point.

le fichier jps

```
-2 3 setxrange
-2 3 setyrange
20 setxunit
quadrillage
traceaxes
marks

1.2 setlinewidth

%% on definit le dessin de notre nouvel objet
0 0 point
0 1 1 1.5 ellipse
bleu
[-1 0 2 0] (->) ligne
orange
[0 -1 0 1 ] (->) ligne
```



Une fois l'objet dessiné, on relève les coordonnées, dans le repère jps, de sa Bounding Box associée. Dans notre exemple, on obtient les points $(-1;-1)$ et $(2;2,5)$

1.6.2 - Enregistrement d'un nouvel objet

On associe alors la procédure de dessin de l'objet à un littéral (**`/mon_dessin`**) dans notre exemple non sans avoir encapsulé le tout dans un **`gsave`**. . . **`grestore`**, et après avoir rajouté l'incantation **`currentpoint translate`** qui permet de tracer le dessin au point courant (l'environnement picture assure l'existence du point courant au moment de l'appel).

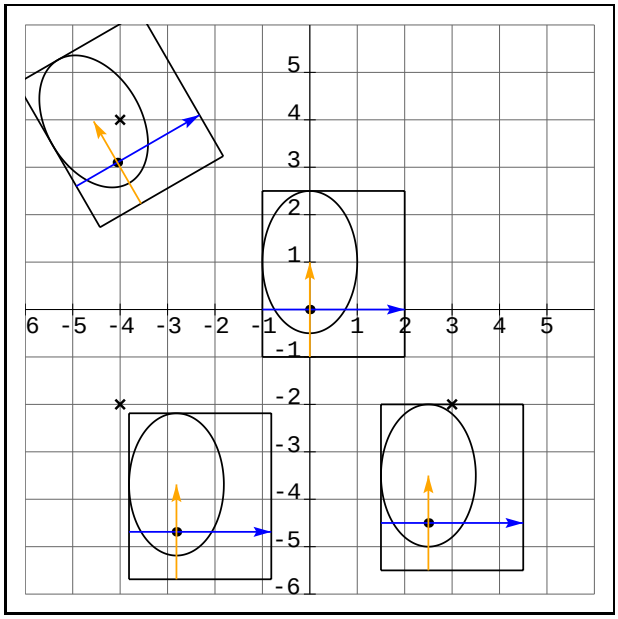
Ne reste plus qu'à créer une procédure dont le nom est celle du dessin du nouvel objet, suffixé par **`_dim`** (ce qui donnera **`mon_dessin_dim`** dans notre exemple), procédure qui est censee donner la BB du nouvel objet lorsqu'on l'appelle. Nota : cette BB doit être donnée dans le repère postscript. Si on ne la connaît qu'en coordonnées jps, il faut utiliser la fonction **`jtoppoint`** qui se chargera du changement de repere.

le fichier jps

```
20 setxunit
quadrillage
traceaxes
marks

%% on definit le dessin de notre nouvel objet
/mon_dessin {
gsave
   currentpoint translate      %% incantation a rajouter pour pouvoir
                               %% utiliser cette procedure dans
                               %% l'environnement 'picture'
```

le fichier jps (suite)

```
   %% Maintenant le dessin
   0 0 point
   0 1 1 1.5 ellipse
   bleu
   [-1 0 2 0] (->) ligne
   orange
   [0 -1 0 1 ] (->) ligne
grestore
} def

%% puis la procedure donnant ses dimensions
/mon_dessin_dim {
   %% les dimensions, dans le repere ps, de la BB du dessin;
   -1 -1 jtoppoint
   2 2.5 jtoppoint
} def

%% on encadre les dessins produits, pour mieux voir la Bounding Box
boxit_all
%% puis on place les dessins
0 0 (mon_dessin) bbpict                 %% point bb au point (0, 0)

-4 4 [1 dup] {30} (mon_dessin) ccpict   %% point cc au point (-4, 4)
                                        %% echelle (1, 1)
                                        %% rotation : 30 degre
-4 4 times2                             %% une croix pour bien voir (-4, 4)

-4 -2 (mon_dessin) drpict               %% dans la direction down/right
                                        %% par rapport au point (-4, -2)
                                        %% d'ou un leger decalage
                                        %% ajoute par le format
-4 -2 times2                            %% une croix pour bien voir (-4, -2)

3 -2 () (mon_dessin) dcpict             %% dans la direction down/center
                                        %% par rapport au point (3, -2)
                                        %% mais le () supprime le decalage
                                        %% ajoute par le format
3 -2 times2                             %% une croix pour bien voir (3, -2)
```



1.7 - Points spéciaux

Chaque objet peut disposer de son propre dictionnaire, dont le nom est celui de l'objet, suffixé par **_dic**. Ainsi, le dictionnaire éventuel associé à l'objet **mon_objet** sera **mon_objet_dic**. L'environnement 'picture' utilise lui le dictionnaire *Pictdic*. À chaque appel, on nettoie le dictionnaire *Pictdic*, puis on y copie le dictionnaire de l'objet. On fait ensuite subir à tous les points qui y sont définis les mêmes transformations qu'à l'objet.

Si le dictionnaire de l'objet n'existe pas, la commande

```
/mon_objet_dic 2 dict def
```

en créera un en lui réservant la place mémoire pour 2 points spéciaux.

À partir du moment où le dictionnaire est crée, la commande

```
mon_objet_dic /bb {0 0} put
```

définira alors le point *bb* de coordonnées $(0, 0)$ dans le repère lié à l'objet mon_objet.

Lors de l'appel au dessin d'un objet, l'environnement 'picture' fait donc subir à tous les points spéciaux les mêmes transformations qu'à l'objet, ce qui permet de les retrouver plus tard. Ainsi, si l'on demande le dessin de mon_objet, la commande

```
/bb pictget
```

déposera sur la pile les coordonnées du point *bb* **lors du dernier appel de l'environnement 'picture'**. Si l'on souhaite utiliser ces coordonnées plus tard, il faut les sauvegarder, par exemple par une commande du type

```
/bb pictget /bb defpoint
```

qui a pour effet de sauvegarder ces coordonnées sous le nom *bb* dans le dictionnaire courant.

Si on souhaite conserver tous les points spéciaux, la commande

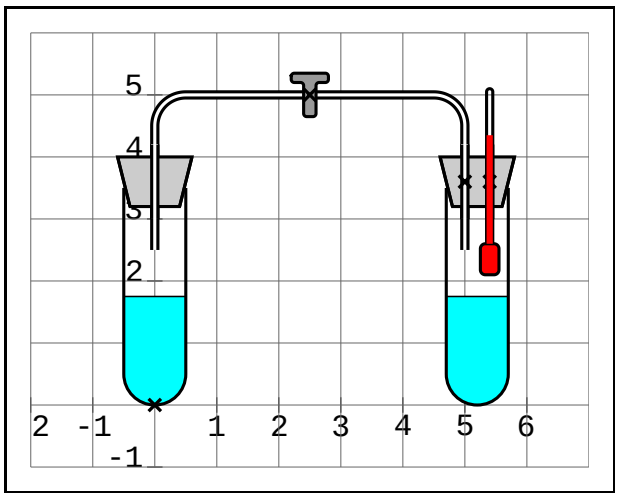
```
Pictdic currentdict copy
```

copiera l'intégralité du dictionnaire *Pictdic* dans le dictionnaire courant.

Ci-dessous un exemple d'application où tout le montage de chimie est lié par les points spéciaux.

le fichier jps

```
uselabo
20 setxunit
-2 7  setxrange
-1 6 setyrange
1 setlinewidth

1 setlinewidth
marks
quadrillage

/aspectLiquides [{cyan}] def
/niveauLiquides [50] def

%% on dessine le 1er tube a essais avec bouchon et tubeU
withbouchon
/Tubes [(TubeU)] def
O (TubeEssais) bbpict

O times2                                %% une croix sur O
/uc pictget /A defpoint                 %% on recupere le point /uc du tubeU et on le nomme A
/ConnectOut pictget times2              %% une croix sur /ConnectOut
                                               %% on dessine le 2eme tube a essais avec
/Tubes [(relax) dup] def                       %% un bouchon a 2 trous mais pas de tube
/ConnectOut pictget (TubeEssais) /trou1 spict  %% de telle facon que son point /trou1
                                               %% soit au point /ConnectOut du tubeU
/trou1 pictget times2                   %% une croix sur /trou1
/trou2 pictget times2                   %% une croix sur /trou2
/trou2 pictget (Thermometre) ccpict     %% on dessine le centre du Thermometre au point
                                        %% /trou2 du tube a essais

A (RobinetTube) ccpict                  %% puis le robinet au point /uc du tubeU,
                                        %% prealablement sauvegarde en A
A times2                                %% une croix sur A
```


5
4
3
2
2 -1 1 2 3 4 5 6
-1