# AVL CONCERTO™ 2015

## Scripting

Lycée VAUBAN - Presentation – 17-12-2014
AVL France

# SOMMAIRE

## Programme de la Formation

# VOIES ET DATASETS

## Codage sous Concerto

- Les voies sélectionnées et chargées dans la mémoire du PC deviennent des Datasets.

- Les Datasets sont :

  - des objets multi formats : valeurs numériques, chaînes de caractères;
  - des objets disposants de méthodes;
  - des « vecteurs » multi colonnes (enregistrement avec base implicite).

-Les Datasets sont nommés de la manière suivante :

**FichierAlias : Clés'NomVoie**.

**Exemples d'appel de données :**

PUMA1:D'SPEED
IFILE1:CA'PCYL1

# LES TROIS FORMATS

## Les Différents Types de Fichiers de Codage

| ORGANISATION DES FORMULES – MACROS - SCRIPTS | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **TYPES** | **BUT** | **EXTENSION** | **ARGUMENTS E/S** | | **POSSIBILITES D'APPEL** | | | **MOTS CLE OBLIGATOIRES** | **MOTS CLE FACULTATIFS** |
| | | | **E** | **S** | **FRM** | **MAC** | **CSF** | | |
| FORMULES | Génération des voies calculées | .FRM | **x** | **O** | **O** | **O** | **x** | VarCalc=…..<br>**Return** VarCalc | |
| MACROS | Modification des affichages ou de valeurs numériques | .MAC | **O** | **O** | **O** | **O** | **X quand appelée par une formule**<br>**O quand appelée par un script** | **Arg** Var1, Var2,…<br>VarCalc=Var1+….<br>**Return** VarCalc | |
| SCRIPTS | Automatisation de CONCERTO | .CSF | **Libre** | **Libre** | **O** | **O** | **O** | **x** | **Arg** Var1, Var2,…<br>VarCalc=Var1+….<br>**Return** VarCalc |

# LES DIFFERENCES ENTRE LES FORMATS

## Synthèse

Un **script** dispose de tous les droits de calcul et de manipulation de CONCERTO et de son environnement.

Une **formule** n'est utilisée qu'au sein d'un groupe de fichiers et ne supporte aucune fonction de script.

Une **macro** est toujours appelée par :

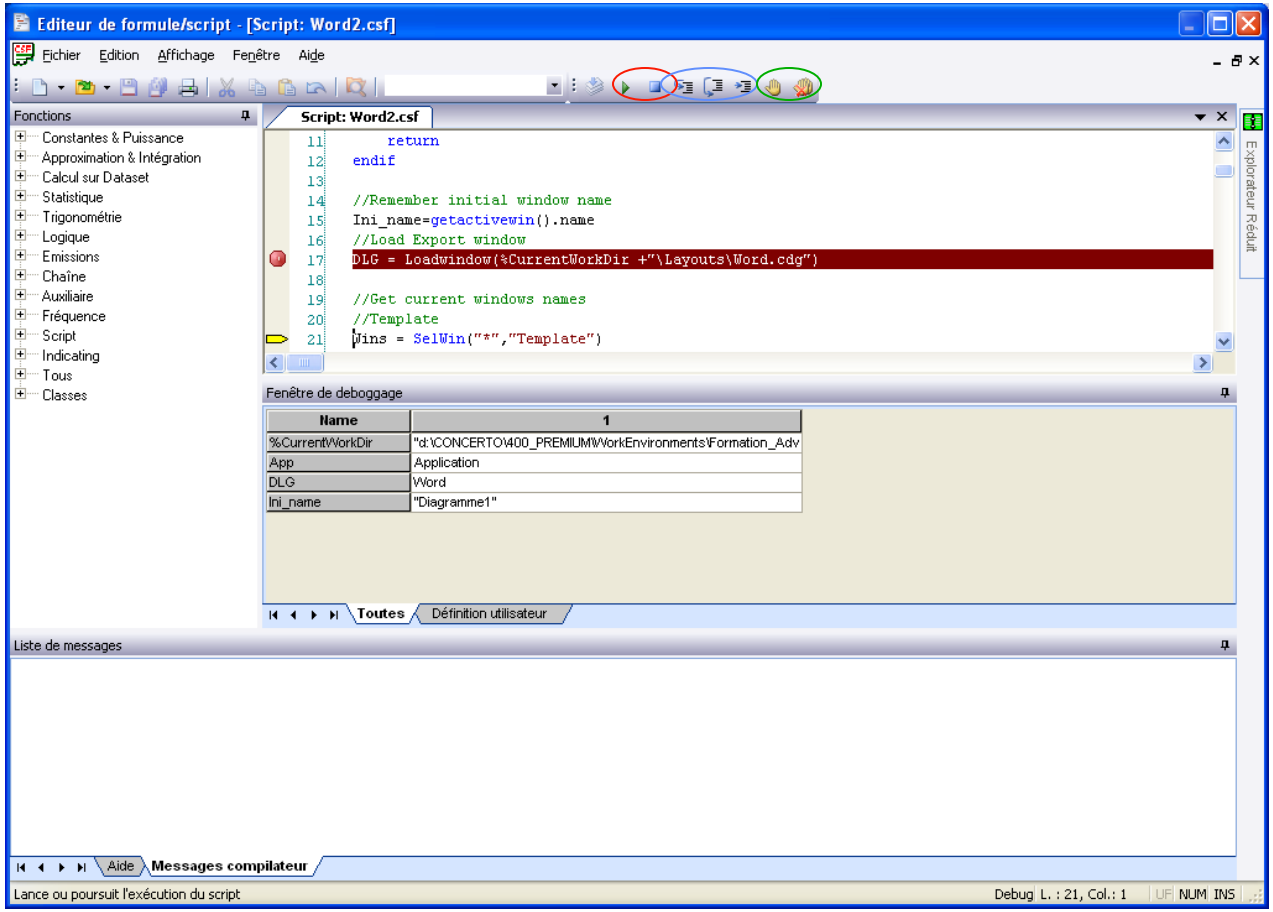
- Un **script** : alors elle hérite des droits du script et peut appeler d'autres scripts ou contenir des fonctions de script;

- Une **formule** : alors elle hérite des droits de la formules et ne peut pas appeler de script ou contenir de fonctions de script;

- Un **onglet de transformation** d'un axe : alors elle se comporte comme une macro appelée par une formule.

# LE DEBUGGER

## Le Codage sous Concerto




Lancer l'exécution

Arrêter l'exécution

Step Into (Pas à Pas)

Step Over

Run to Cursor

Point d'arrêt

# RAPPELS DE PROGRAMMATION

## Le Codage sous Concerto

Si l'expression de *condition* est vraie, les instructions1 sont exécutées. Dans le cas contraire, les instructions2 sont exécutées. Le mot clé else ainsi que les instructions2 ne sont pas obligatoires.

Conditions multiples possibles : elseif (condition2) then

**Tests:**

```
if (condition) then
        instructions1
else
        instructions2
endif
```

**Boucles:**

```
for compteur = Val_Ini to Val_Fin [step Pas]
        instructions
next compteur
```

Toutes les instructions entre for et next sont exécutées à chaque « tour ». Au début de la boucle, *counter* vaut *Val_Ini,* puis augmente de *Pas* à chaque tour jusqu'à *Val_Fin.*
*Pas* peut être positif comme négatif.

Toutes les instructions entre while et endwhile sont exécutées du moment que la condition est remplie.
Attention aux boucles sans fin => touche Echap.

**Tant que:**

```
while condition
        instructions
endwhile
```

**Interruption d'exécution:**

```
break
```

Cette instruction met un terme à une boucle for ou while. Elle s'utilise principalement en association avec une instruction "if...then".
Dans les boucles emboîtées, break utilisée seule permet de retourner à la boucle supérieure suivante.

# FORMULES ET MACROS : REGLES GENERALES

## Le Codage sous Concerto

Dans Concerto, les **formules** ne possèdent pas d'arguments. Elles travaillent directement avec les voies de mesure. Elles se terminent toujours par « *return expression* ». Par la suite, une formule s'utilise comme une voie de mesure.

Dans Concerto, les macros possèdent des arguments: la première ligne est toujours « arg var1, var2 … ». Elles permettent la modification d'affichage de diagramme (macro de transformation) ou la modification de valeurs numériques (sortes de sous programmes). Elles se terminent toujours par « return expression ».

# EXERCICES RELATIFS AUX FORMULES

## Enoncés

**Exercice1 : Retour du maximum d'une voie**

Dans l'éditeur de formules/scripts, créez une nouvelle formule. Celle-ci devra retourner le maximum de la voie D'N pour les essais ATFFILE.

**Exercice2 : Modification d'une voie**

On cherche à modifier la voie D'N pour les données ATFFILE, selon une condition:

- Si la valeur est supérieure ou égale à 2000, alors, elle doit être imposée à 0;
- Si la valeur est inférieure à 2000, elle n'est pas modifiée.

Le dataset D'N devra d'abord être sélectionné, puis, pour chacun de ses points, un test sera effectué sur la valeur de la voie: si elle est supérieure à 2000, alors elle sera modifiée, sinon elle sera conservée.

# EXERCICES BONUS RELATIFS AUX FORMULES

## Enoncés

**Renvoi d'un tableau:**

Créez un tableau de datasets de dimension (1,2). Dans la première case, placez la voie D'N et dans la seconde, D'MD .

Créez d'abord un tableau de datasets de dimensions adaptées, puis cherchez dans la classe « tableau de dataset » les fonctions permettant le traitement que vous souhaiter.

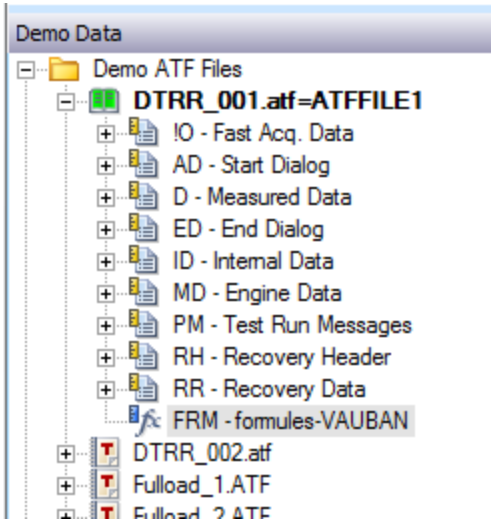
Si vous ajoutez un « # » au nom de la formule, vous pourrez lire les deux datasets séparément.

Dans la formule précédente, imposez les noms des datasets: « Test1 » et « Test2 ».

Dans une autre formule, récupérez le tableau et tentez d'extraire le dataset « Test1 ».

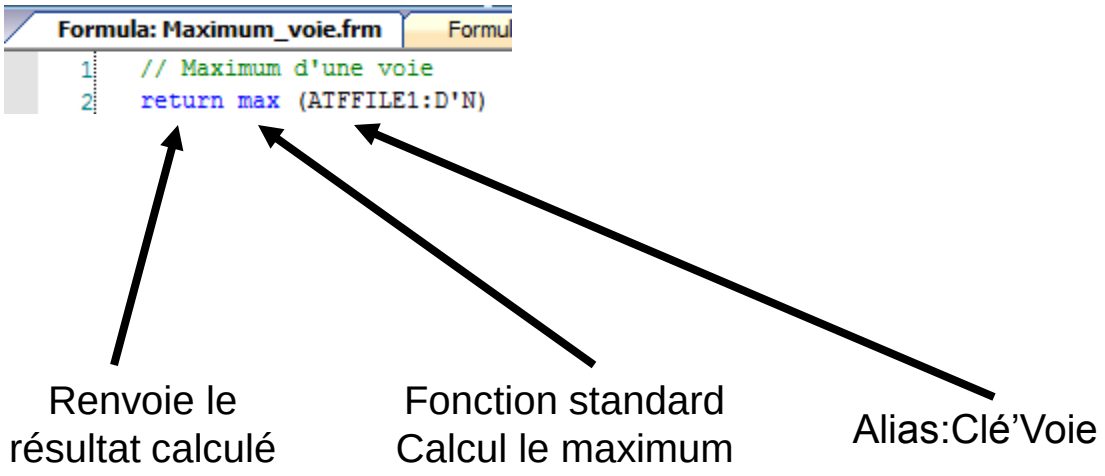
# EXERCICES RELATIFS AUX FORMULES

## Corrigés




**Exercice1 : Retour du maximum d'une voie**

Depuis l'explorateur de données, créez une nouvelle formule en faisant un click droit au niveau de la clé FRM. Vous pouvez alors taper le code de cette formule.




Après sauvegarde, la formule Maximum_Voie est visible dans l'explorateur de données.

# EXERCICES RELATIFS AUX FORMULES

## Corrigés

### Exercice2 : Modification d'une voie

Formule: Maximum_Voie.frm | **Formule: Modification_Voie.frm**

```
1   // Modification d'une voie
2   Voie = FL'P
3
4   for i=1 to Voie.count
5       if Voie.y[i]>=50 then
6           Voie.y[i]=80
7       endif
8   next i
9
10  return Voie
```

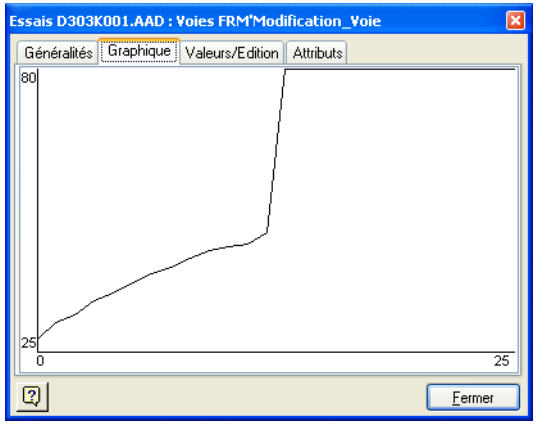
Sélection de la voie

On boucle sur tous les éléments de la voie. Voie.count permet de connaitre le nombre d'éléments du dataset Voie

Test sur la i-ème valeur de Voie

Modification de la valeur

Retour du dataset modifié



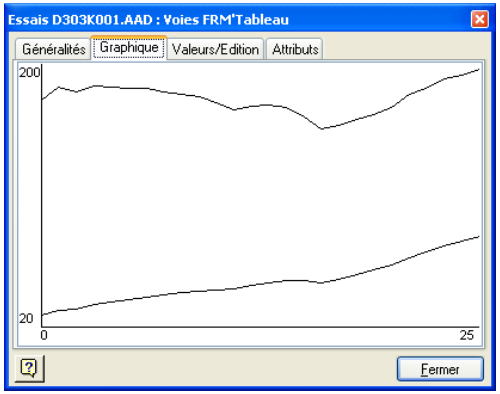
# EXERCICES BONUS RELATIFS AUX FORMULES

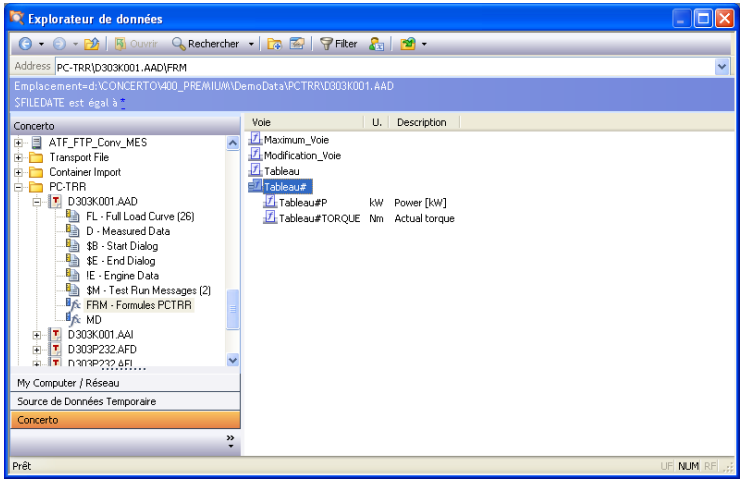
## Corrigés

### Renvoi d'un tableau

Formule: Maximum_Voie.frm | Formule: Modification_Voie.frm | **Formule: Tableau.frm**

```
1  // Renvoi d'un tableau
2  Tableau = newdsarray(1,2)
3  Tableau.putcell(FL'P,1,1)
4  Tableau.putcell(FL'TORQUE,1,2)
5  return Tableau
```




En sauvegardant le nom de la formule avec un « # », les deux datasets peuvent être lus séparément:

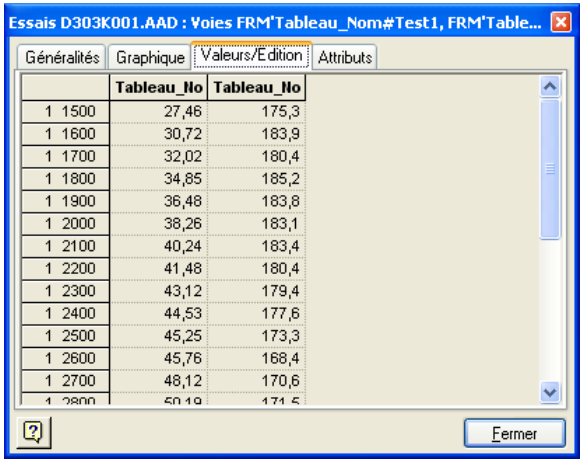
# EXERCICES BONUS RELATIFS AUX FORMULES

## Corrigés (suite)

**Formule: Tableau_Nom#.frm** | Formule: Maxi

```
1  // Renvoi d'un tableau
2  Tableau = newdsarray(1,2)
3  dataset1 = FL'P
4  dataset2 = FL'TORQUE
5  dataset1.name = "Test1"
6  dataset2.name = "Test2"
7  Tableau.putcell(dataset1,1,1)
8  Tableau.putcell(dataset2,1,2)
9  return Tableau
```

=>




Deux méthodes pour le dernier exemple: extraire directement le n-ième dataset du tableau, ou récupérer le tableau puis extraire le dataset:

**Formule: Extraction_Methode1.frm** | Formule: Tableau_Nom#.frm

```
1  // renvoie d'un dataset extrait d'un tableau
2  return Tableau_Nom#Test1
```

**Formule: Extraction_Methode2.frm** | Formule: Extraction_Method

```
1  // Renvoie d'un dataset extrait d'un tableau
2  Tableau = Tableau_Noms#
3  dataset = Tableau.getcell(1,1)
4  return dataset
```

# EXERCICES RELATIFS AUX MACROS

## Enoncés

**Exercice1 : Lissage d'une voie**

Créez tout d'abord un diagramme représentant les voies D'BLOWBY du groupe ATFFILE.
Créez alors une nouvelle macro qui utilisera la fonction de lissage de la bibliothèque.

La macro doit récupérer en argument la voie sur laquelle elle s'applique. Il suffit ensuite de retourner cette voie après « passage » dans la fonction de lissage.

Les macros applicables aux axes sont définissables dans la page propriétés de l'axe concerné en tant que **Transformation**.

Modifiez ensuite la macro pour pouvoir opérer des lissages successifs.

Une méthode simple consiste à passer le nombre de lissages en second argument, et d'effectuer une boucle de 1 à n contenant la fonction de lissage.

N'oubliez cependant pas de spécifier le second argument … à moins qu'on ne puisse définir un nombre de lissages par défaut.

# EXERCICES RELATIFS AUX MACROS

## Enoncés

**Exercice2 : Décalage d'une voie**

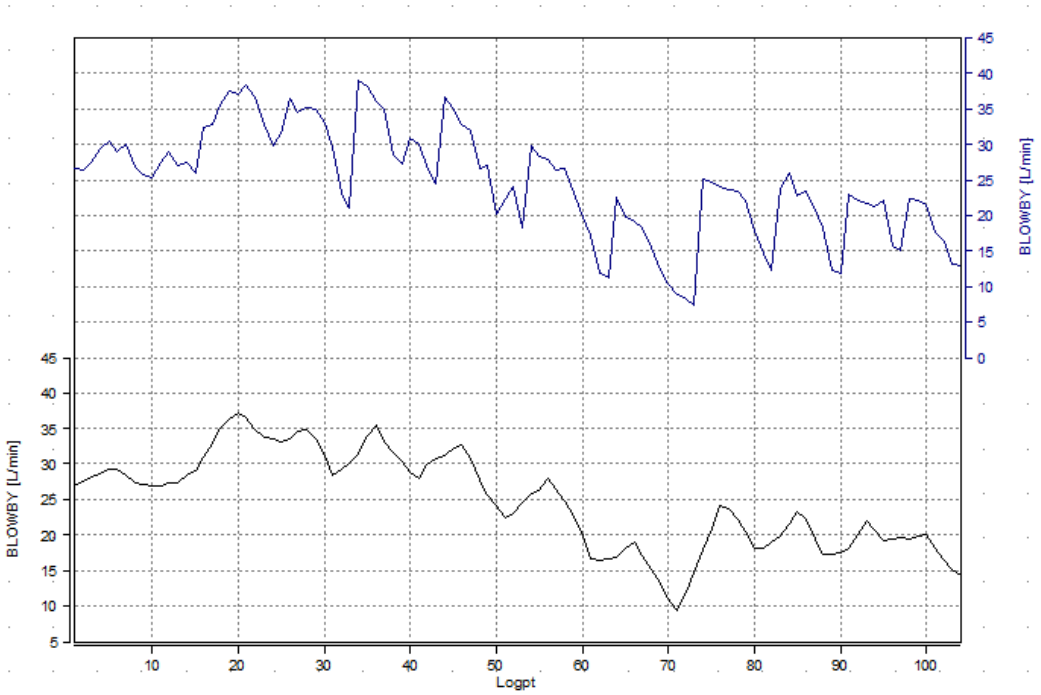
Pour le même diagramme que précédemment, créez une macro qui récupérera deux arguments, l'un servant de décalage (offset) pour les abscisses, et l'autre de décalage pour les ordonnées.

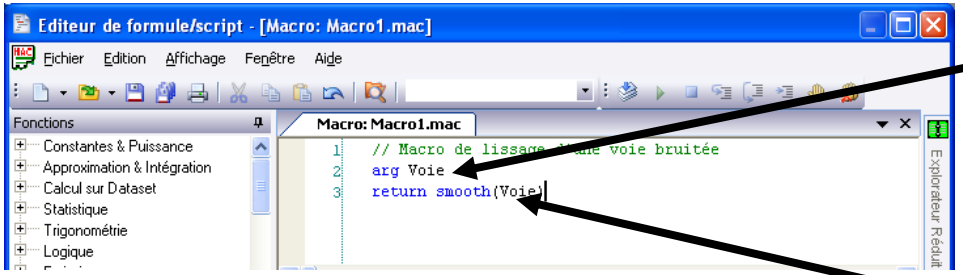
En cherchant bien, vous devriez trouver une fonction qui permet de créer un dataset à partir de deux autres datasets, l'un servant de voie de base et l'autre de valeurs en y.

Pourquoi ne pas récupérer la voie de base ainsi que la voie en y du dataset à modifier, de leur ajouter le décalage respectif pour fabriquer ensuite un nouveau dataset ?

# EXERCICES RELATIFS AUX MACROS

## Corrigés de l'Exercice 1 : Lissage d'une voie




Passage de la voie en argument: C'est le diagramme qui transmet la voie à la macro.

Fonction de lissage et retour de la voie lissée

# EXERCICES RELATIFS AUX MACROS

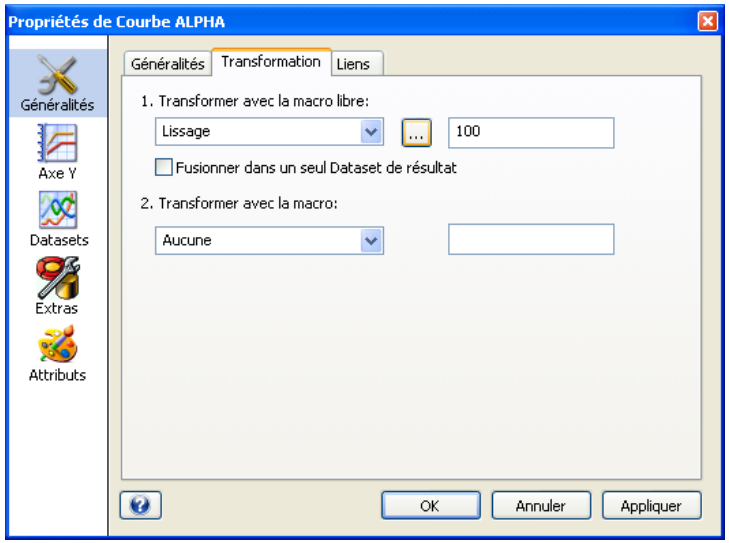
## Corrigés : Lissage multiple

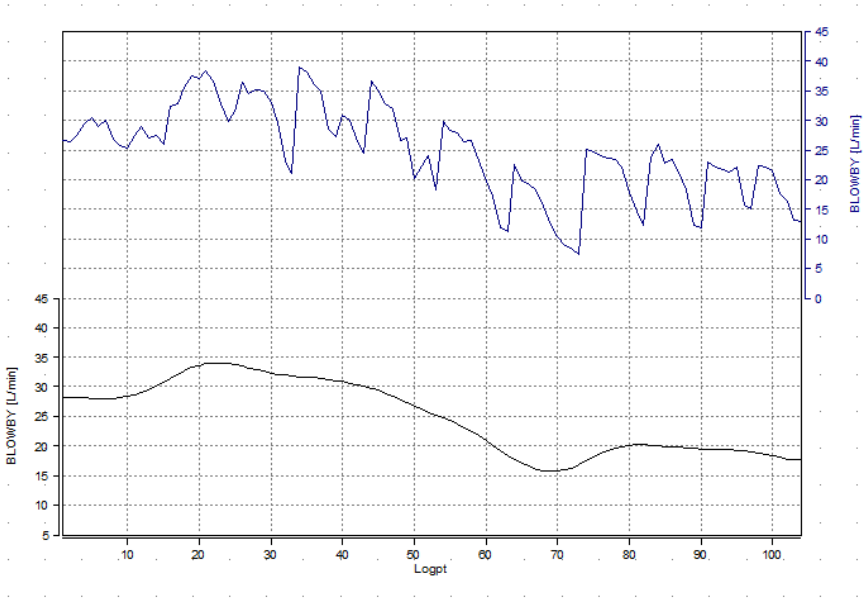
Macro: Lissage.mac

```
1  // Macro de lissage d'une voie bruitée
2  arg Voie, n=0
3
4  for i=1 to n
5      Voie = smooth(Voie)
6  next i
7
8  return Voie
```

n récupère la valeur définie dans le champ texte de l'onglet de transformation.

0 est une valeur par défaut, qui n'est utilisée que s'il n'y a rien de spécifié dans le champ texte (100 ci-dessous).



# EXERCICES RELATIFS AUX MACROS

## Corrigé de l'Exercice 2 : Décalage d'une Voie

Macro: Decalage.mac

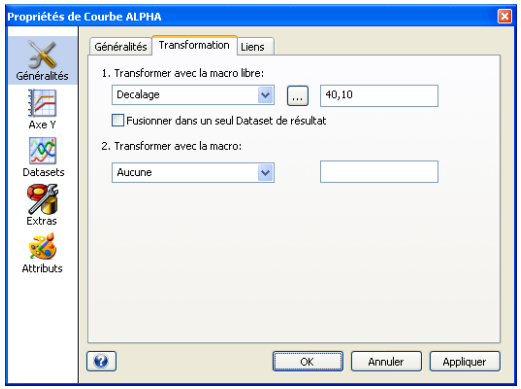
```
1  // Décalage en X et en Y
2  arg Voie, Dec_x=0, Dec_y=0
3
4  Px = xds(Voie)
5  Py = Voie
6
7  X_mod = Px + Dec_x
8  Y_mod = Py + Dec_y
9
10 Voie_Décalée = create(X_mod,Y_mod)
11
12 return Voie_Décalée
```

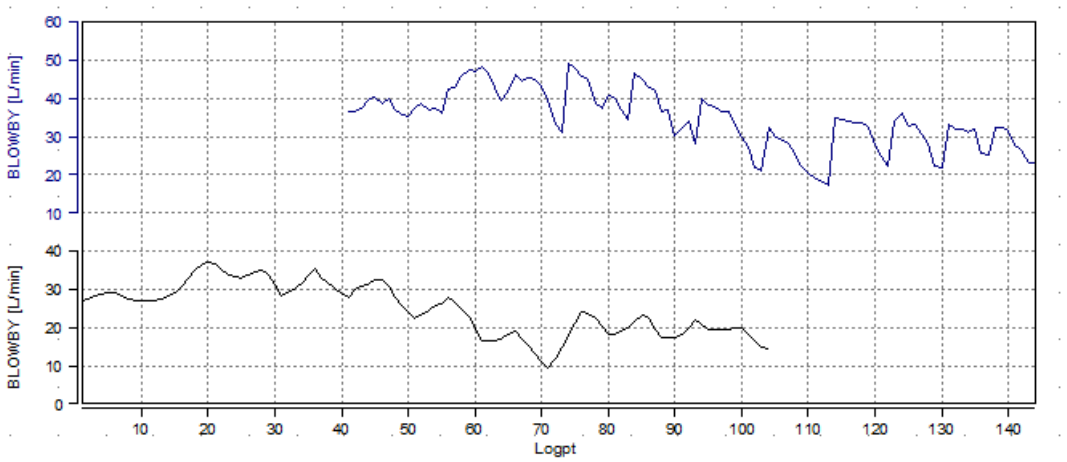
Récupération des arguments,

Séparation partie en x et y,

Ajout des décalages respectifs,

Création d'un nouveau dataset.



# APPEL D'UNE MACRO

## Utilisation Simple des Macros

Macro: Decalage.mac | **Macro: Affine.mac**

```
1  arg VALEUR1, VALEUR2
2
3  RESULTAT = VALEUR1 + VALEUR2
4
5  return RESULTAT
```

Pour appeler une macro depuis un autre fichier (formule, macro ou script), il suffit de connaître le nombre d'arguments dont a besoin cette macro.

Macro: Decalage.mac | Macro: A

```
1  A=3
2  B=7
3  CALCUL=Affine(A,B)
4  debug(CALCUL)
```

Fenêtre de deboggage

| Name | 1 |
|---|---|
| A | 3 |
| B | 7 |
| CALCUL | 10 |

Le script a appelé la macro Affine.mac en lui passant A et B en arguments.

Affine a retourné le résultat du calcul à partir de ces valeurs:

# CREATION D'UNE MACRO CALCGRAF

## Codage Avancé sous Concerto

Définition de l'entête de la macro => Apparence de la boîte Calcgraf

*//_comp*
*// _srcx…*
*// _defnames …..*

Définition des arguments

*arg _src1, _src2, …, _srcx, _comp, _param1 …*

Définition du corps de la macro

Définition des sorties

*if _comp=0 then …*
*if _comp=1 then …*

# CREATION D'UNE MACRO CALCGRAF

## Ecriture de l'entête

Nous allons fabriquer notre propre macro Calcgraf. Celle-ci apparaîtra dans la bibliothèque et pourra être utilisée dans un modèle comme n'importe quelle macro déjà existante.

Cette macro aura deux entrées. Sur la première, une intégration ou une dérivation pourra être effectuée, et l'ordre de l'intégration ou de la dérivation devra être spécifié. Une inversion de la voie placée en seconde entrée pourra être effectuée.

Les premières lignes qui définissent votre macro Calcgraf sont toujours les mêmes. Elles se présentent comme des lignes de commentaire commençant par un double slash mais elles ne doivent en aucun cas être supprimées. Il s'agit de la définition de votre "boîte":

```
//_comment = Traitement divers
//_comp = Alg | RES(A),RES2(A)
//_src1 = Signal|Signal 1
//_src2 = Signal|Signal 2
//_defnames = Sortie 1,Sortie 2
//_D = Ordre Dérivation | 0,1,2
//_Inv = Inversion du signal | Non, Oui
//_I = Ordre Integration | 0,1,2
```

La première ligne est un simple commentaire décrivant brièvement votre macro. La seconde défini votre ou vos sorties, et doit donc toujours commencer par //_comp. Le type de données est spécifié entre parenthèses.

Ici A signifie que c'est une voie basée sur des degrés vilebrequin. Bien entendu, d'autres possibilités existent, comme les voies basées sur le temps (T). Les différentes possibilités sont les suivantes:

# CREATION D'UNE MACRO CALCGRAF

## Ecriture de l'entête

A : voie basée sur les angles vilebrequin (Angle based curves),
T : la voie de base est temporelle (time based curves),
C : voie cyclique (cyclic result curves),
V : voie non cyclique (non cyclic result value),
U : format utilisateur
M : sans base réelle ( ex : une fréquence pour une FFT)
Y : voie purement cyclique.

Les lignes suivantes définissent les entrées, s'il y en a plusieurs. Une seule peut être bien entendu suffisante, en fonction du traitement que vous désirez effectuer. Ces lignes commencent toujours par //_srcX; X étant le numéro de l'entrée. Vous pouvez définir par *Commentaire \ Nom* le nom qui sera indiqué en entrée dans la boîte.

*//_comment = Traitement divers*
*//_comp = Alg | RES(A),RES2(A)*
*//_src1 = Signal|Signal 1*
*//_src2 = Signal|Signal 2*
*//_defnames = Sortie 1,Sortie 2*
*//_D = Ordre Dérivation | 0,1,2*
*//_Inv = Inversion du signal | Non, Oui*
*//_I = Ordre Integration | 0,1,2*

//_defnames est le texte de sortie qui apparaît dans votre boîte.

Enfin, les dernières lignes définissent ce que vous verrez dans la fenêtre propriétés de votre boîte, à savoir le texte ainsi que les différentes propositions de choix. Pour l'ordre de la dérivation, les choix possibles sont 0, 1 ou 2; comme pour l'intégration d'ailleurs. En revanche, pour l'inversion, l'utilisateur pourra opter pour oui ou non.

# CREATION D'UNE MACRO CALCGRAF

## Ligne d'Arguments

La ligne suivante de code est comme suit :

*arg _src1, _src2, _comp, _I = 0 , _D = 0, _Inv = 0*

L'ordre à respecter est le suivant: tout d'abord il faut passer en arguments les entrées (_src1, _src2, _src3, etc …), puis la sortie ainsi que les variables. Vous pouvez remarquer qu'ici des valeurs par défaut ont été fixées: pour _Inv, 0, qui correspond à "Non" (1 aurait correspondu à Oui); et de la même façon, les valeurs par défaut pour l'ordre d'intégration ou de dérivation ont été données à 0.

**ATTENTION :**
**0 ne correspond pas à la valeur numérique donnée par défaut mais à l'index de la valeur** dans la liste de choix possible.

Par exemple, pour l'intégration, les choix étaient 0, 1 et 2. L'index de la première valeur est toujours 0, puis 1, 2 et ainsi de suite.

Il reste ensuite à définir le code lié au traitement :

# CREATION D'UNE MACRO CALCGRAF

## Nouvelle Version de Concerto

```
// Integration
if _I <> 0 then
        for i = 1 to _I
                _src1 = integrate(_src1)
        next i
endif
// Dérivation
if _D <> 0 then
        _src1 = derivate(_src1,_D)
endif
// Inv
if _Inv = 1 then
        _src2 =  (-1) * _src2
endif
```

Enfin, les sorties doivent être correctement assignées. Un test sur la valeur de _comp doit être effectué avant de retourner la variable souhaitée. A la première sortie est associée la valeur 0, tandis que la deuxième sortie est associée à la valeur 1 (puis 2 pour la troisième sortie, et ainsi de suite …).

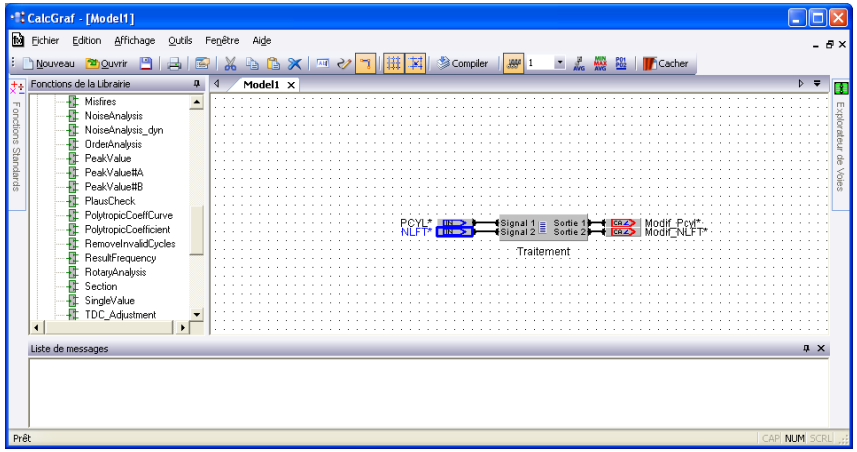
```
if _comp = 0 then return _src1
if _comp = 1 then return _src2
```

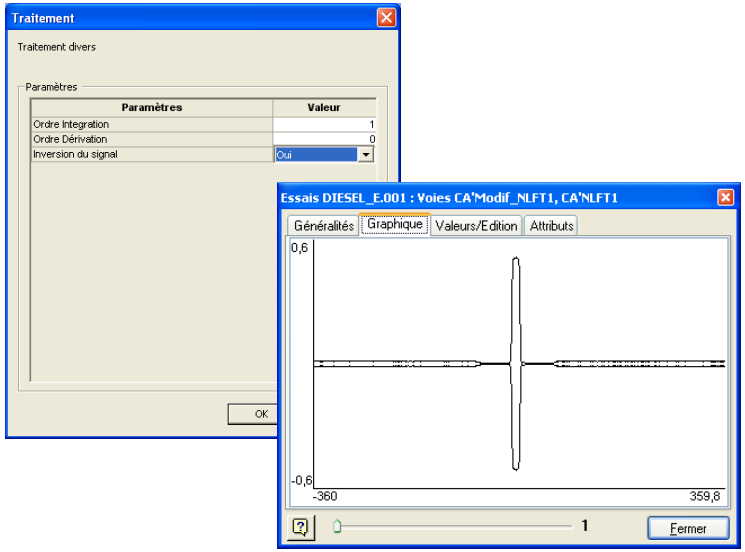
Dans cet exemple, la première sortie retournera _src1 tandis que la seconde retournera _src2.

Vous pouvez désormais sauvegarder votre macro avec les autres macros de Calcgraf (dans un des répertoires spécifique Indimacro), ici sous le nom que vous désirer, et l'utiliser dans un modèle:

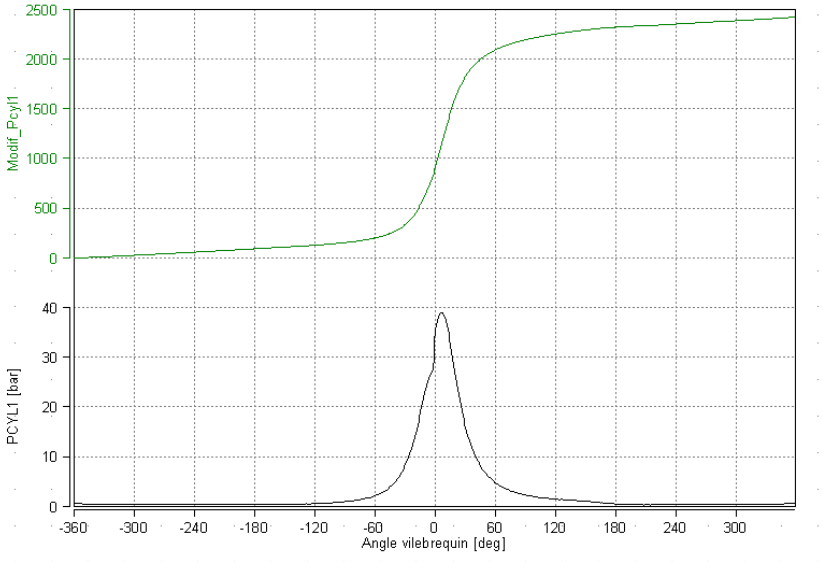
# CREATION D'UNE MACRO CALCGRAF

## Résultats




Créations du modèle;
Définition des paramètres de calculs;
Visualisation des formules obtenues.



# LES SCRIPTS

## Le Codage Sous Concerto

Script: Ouverture.csf

```
1  // ##############################################################################
2  // OUVERTURE DE FICHIER
3  // ##############################################################################
4
5  LoadLayout(%CurrentWorkDir *+ "\Layouts\Tutoriall.cly",1)
6  LoadfileBrowser("PC-TRR")
```

Ouverture d'un dossier
Sélection des données à afficher

Script: Ouverture.csf | Script: Script3.csf

```
1  // ##############################################################################
2  // FERMETURE DE FICHIER
3  // ##############################################################################
4
5  MonFichier = Filename("PCTRR1")
6  Fichier = selfile("PC-TRR\" + MonFichier)
7  Fichier.close
8
9  // Fermeture de toutes les fenêtres
10 NomFenetre = selwin()
11 NomFenetre.close
```

Récupération du nom de l'essai ouvert sous l'alias PCTRR1;
Fermeture de cet essai

Sélection des fenêtres
Fermeture de celles-ci

**A vous de jouer:**
A partir de ces deux exemples, créer un script qui échangera les données d'un dossier existant. Vous n'aurez pas à toucher au dossier, mais juste récupérer les données ouvertes, les fermer et permettre à l'utilisateur d'en choisir d'autres.

# LES SCRIPTS

## Correction

Macro Function: VAUBAN3.mac | **Script: Changement_fichier.csf**

```
1   // ####################################
2   // SCRIPT : CHARGEMENT DES DONNEES
3   // ####################################
4
5   // Fermeture de l'ancien fichier
6   NomFichier = FileName(ATFFILE1:D'N)
7   Fichier_a_fermer = "Demo ATF Files\" + NomFichier
8   Fichier = SelFile(Fichier_a_fermer)
9   Fichier.Close()
10
11  // Chargement des nouvelles données
12  // Fichier qu iva s'ouvrir en alias1
13  loadfilebrowser("Demo ATF Files")
14  NomFichier = FileName(ATFFILE1:D'N)
15  Fichier_a_fermer = "Demo ATF Files\" + NomFichier
16  traceinfo("Le fichier" + Fichier + " a été ouvert")
17
18  // Rafraichissement de toutes les données
19  a = selwin()
20  a.AutoRange()
```

# EXERCICES BONUS RELATIFS AUX SCRIPTS

## Enoncés

**Exercice1 : Montrer tous les cycles**

Au préalable, vous devrez tracer un signal Indicating possédant plusieurs cycles.

Editez un script dont le but est de reconnaître les voies présentes dans un diagramme et d'en afficher tous les cycles existants.

Dans concerto, pour sélectionner le n-ième cycle d'une voie, il faut placer le numéro du cycle entre crochets (NomVoie[n]).

**Exercice2 : Changer les bornes d'un diagramme**

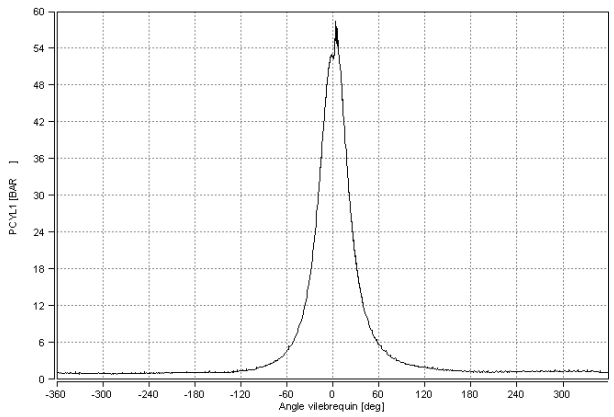
Une interface viendra vous demander de spécifier les nouvelles bornes à imposer pour l'abscisse de votre diagramme. Une vérification sur ces valeurs sera opérée (la borne min devant toujours être inférieure à la borne max).

# CORRIGES DES EXERCICES BONUS RELATIFS AUX SCRIPTS

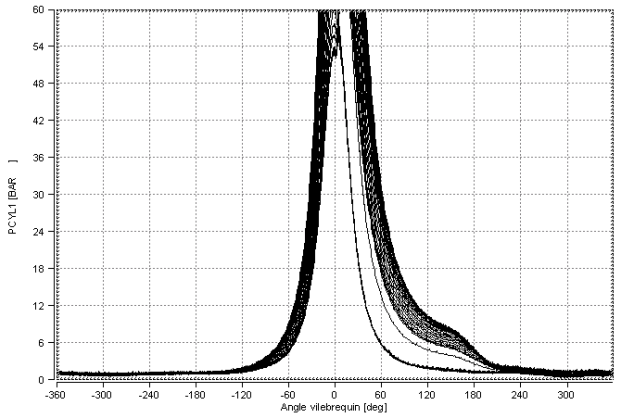
## Exercice 1 : Montrer tous les cycles

Script: Ouverture.csf | **Script: Montrer_Cycle.csf**

```
1   // ########################################################################
2   // SCRIPT : MONTRER TOUS LES CYCLES
3   // ########################################################################
4
5   Win = getactivewin()          // Sélectionne la fenêtre active
6   Obj = Win.selobj()            // Sélectionne tous les objets;
7                                 // Il ne doit y avoir q'un seul objet courbe
8
9   FirstDS = Obj.selds()
10
11  // Balayage de l'ensemble des cycles
12  for i=1 to Win.CycleCount
13      Obj.addds(FirstDS.name + "[" + i + "]")
14  next i
15
16  Sous_Obj = Obj.selobj()
17  Sous_Obj.Colour = FirstDS.Colour
```




=>



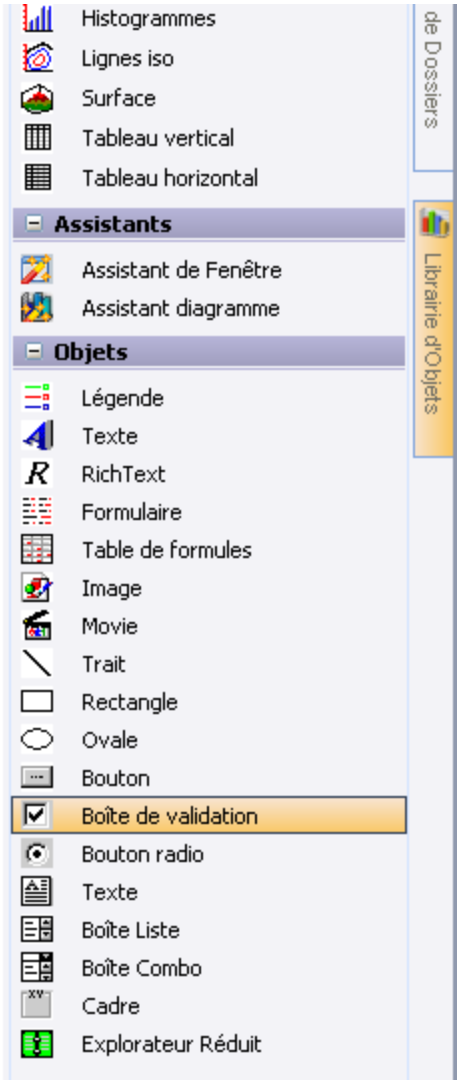
# CORRIGES DES EXERCICES BONUS RELATIFS AUX SCRIPTS

## Changer les Bornes d'un Diagramme

```
// ################################################
// SCRIPT : DEFINITION DE NOUVELLES BORNES
// ################################################

// Recherche des limites sur les angles vilebrequin
Inf_CA = min(Xds(IFILE1:CA'PCYL1))
Sup_CA = max(Xds(IFILE1:CA'PCYL1))

// Initialisation des valeurs
Min_CA = 0
Max_CA = 0

// Attribution des valeurs par défaut définissant la plus grande plage
SetUserVar("Min_CA",Cstr(Inf_CA))
SetUserVar("Max_CA",Cstr(Sup_CA))

// Contrôles sur la validité des bornes pour éviter des bornes abérrantes

    while (Min_CA < Inf_CA) or (Min_CA >= Max_CA) or (Max_CA > Sup_CA)
        Input("Bornes angle vilebrequin","Mini_CA","Minimum:","Maxi_CA","Maximum:")
        Max_CA = CReal(GetUserVar("Maxi_CA"))
        Min_CA = CReal(GetUserVar("Mini_CA"))

        if Max_CA > Sup_CA then
            message="Attention, la valeur maximum est de " + Cstr(Sup_CA) + "°CA"
            MsgBox(message)
            SetUserVar("Max_CA",Cstr(Sup_CA))
        endif

        if Inf_CA > Min_CA then
            message="Attention, la valeur minimum est de " + Cstr(Inf_CA) + "°CA"
            MsgBox(message)
            SetUserVar("Min_CA",Cstr(Inf_CA))
        endif

        if Min_CA > Max_CA then
            message="Attention, la limite mini doit être inférieure à la limite maxi"
            MsgBox(message)
            Min_Ca = Cstr(Inf_CA)
            Max_Ca = Cstr(Sup_CA)
        endif
    endwhile

// Adaptation du nouvel affichage
Win = Selwin()
Win.X0 = Min_CA
Win.X1 = Max_CA
```

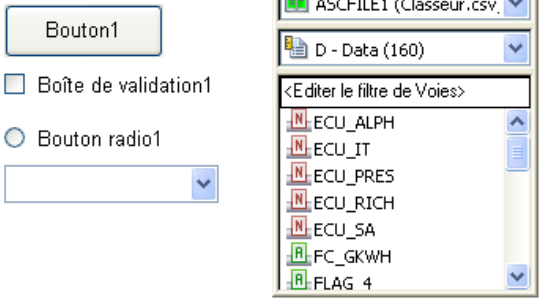
# CREER DES IHMs

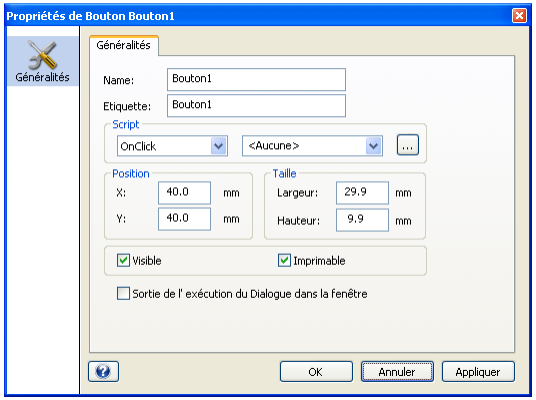
## Le Codage sous Concerto




Possibilité d'insérer dans une page de nombreux objets de contrôles.

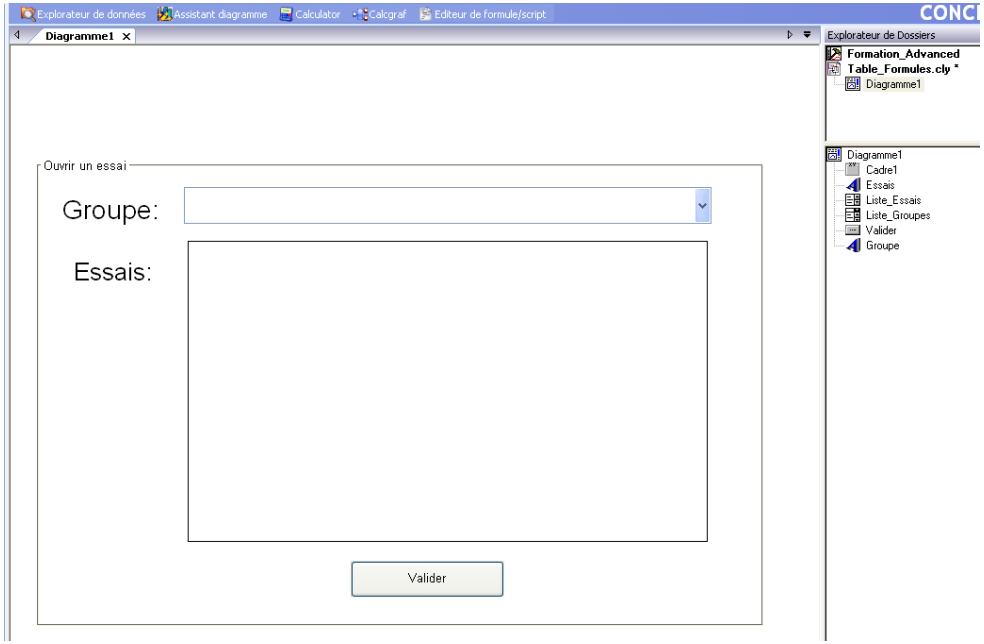
Pour chacun d'entre eux, une page propriétés est disponible.



# EXEMPLE DE TRAVAIL AVEC LES IHMs

## Interface et Scripts Associés




3 scripts associés :

- L'un pour ouvrir et initialiser l'interface;

- Le deuxième pour récupérer le groupe choisi par l'utilisateur et faire afficher les essais correspondants;

- Le dernier pour récupérer l'essai choisi, l'ouvrir et fermer l'interface.

# EXEMPLE DE TRAVAIL AVEC LES IHMs

## Script de Lancement (Manuel)

Script: Ouverture.csf | Script: Montrer_Cycle.csf | **Script: IHM.csf**

```
1  // ****************************************************************
2  // SCRIPT : TRAVAIL SUR LES IHM
3  // ****************************************************************
4
5  // Ouverture de la fenêtre
6  IHM = Loadwindow(%CurrentWorkDir + "\Layouts\Ouvrir un essai.cdi")
7
8  // Nommage de ses composants
9  Liste_Groupes = IHM.selobj("Liste_Groupes")
10 Liste_Essais = IHM.selobj("Liste_Essais")
11 Valider = IHM.selobj("Valider")
12
13 // Définition des scripts associés
14 Liste_Groupes.scriptname = %CurrentWorkDir + "\Scripts\IHM_Essais.csf"
15 Valider.scriptname = %CurrentWorkDir + "\Scripts\IHM_Validation.csf"
16
17 //Selection de l'application et de l'explorateur de données
18 APP = getapplication()
19 DATAEXPLOREUR = Getdataexplorer()
20 GROUPES = DATAEXPLOREUR.GetGroups()
21
22 //On place les noms de groupes récupérés dans la boîtes combo
23 Liste_Groupes.textds() = GROUPES
24
```

Ouverture de la planche

Sélection des éléments

Définition des scripts

Récupération des groupes présents dans l'explorateur de données

# EXEMPLE DE TRAVAIL AVEC LES IHMs

## Script lancé lorsque l'Utilisateur fait un choix dans la Liste de Groupes

Script: Ouverture.csf | Script: Montrer_Cycle.csf | Script: IHM.csf | **Script: IHM_Essais.csf**

```
1  // ****************************************************************
2  // SCRIPT : TRAVAIL SUR LES IHM - Actualisation des essais disponibles
3  // ****************************************************************
4
5  //Sélection de la fenêtre
6  IHM = getactivewin()
7
8  //Définition des composants
9  Liste_Groupes = IHM.selobj("Liste_Groupes")
10 Liste_Essais = IHM.selobj("Liste_Essais")
11
12 //Récupération du groupe sélectionné
13 GROUPES = Liste_Groupes.textds
14 INDEX_GROUPE = Liste_Groupes.getselectedpoints
15 GROUPE = GROUPES.y[INDEX_GROUPE]
16
17 //Mise à jour de la liste d'essai
18 APP = getapplication()
19 DATAEXPLOREUR = Getdataexplorer()
20 ESSAIS = DATAEXPLOREUR.GetFiles(GROUPE)
21 Liste_Essais.textds=ESSAIS
22
23 //Remise à jour de la fenêtre
24 IHM.paint
25
```

Sélection de la planche

Sélection des éléments

Récupération du groupe sélectionné

Récupération des essais présents dans ce groupe et mise à jour de l'interface

# EXEMPLE DE TRAVAIL AVEC LES IHMs

## Script lancé lorsque l'Utilisateur fait un choix dans la Liste d'Essais

Script: IHM_Validation.csf | Script: Ouverture.csf | Script: Montrer_Cycle.csf | Scrip

```
1  // ****************************************************************
2  // SCRIPT : TRAVAIL SUR LES IHM - Validation
3  // ****************************************************************
4
5  //Sélection de la fenêtre
6  IHM = getactivewin()
7
8  //Définition des composants
9  Liste_Groupes = IHM.selobj("Liste_Groupes")
10 Liste_Essais = IHM.selobj("Liste_Essais")
11
12 //Récupération de l'essai à ouvrir
13 ESSAIS = Liste_Essais.textds
14 INDEX_ESSAI = Liste_Essais.setselectedpoints
15 ESSAI = ESSAIS.y[INDEX_ESSAI]
16
17 //Fermeture de la fenetre
18 TraceInfo("L'essai " + ESSAI + " a été ouvert.")
19 IHM.close
20
21 // Ouverture du Fichier
22 FICHIER = SelFile(ESSAI)
23 FICHIER.open
```

Sélection des planches

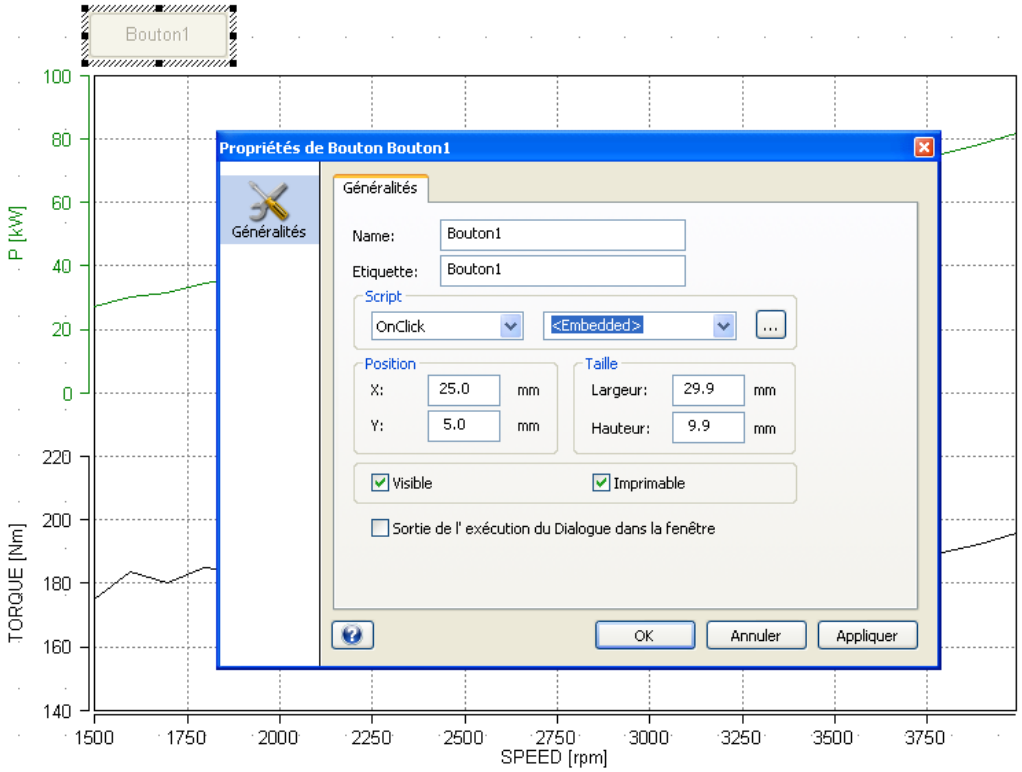
Sélection des éléments

Récupération de l'essai sélectionné

Ouverture de l'essai et fermeture de l'IHM

# LES SCRIPTS EMBEDDED

## Le Codage sous Concerto




Dans le menu contextuel des éléments de contrôle, vous pouvez accéder à l'édition d'un script embedded. Ce script est sauvegardé directement avec l'objet (aucun fichier csf n'est généré).

**Avantage** :
En sauvegardant la planche, vous sauvegardez aussi les actions effectuées par les éléments de contrôle (facilite les échanges).

**Désavantage** :
Debbogage plus difficile.

# SOMMAIRE

## Programme de la Formation

# IDEE GENERALE

## L’Environnement de Travail

Il se compose de:

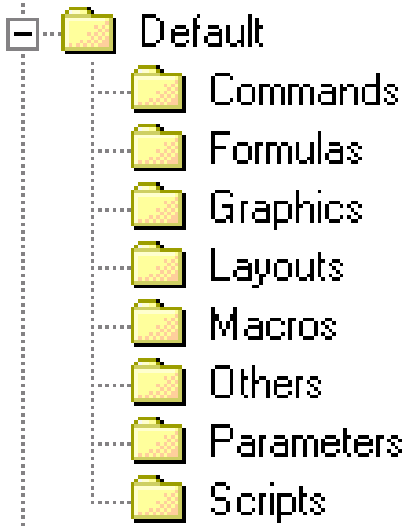
fenêtres d’affichage spécifiques (avec utilisation de formules/macros/scripts);
des adaptations des interfaces utilisateur (barre de commandes);
des scripts (avec démarrages de scripts inclus);
des fichiers spécifiques d’applications à votre convenance.

Fichiers sauvegardés avec l’extension **cwf** (Concerto Work File) qui est l’équivalent d’un fichier zip.

Vous pouvez basculer l’explorateur de dossier et la barre de commande soit en cliquant sur l’icône dédiée soit avec le **menu Affichage/Basculer les barres**.

# STRUCTURE

## L'Environnement de Travail




Après décompression, l'environnement de travail met en place l'architecture de répertoires suivante sous le répertoire **WorkEnvironments.**

Dans le répertoire Commands, chaque fichier (*.cly,*.ccf,*.cdi,*.cre,*.cpt,*.cdg,*.cwf) apparaîtra dans la barre de commande.

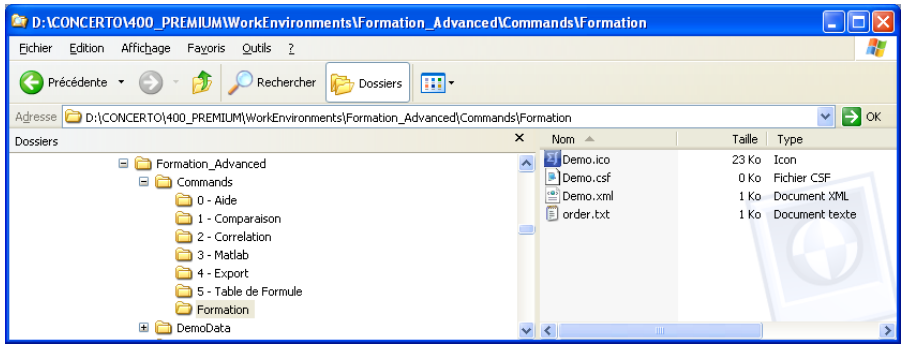
Possibilité d'assigner une icône personnalisée (bmp 32*32 dans le même repértoire + NomdeMonFichier32.bmp) .

1 sous répertoire = une nouvelle barre de commande

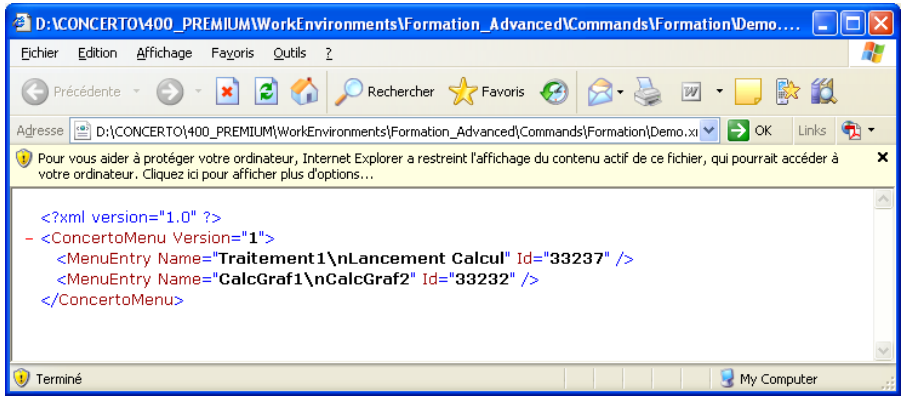
**%CurrentWorkDir** = le chemin jusqu'au nom de l'environnement de travail courant.

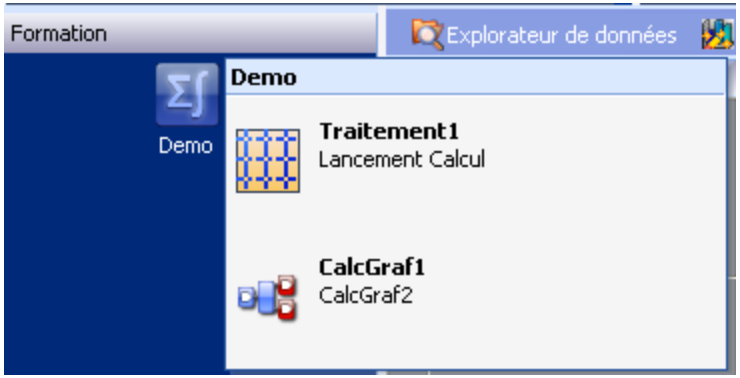
# LA BARRE DE COMMANDE

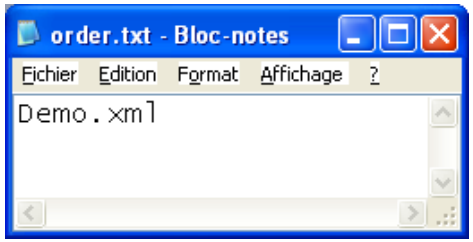
## Nouveauté 4.0




Utilisation de fichiers XML pour créer des sous menus

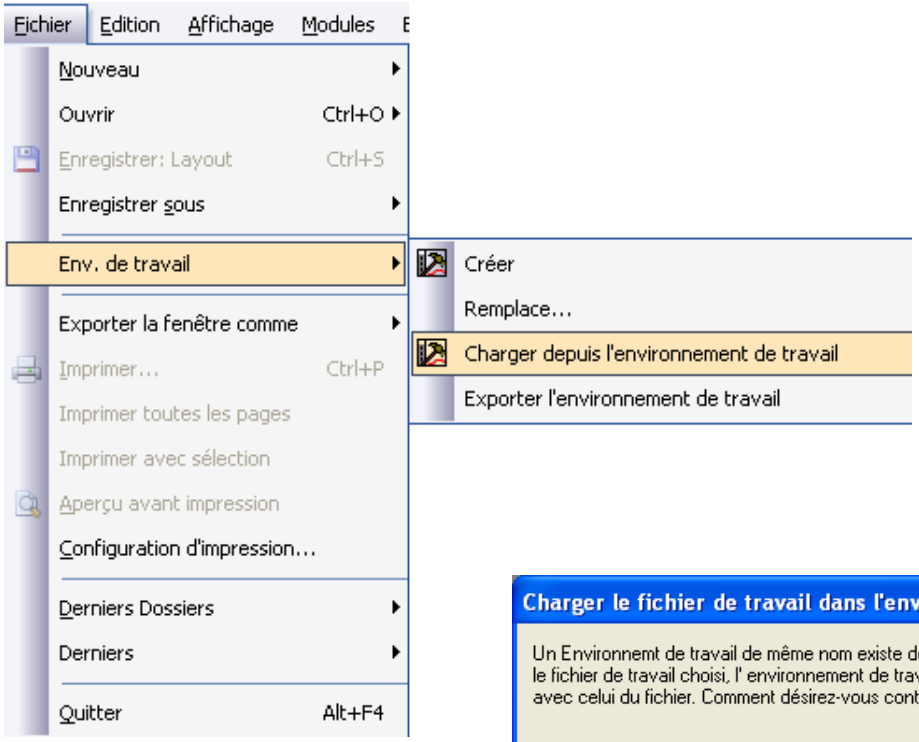


# OUVERTURE

## L'Environnement de Travail

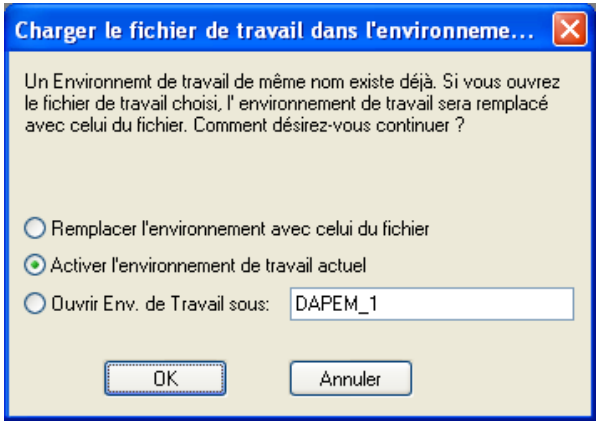



Si le nom existe déjà, 3 possibilités:

- Il doit être écrasé;

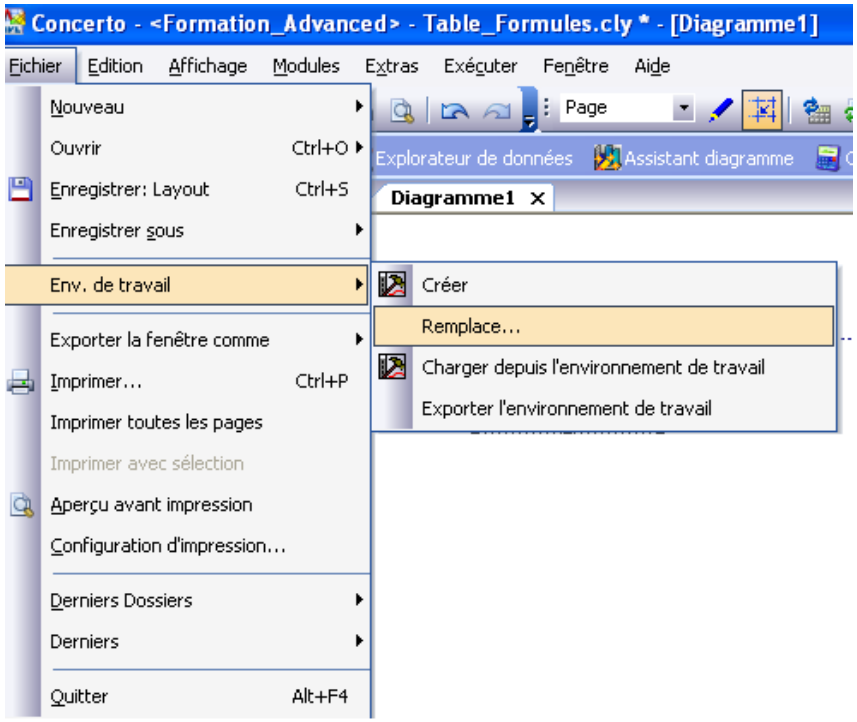
- Le Fichier de travail n'est pas ouvert et l'environnement de travail existant est activé à sa place;

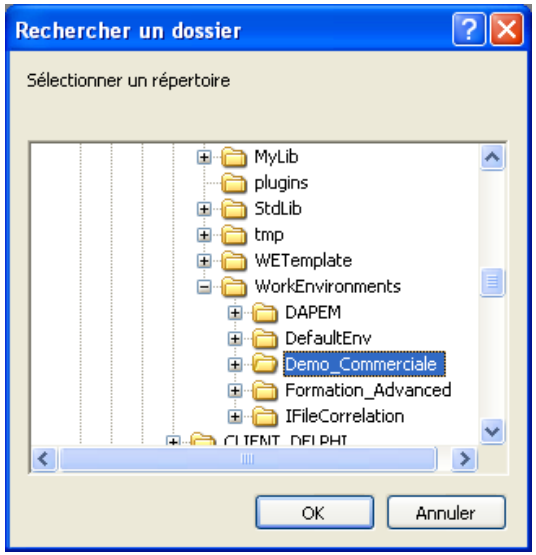
- Le fichier de travail est ouvert mais l'environnement de travail contenu est défini sous un autre nom.



# ACTIVATION

## L'Environnement de Travail



# SCRIPTS ASSOCIES

## L'Environnement de Travail

### Script de Démarrage

Automatiquement exécuté à l'ouverture d'un Environnement de travail (chargement d'un Fichier de travail ou activation d'un Environnement de travail antérieur).

⇒ STARTUP.CSF, à enregistrer dans le répertoire *Scripts* de l'Environnement de Travail

### Script d'alimentation

=> POWERUP.CSF dans le sous-répertoire *****Scripts* de l'Environnement de travail actuel, ce dernier est automatiquement exécuté au démarrage de **Concerto**.

### Script de fermeture

=> CLOSE.CSF dans le sous-répertoire *****Scripts* de l'Environnement de travail actuel, ce dernier est automatiquement appelé lorsque l'Environnement de travail est fermé, à savoir lorsque:

- un nouvel Environnement de travail est créé;
- un second Environnement de travail est chargé à partir d'un Fichier de travail;
- un ancien Environnement de travail existant est activé.

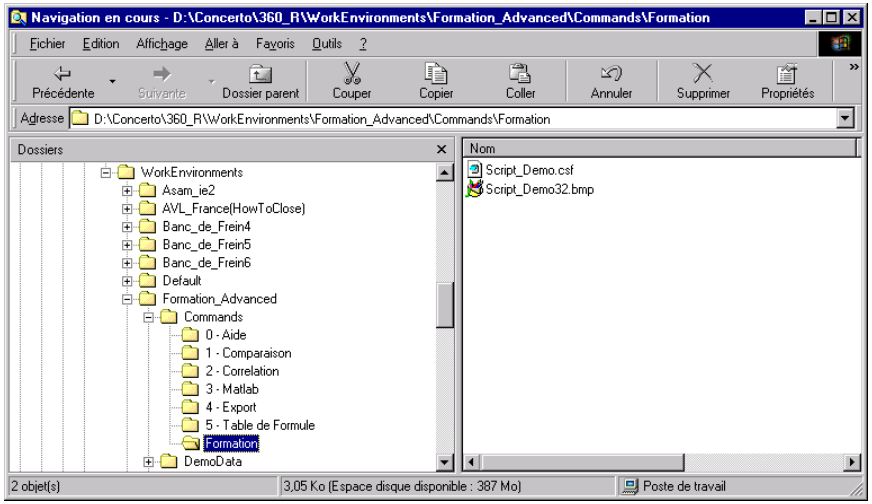
# EXERCICES

## L’Environnement de Travail


Ajoutez un onglet à votre barre de commande et créez un script à l’intérieur de celui-ci. Vous pourrez ensuite faire l’icône associé.

Pour ce faire, il vous suffit d’ajouter un répertoire dans l’environnement de travail sous le répertoire « Commands ».

Ajoutez ensuite un fichier script, puis créez une image bitmap de 32 par 32 pixels qui portera le même nom que votre script avec 32 en plus, avant l’extension .bmp.



Bravo, et merci de votre attention !

# Des Questions ?

Discussion