PREMIERS PAS AVEC L'INTELLISENSE

En tapant DEFC dans un programme, je veux que VFP saisisse le nom de la classe que je veux définir ainsi que le nom de la classe parent et écrive automatiquement :

```
DEFINE CLASS classe1 AS custom && jme 28/08/2008
  *


  PROCEDURE Init

  ENDPROC && Init

  PROCEDURE Destroy

  ENDPROC && Destroy

  PROCEDURE Error( nError, cMethod, nLine)

  ENDPROC && Error


ENDDEFINE && classe1
```

Avant, j'utilisais CEE (Cob Editor Expansions) mais l'intellisense de VFP9 peut le faire et assez simplement. Voici mes premiers pas avec cet outil.

Toutes les données utilisées par l'intellisense se trouve dans la table foxcode.dbf (pas d'index) dont la localisation est donnée par la variable système _foxcode.

La table FOXCODE, que l'on doit ouvrir par
`USE (_foxcode) ALIAS fcok IN 0 SHARED AGAIN`
(il ne faut pas ouvrir celle qui est dans le répertoire VFP9, ce n'est pas la bonne; on peut ouvrir la table même si l'intellisense est actif et il n'est pas nécessaire de fermer la 'copie' pour que ses données soient prises en compte) contient différent types de lignes. Seuls 2 types nous interessent :

1. le type 'S' contient un programme VFP (un Script) qui va traiter la ligne sur laquelle on se trouve. Le nom de ce script est dans le champ 'abbrev', le code lui-même se trouve dans le champ 'data'. Un script peut être utilisé par de nombreuses 'macros'; il se résume en fait en un calcul de variables que l'on va utiliser dans un TEXTMERGE() dont le paramètre est le champ data de la deuxième ligne qui nous intéresse ici : la ligne 'U'.

2. le type 'U' contient le nom de la 'macro' dans le champ 'abbrev', un texte à afficher à la place de l'abréviation et contenant des 'inclusions textmerge' placé dans le champ 'data', et un champ 'user' à notre disposition. On a une ligne 'U' par macro; chaque ligne 'U' peut appeler un script si on met le nom de ce script entre {} dans le champ 'cmd'.

et finalement, le fonctionnement est assez simple

commençons par un script : j'ai créé mon propre script en recopiant le script {stmthandler'} standard : j'ai ajouté plusieurs variables et surtout la possibilité de saisir 1 ou 2 chaines de caractères avec INPUTBOX en fonction d'éléments placés dans le champ USER de la ligne 'U'. L'appel du script se fait en lui passant une 'structure' en paramètre. La structure de cet objet est la suivante (j'ai mis en bleu, les propriétés que j'ai utilisées):

Property Name	*eValue*
Abbrev	Specifies contents of the Abbrev field.
Case	Specifies contents of the Case field.
Cmd	Specifies contents of the Cmd field.
CursorLocChar	Specifies a special character denoting the location to position the cursor after script runs. (Default character is a tilde (~).)
Data	Specifies contents of the Data field.
DefaultCase	Specifies default letter casing setting in the IntelliSense table as derived from the Version record, whose Type is set to "V".
Expanded	Specifies contents of the Expanded field.
Filename	Specifies the name of file being edited.
FullLine	Specifies full text from the line currently typed.
Icon	Specifies the icon to use with Items array.
Items	Specifies an array to use for populating a list box that displays following running a script. Requires ValueType set to "L". Items[1,1] – Text to display in list Items[1,2] – Value tip for item The only required element is the first element for each row in the Items array. By default, the array is sorted in ascending order so incremental seek operations can be performed. Users can use the ItemSort property to turn this functionality off and use a natural sort order.
ItemScript	Specifies the script to use with Items array.
ItemSort	Specifies whether to sort Items array (Default is set to .T.)
Location	Specifies type of editor to edit: 0 – Command Window 1 – Program 8 – Menu Editor 10 – Code Editor 12 – Stored Procedure
Menuitem	Specifies the menu item selected if user is running a script with ValueType set to "L". Can be used in a subsequent script.
ParamNum	Specifies parameter number of the function for script call made within a function.
Save	Specifies contents of the Save field.
Source	Specifies contents of the Source field.
Timestamp	Specifies contents of the Timestamp field.
Tip	Specifies contents of the Tip field.
Type	Specifies contents of the Type field.

UniqueId	Specifies contents of the UniqueId field.
User	Specifies contents of the User field.
UserTyped	Specifies the text the user typed. Does not include the activator key or any leading spaces or tabs. To include activator key, leading spaces or tabs, use FullLine property instead.
ValueTip	Specifies Quick Info tip to display when ValueType is set to "T".
ValueType	Specifies handler for action following script execution: L – Displays a drop-down list box populated from the Items array. V – Displays a list of values. T – Displays a Quick Info Tip window from ValueTip.

Les principales propriétés sont :

- **fullline** : la ligne sur lequel le curseur clignotant est et donc sur laquelle on va trouver l'abréviation qui vient d'être tapée
- **data** : le texte de la macro. C'est sur cette propriété que l'on va appliquer un TEXTMERGE qui donnera le résultat à imprimer à la place de l'abréviation.
- **location** nous indique dans quel type de document on est. On ne fait pas d'expansion dans la fenêtre de commande.
- **user** contient le champ 'user' de la lgne 'U' appelante. Ce champ est à la disposition du programmeur. J'y ai mis les textes des libellés des INPUTBOX.

Voici le code de mon script que j'ai appelé {stmthandlerjm} :

```
LPARAMETER oFoxcode
* ce code est une adaptation du code de la ligne 'S', 'stmthandler'
* j'y ajoute la saisie d'un texte dans une boîte INPUTBOX; ce code est
* rangé dans la variable locale lctexte que l'on peut utiliser dans
* le memo DATA de la ligne User de la 'macro' . Voir 'IFF'. On peut même
* saisir 2 textes dans 2 INPUTBOX.
* lnligne contient le nombre de lignes dans le champ USER (que l'on retrouve dans oFoxcode.user)
* ltuser(1) contient, ligne par ligne, le contenu de oFoxcode.user
* Si le memo contient 2 lignes (éventuellement + 1 de commentaire), on lance une saisie
*     la premiere ligne contient le texte à mettre juste au dessus de la zone de saisie
*     la deuxième ligne contient le titre de la fenêtre INPUTBOX
* Si le memo contient 4 lignes (éventuellement + 1 de commentaire), on lance deux saisies
*     la premiere ligne contient le texte à mettre juste au dessus de la 1ère zone de saisie
*     la deuxième ligne contient le titre de la 1ère fenêtre INPUTBOX
*     la troisième ligne contient le texte à mettre juste au dessus de la 2éme zone de saisie
*     la quatrème ligne contient le titre de la 2éme fenêtre INPUTBOX
* la variable lcdate contient la date du jour (DTOC(DATE()))
* la variable lcdatetime contient TTOC(DATETIME())

LOCAL lcLine, lcSpace, lnPos, lcCase, lcData, lctexte,  lctexte2, lnligne, lcdate, lcdatetime
LOCAL ARRAY ltuser(1)

lcLine = oFoxcode.FullLine  && ligne sur laquelle on travaille
IF oFoxcode.Location=0 OR GETWORDCOUNT(lcLine)#1
  * si on est dans la fenêtre de commande ou si l'abréviation n'est
```

```
  * pas seule, on ne fait rien.
  RETURN oFoxcode.UserTyped
ENDIF
* on calcul l'indentation courante (très important):
lnPos = ATC(ALLTRIM(oFoxcode.Abbrev),lcLine)
lcSpace = SUBSTR(lcLine,1,lnPos-1)
oFoxcode.valuetype = "V"

* Get case celà m'enquiquinait alors on ne traite plus la casse !
lcCase = oFoxCode.Case
IF EMPTY(ALLTRIM(lcCase))
        lcCase = oFoxCode.DefaultCase
        IF EMPTY(ALLTRIM(lcCase))
                lcCase = "M"
        ENDIF
ENDIF
lcdate = DTOC( DATE()) && peut être utilisé dans le TEXTMERGE
lcdatetime = TTOC( DATETIME()) && idem
lnligne = ALINES(ltuser, oFoxCode.user)
IF m.lnligne>=2
  * on saisit la première chaine
  lctexte = INPUTBOX( ltuser(1), ltuser(2))
  IF m.lnligne >=4
    * on saisit la deuxième chaine
    lctexte2 = INPUTBOX( ltuser(3), ltuser(4))
  ENDIF
ELSE
  lctexte = INPUTBOX("texte à ajouter: ", "Vous pouvez saisir un texte complémentaire")
ENDIF
* Process and return expanded control statement
*lcData = AdjustCase(oFoxCode.Data, m.lcCase) on ne change pas la casse
lcData = oFoxCode.Data  && contenu du champ data de la ligne 'U' appelante

* c'est la commande principale du script :
RETURN TEXTMERGE(m.lcData, .T.)
```

Regardons maintenant une macro compliquée : DEFC. Elle doit nous donner la structure de base d'une définition de classe. On donne le nom de la classe et le nom de la classe de base.

le champ abbrev contient 'DEFC' : lorsque l'on tapera 'defc' en début de ligne, l'expansion sera lancée.
le champ cmd contient {stmthandlerjm} : c'est donc ce script qui est lancé avec comme paramètre un objet qui reprend les données de la ligne 'U' 'DEFC' et d'autres informations utiles.
Le champ data contient :

```
DEFINE CLASS <<m.lctexte>> AS <<m.lctexte2>> && jme <<m.lcdate>>
<<m.lcSpace>>        *
<<m.lcSpace>>        ~
<<m.lcSpace>>
<<m.lcSpace>>        PROCEDURE Init
<<m.lcSpace>>
<<m.lcSpace>>        ENDPROC && Init
<<m.lcSpace>>
<<m.lcSpace>>        PROCEDURE Destroy
<<m.lcSpace>>
```

```
<<m.lcSpace>>		ENDPROC && Destroy
<<m.lcSpace>>
<<m.lcSpace>>		PROCEDURE Error( nError, cMethod, nLine)
<<m.lcSpace>>
<<m.lcSpace>>		ENDPROC && Error
<<m.lcSpace>>
<<m.lcSpace>>
ENDDEFINE && <<m.lctexte>>
```

Le contenu de ce champ est passé au script. Le script calcule les variables lctexte, lctexte2 et lcspace (l'indentation) puis fait un textmerge et renvoit le résultat qui est imprimé à la place de l'abbréviation que nous venons de taper.
Rappel : le ~ (fin de la troisième ligne) indique où doit être placé le curseur après l'expansion mais il y a un petit bug et le curseur est souvent placé un caractère avant le tilda.

Je vous joins une table foxcodejm.dbf que vous pouvez 'appender' à foxcode.dbf pour avoir immédiatement toutes les 'macros' que je viens de construire.

Comme d'habitude, ceci est un premier jet : toutes les critiques sont les bien-venues !

Jean à Grenoble
28 août 2008